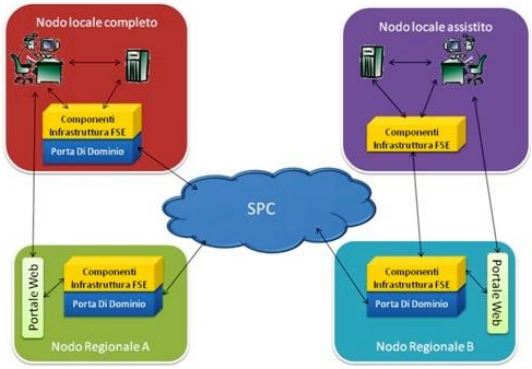
Presidenza del Consiglio dei Ministri
Dipartimento per la digitalizzazione della Pubblica Amministrazione e l'innovazione tecnologica

Consiglio Nazionale delle Ricerche
Dipartimento delle Tecnologie dell'Informazione e delle Comunicazioni

Progetto OpenInFSE

"Realizzazione di un'infrastruttura operativa a supporto dell'interoperabilità delle soluzioni territoriali di fascicolo sanitario elettronico nel contesto del sistema pubblico di connettività"



InFSE:

Infrastruttura tecnologica del Fascicolo Sanitario Elettronico

Note di rilascio

versione 1.3

eGov 2012 - Obiettivo Salute

Giugno 2012

Indice

1 Obiettivi del documento

Il presente documento ha lo scopo di presentare le componenti software per la realizzazione di un'infrastruttura tecnologica del Fascicolo Sanitario Elettroncio (FSE), in maniera conforme alle linee guida InFSE, sviluppate nell'ambito del progetto OpenInFSE.

In primo luogo, sono descritte le tecnologie utilizzate per la realizzazione delle componenti.

Successivamente, il documento descrive i servizi di ogni componente software rilasciata, i quali sono stati implementati come Web Services.

Le componenti software ed i servizi rilasciati sono i seguenti:

1. Componente *Interfaccia di Accesso*
   1.1. Servizio *IDocument*
   1.2. Servizio *IEntry*
   1.3. Servizio *IRegistryFederation*
   1.4. Servizio *IEvent*
   1.5. Servizio *IBroker Federation*
2. Componente *Registro Indice Federato*
   2.1. Servizio *IMetadataMgt*
   2.2. Servizio *IQueryMgt*
   2.3. Servizio *IEventMgt*
   2.4. Servizio *IRegistryFederationMgt*
3. Componente *Gestore dei Documenti*
   3.1. Servizio *IDocumentMgt*
4. Componente *Gestore Gerarchico degli Eventi*
   4.1. Servizio *IBrokerFederationMgt*
   4.2. Servizio *INotificationBrokerMgt*
   4.3. Servizio *ISubscriptionMgt*
   4.4. Servizio *IBrokerFederationMgt*
   4.5. Servizio *IConsumer*
5. Componente *Gestore delle Politiche di Accesso*

Per ogni servizio, sono dettagliate le funzionalità implementate ed i parametri di configurazione da impostare per un corretto utilizzo.

Il rilascio comprende i codici binari, in formato WAR, oltre a semplici applicazioni client di test. Si noti che, per utilizzare correttamente i file WAR, occorre impostare opportuni file di configurazione inclusi negli archivi, come descritto in questo documento, ed effettuare il deploy dei servizi (ad es. trasferendo i file WAR all'interno della directory *webapps* di Apache Tomcat).

1.1 Versioni del documento

Titolo	OpenInFSE - Note di rilascio
Data	15/06/2012
Versione	1.3
Stato	DEF

Storia delle principali revisioni:

Versione	Stato	Data	Descrizione Modifica
1.3	DEF	15/06/2012	Rilascio delle componenti Gestore Gestore delle Politiche di Accesso e Gerarchico degli Eventi.
1.2	Beta release	02/04/2012	Implementazione di librerie per l'interazione con registri/repository. Utilizzo di riferimenti simbolici in luogo di URL. Possibilità di indirizzare la richiesta ad uno specifico registro. Gestione di documenti in formato PDF/A. Invocazione dei servizi con modalità POST.
1.1	Beta release	15/11/2011	Correzione di bug minori.
1.0	Alpha release	15/07/2011	Prima versione rilasciata.

1.2 Unità operative del CNR coinvolte nella realizzazione

- ICAR, sede di Napoli;
- ICAR, sede di Palermo;
- ICAR, sede di Cosenza;
- IIT, sede di Pisa;
- URT-DSP "Sistemi di Indicizzazione e Classificazione", sede di Cosenza.

2 Tecnologie utilizzate

Questa sezione descrive le tecnologie e le piattaforme utilizzate per l'implementazione ed il testing dei servizi.

- **Sistemi operativi testati:** Microsoft Windows 7, Mac OS X 10.6.8, Linux (distribuzioni Red Hat e Ubuntu)
- **Java Virtual Machine:** J2SE 1.6
- **Ambiente di sviluppo:** NetBeans IDE 7.0
- **Framework di sviluppo per Web Services:** JAX-WS Metro 2.0
- **Application server:** Apache Tomcat 7
- **Registry/Respository:** freebXML Registry OMAR v3.1
- **Middleware Publish/Subscribe:** Apache Active MQ 5.5.1

La scelta di un registry e di un repository di riferimento da utilizzare per l'espletamento delle funzionalità delle componenti è ricaduta sull'implementazione open source di ebXML OMAR, la quale funge da back-end rispetto ai servizi per la gestione dei metadati e dei documenti. Tuttavia, si noti che è possibile utilizzare registry e repository differenti, realizzando opportuni wrapper.

Il downlaod di OMAR può essere effettuato all'indirizzo:
http://sourceforge.net/projects/ebxmlrr/files/freebxml-registry/3.1/.

Il sistema OMAR ha le seguenti dipendenze:

- Java Web Services Developer Pack (JWSDP);
- Apache Tomcat.

In particolare, OMAR è stato testato con JWSDP 1.6 e Apache Tomcat 5.5.

Si noti che le librerie del sistema open source OMAR incluse nei progetti e nei file WAR devono essere sostituite con quelle del registro utilizzato, come descritto dettagliatamente nel manuale di installazione.

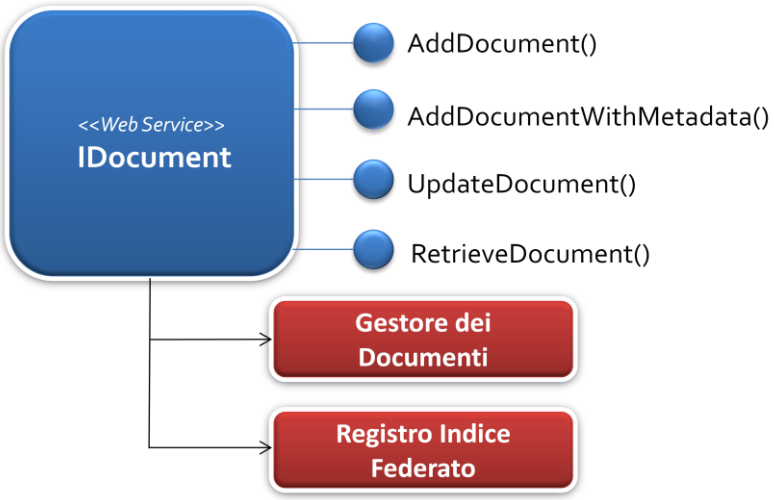
3 Interfaccia di Accesso

In questa sezione sono descritti i seguenti servizi della componente *Interfaccia di Accesso*:

- *IDocument*;
- *IEntry*;
- *IRegistryFederation;*
- *IEvent*;
- *IBrokerFederation*.

3.1 Servizio IDocument

IDocument è il servizio per l'accesso ed il caricamento di documenti nel FSE. Esso invoca i servizi *IMetadataMgt* della componente *Registro Indice Federato*, e *IDocumentMgt* della componente *Gestore dei Documenti*.



Come descritto di seguito, questo servizio può interagire anche il servizio *IDocument* dislocato presso un altro nodo regionale, per il recupero di un documento extra-regionale.

Di seguito sono descritti tutti i metodi, con i parametri di ingresso/uscita, del servizio *IDocument*.

Servizio: **IDocument**	
Metodo: ***AddDocumentWithMetadata***	È il metodo che permette l'inserimento di un nuovo documento e dei relativi metadati.
Parametro IN: Owner	Riferimento usato per identificare il servizio *IDocumentMgt* dove memorizzare il documento.
Parametro IN: Status	Parametro che permette di specificare lo stato iniziale del documento.
Parametro IN: Metadata	Metadati da memorizzare in un registro conforme alle specifiche ebXML RegRep 3.0.
Parametro IN: DocumentObj	Documento in formato HL7-CDA2 con, opzionalmente, il documento in formato PDF/A o lo stylesheet in formato XLST.
Parametro OUT: DocumentID	Insieme di identificativi del documento richiesto comprendenti il riferimento al servizio *IDocument* del nodo regionale, il riferimento al servizio *IDocumentMgt* dove reperire il documento e l'identificativo del documento. Queste informazioni sono memorizzate nei registri.

Servizio: **IDocument**	
Metodo: ***AddDocument***	È il metodo che permette l'inserimento di un nuovo documento.
Parametro IN: Owner	Riferimento usato per identificare il servizio *IDocumentMgt* dove memorizzare il documento.
Parametro IN: Status	Parametro che permette di specificare lo stato iniziale del documento.
Parametro IN: DocumentObj	Documento in formato HL7-CDA2 con, opzionalmente, il documento in formato PDF/A o lo stylesheet in formato XLST. Dal documento in formato HL7-CDA2 sono estratti automaticamente i metadati e memorizzati nel registro invocando il servizio *IMetadataMgt* noto.
Parametro OUT: DocumentID	Insieme di identificativi del documento richiesto comprendenti il riferimento al servizio *IDocument* del nodo regionale, il riferimento al servizio *IDocumentMgt* dove reperire il documento e l'identificativo del documento. Queste informazioni sono memorizzate nei registri.

Servizio: **IDocument**	
Metodo: ***UpdateDocument***	È il metodo che permette l'aggiornamento di un documento pre-esistente.
Parametro IN: Owner	Riferimento usato per identificare il servizio *IDocumentMgt* dove memorizzare il documento.
Parametro IN: Status	Nuovo stato del documento.
Parametro IN: DocumentObj	Nuova versione del documento (in formato HL7-CDA2 con, opzionalmente, il documento in formato PDF/A o lo stylesheet in formato XLST). Dal documento in formato HL7-CDA2 sono estratti automaticamente i metadati e memorizzati nel registro invocando il servizio *IMetadataMgt* noto.
Parametro IN: DocumentID	Insieme di identificativi del documento richiesto comprendenti il riferimento al servizio *IDocument* del nodo regionale, il riferimento al servizio *IDocumentMgt* dove memorizzare il documento e l'identificativo del documento. Queste informazioni sono presenti nei registri.
Parametro IN: Version	Indica la versione del documento che si sta aggiornando.

Servizio: **IDocument**	
Metodo: ***RetrieveDocument***	È il metodo che permette l'acquisizione di un documento. Se questo servizio è dislocato presso il nodo locale, esso propaga la richiesta al servizio corrispondente presso il nodo regionale. Il servizio del nodo regionale verifica se il documento è ubicato presso il proprio dominio o meno. Nel primo caso, invoca il servizio *IDocumentMgt* specificato, nel secondo propaga la richiesta al servizio *IDocument* di un altro nodo regionale.
Parametro IN: DocumentID	Insieme di identificativi del documento richiesto comprendenti il riferimento al servizio *IDocument* del nodo regionale, il riferimento al servizio *IDocumentMgt* dove reperire il documento e l'identificativo del documento. Queste informazioni sono presenti nei registri.
Parametro OUT: DocumentObj	Documento in formato HL7-CDA2 con, opzionalmente, il documento in formato PDF/A o lo stylesheet in formato XLST.

3.1.1 Scenari di interazione

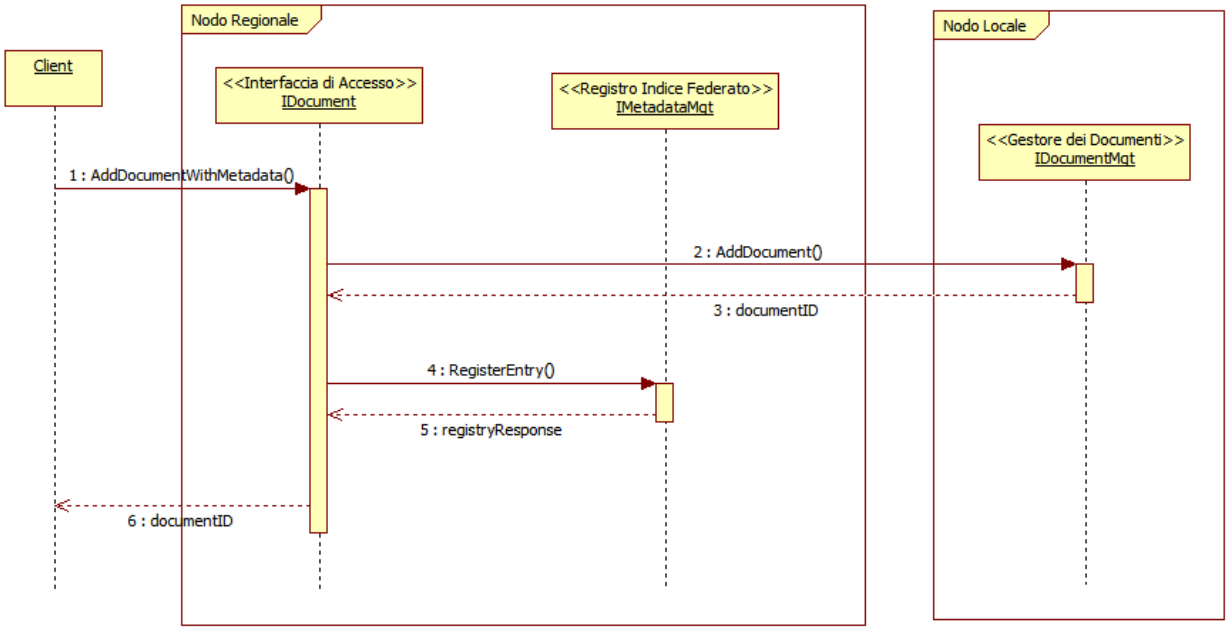
Questa sezione descrive i possibili scenari di interazione per la gestione dei documenti attraverso il servizio *IDocument*. In particolare, si distinguono i seguenti casi:

- caricamento di un documento in un repository sito presso uno specifico dominio regionale;
- aggiornamento di un documento in un repository sito presso uno specifico dominio regionale;
- recupero di un documento da un repository sito presso lo stesso dominio regionale in cui è stata effettuata la richiesta;
- recupero di un documento da un repository sito presso un dominio regionale differente da quello in cui è stata effettuata la richiesta.

3.1.1.1 Caricamento di un documento

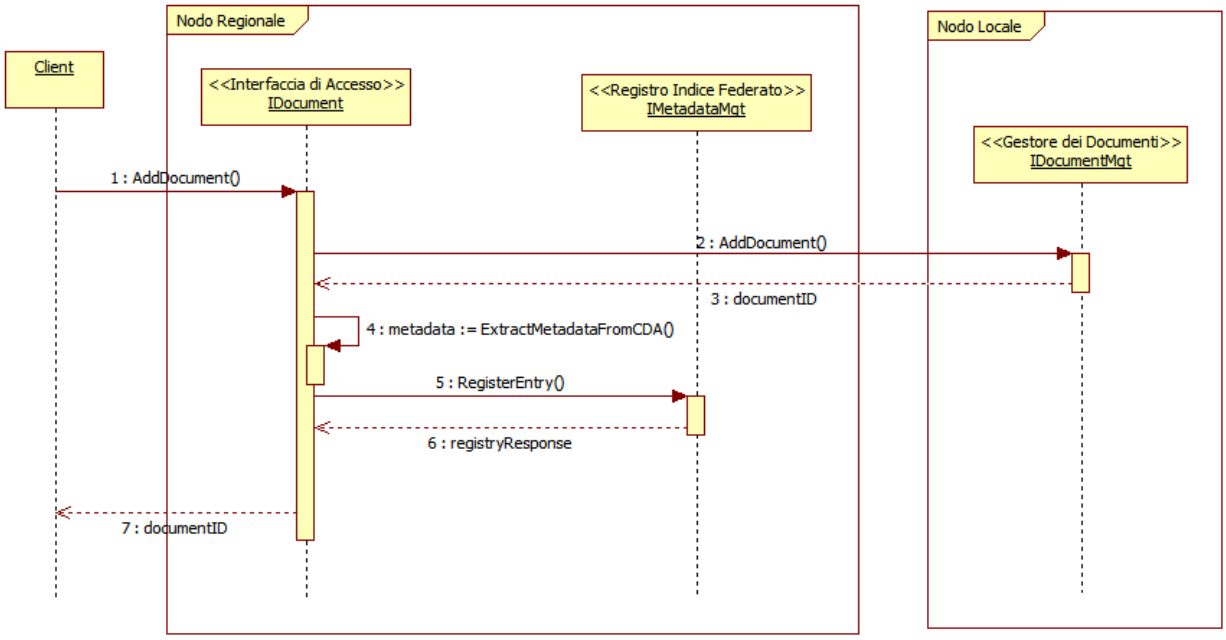
Questo scenario può essere realizzato invocando le operazioni *AddDocumentWithMetadata()*, che richiede in input i metadati da memorizzare in un registro, o *AddDocument()*, che estrae automaticamente i metadati dal documento HL7-CDA da caricare. Gli scenari fanno riferimento al caso in cui il servizio *IDocument* sia dislocato presso il nodo regionale, ma il comportamento del servizio è analogo nel caso in cui il servizio sia dislocato presso un nodo locale. Non si prevedono scenari per la memorizzazione di un documento presso domini extra-regionali.

AddDocumentWithMetadata()



1. Il client invoca l'operazione *AddDocumentWithMetadata()*, indicando dove memorizzare il documento (riferimento al servizio *IDocumentMgt*), lo stato iniziale del documento, i metadati ed il documento da memorizzare;
2. il servizio *IDocument* invia il documento al servizio *IDocumentMgt* specificato (la cui URL è ottenuta risolvendo il riferimento indicato);
3. il servizio *IDocumentMgt* memorizza il documento in un repository e restituisce l'identificativo del documento;
4. il servizio *IDocument* invia i metadati specificati (tra cui il riferimento al servizio *IDocument* del nodo regionale, il riferimento al servizio *IDocumentMgt* e l'identificativo del documento) al servizio *IMetadataMgt*;
5. il servizio *IMetadataMgt* memorizza in metadati in un registro e restituisce un parametro di ritorno;
6. il servizio *IDocument* restituisce gli identificativi al client.

AddDocument()

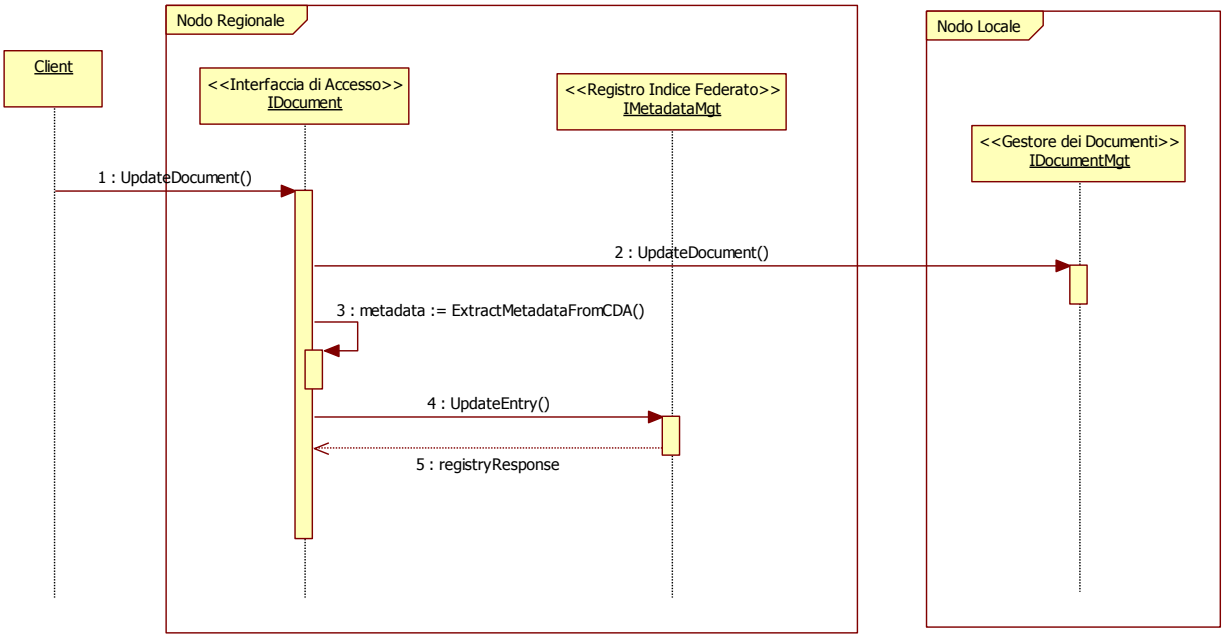


1. Il client invoca l'operazione *AddDocument()*, indicando dove memorizzare il documento (riferimento al servizio *IDocumentMgt*), lo stato iniziale del documento ed il documento da memorizzare;
2. il servizio *IDocument* invia il documento al servizio *IDocumentMgt* specificato (la cui URL è ottenuta risolvendo il riferimento indicato);
3. il servizio *IDocumentMgt* memorizza il documento in un repository e restituisce l'identificativo del documento;
4. il servizio *IDocument* estrae i metadati dal documento;
5. il servizio *IDocument* invia i metadati estratti (tra cui il riferimento al servizio *IDocument* del nodo regionale, il riferimento al servizio *IDocumentMgt* e l'identificativo del documento) al servizio *IMetadataMgt*;
6. il servizio *IMetadataMgt* memorizza i metadati in un registro e restituisce un parametro di ritorno;
7. il servizio *IDocument* restituisce gli identificativi al client.

3.1.1.2 Aggiornamento di un documento

Questo scenario può essere realizzato invocando l'operazione *UpdateDocument()*. Lo scenario fa riferimento al caso in cui il servizio *IDocument* sia dislocato presso il nodo regionale, ma il comportamento del servizio è analogo nel caso in cui il servizio sia dislocato presso un nodo locale. Non si prevedono scenari per l'aggiornamento di un documento presso domini extra-regionali.

UpdateDocument()



1. Il client invoca l'operazione *UpdateDocument()*, indicando dove memorizzare il documento (riferimento al servizio *IDocumentMgt*), lo stato iniziale del documento, il documento da memorizzare, l'identificativo del documento da sostituire (comprendente il riferimento al servizio *IDocument* del nodo regionale, il riferimento al servizio *IDocumentMgt* e l'identificativo del documento), che può essere ottenuto interrogando un registro, e la versione;
2. il servizio *IDocument* invia il documento al servizio *IDocumentMgt* specificato (la cui URL è ottenuta risolvendo il riferimento indicato), il quale provvede a memorizzarlo in un repository;
3. il servizio *IDocument* estrae i metadati dal documento;
4. il servizio *IDocument* invia i metadati estratti (tra cui il riferimento al servizio *IDocument* del nodo regionale, il riferimento al servizio *IDocumentMgt* e l'identificativo del documento) al servizio *IMetadataMgt*;
5. il servizio *IMetadataMgt* memorizza i metadati in un registro e restituisce un parametro di ritorno.

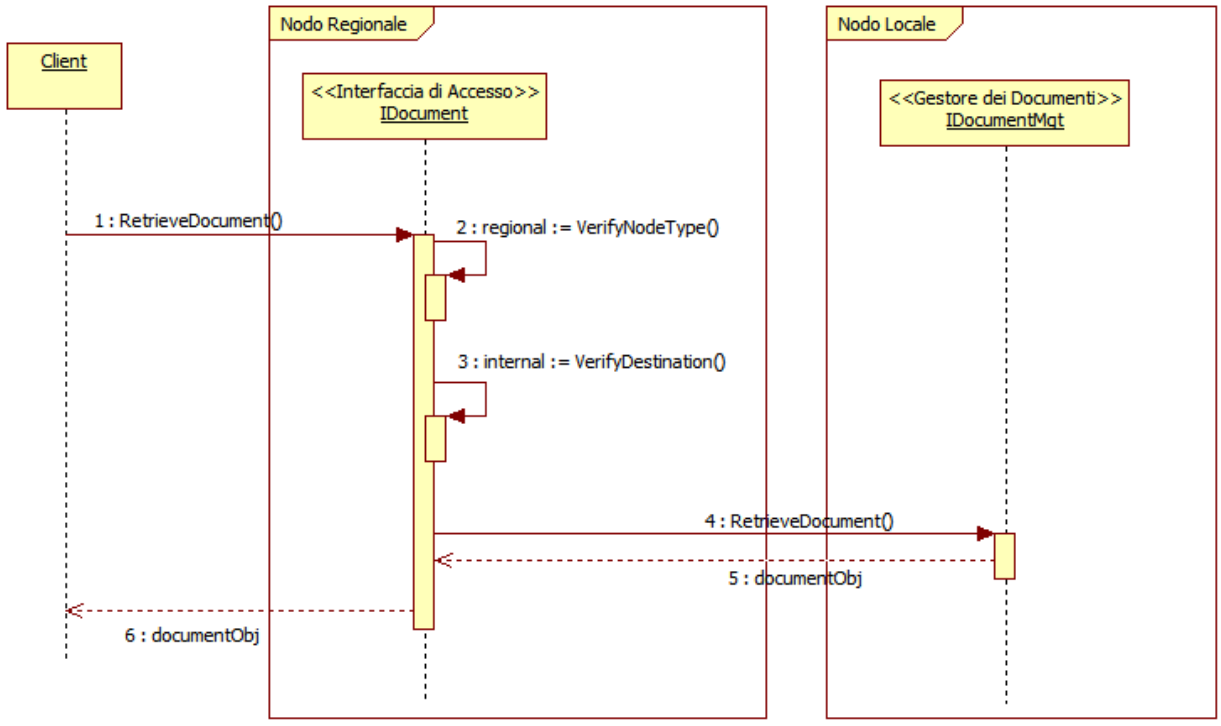
3.1.1.3 Recupero di un documento dal dominio regionale

Questo scenario può essere realizzato invocando l'operazione *RetrieveDocument()*. Il recupero di un documento da un repository sito presso lo stesso dominio regionale in cui è stata effettuata la richiesta prevede due possibili scenari:

- la richiesta viene inoltrata al servizio *IDocument* del nodo regionale;
- la richiesta è inviata al servizio *IDocument* di un nodo locale (ad es. una struttura sanitaria).

IDocument Regionale

RetrieveDocument()

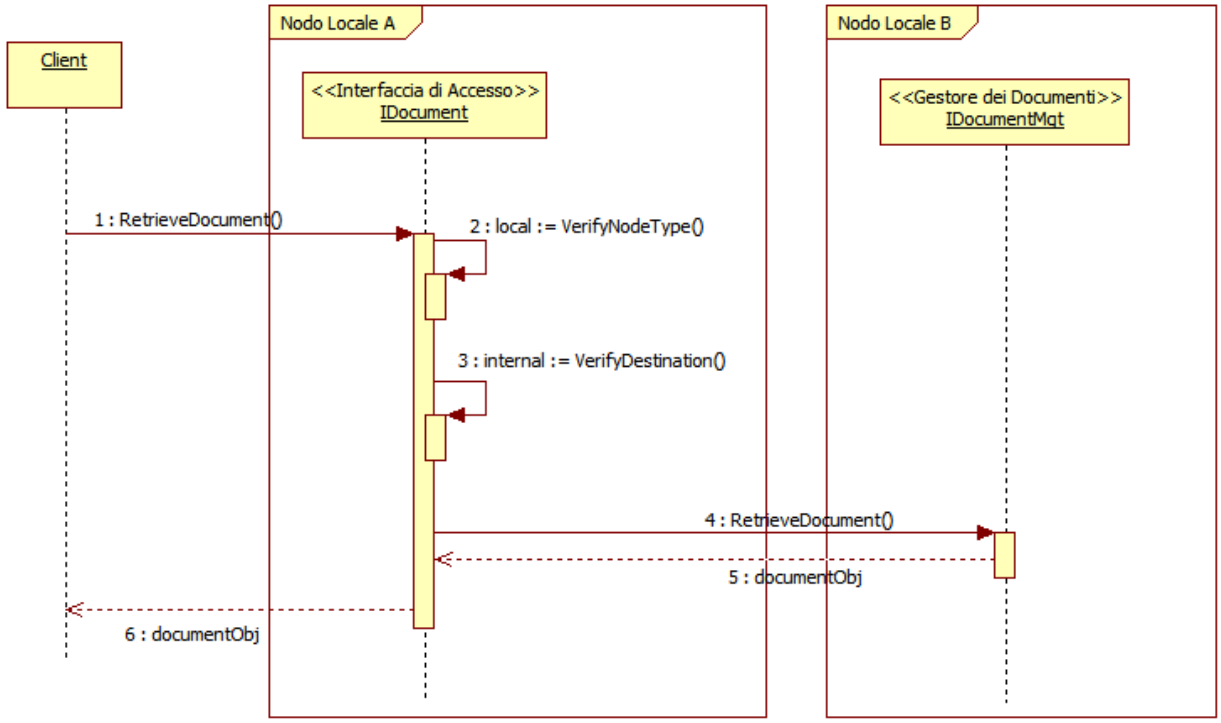


1. Il client invoca l'operazione *RetrieveDocument()*, indicando l'identificativo del documento da recuperare (comprendente il riferimento al servizio *IDocument* del nodo regionale, il riferimento al servizio *IDocumentMgt* e l'identificativo del documento), che può essere ottenuto interrogando un registro;
2. il servizio *IDocument* verifica se è dislocato presso un nodo regionale o locale;
3. il servizio *IDocument* verifica se il documento è sito nel dominio regionale o in un altro dominio (confrontando il riferimento al servizio *IDocument* del nodo regionale specificato con il proprio);
4. il servizio *IDocument* invoca il servizio *IDocumentMgt* specificato (la cui URL è ottenuta risolvendo il riferimento indicato);

5. il servizio *IDocumentMgt* recupera il documento da un repository e lo restituisce al servizio *IDocument*;
6. il servizio *IDocument* restituisce al client il documento in formato binario.

IDocument Locale

RetrieveDocument()



1. Il client invoca l'operazione *RetrieveDocument()*, indicando l'identificativo del documento da recuperare (comprendente il riferimento al servizio *IDocument* del nodo regionale, il riferimento al servizio *IDocumentMgt* e l'identificativo del documento) che può essere ottenuto interrogando un registro;
2. il servizio *IDocument* verifica se è dislocato presso un nodo regionale o locale;
3. il servizio *IDocument* verifica se il documento è sito nel dominio regionale o in un altro dominio (confrontando il riferimento al servizio *IDocument* del nodo regionale specificato con quello al servizio *IDocument* del proprio nodo);
4. il servizio *IDocument* invoca il servizio *IDocumentMgt* specificato (la cui URL è ottenuta risolvendo il riferimento indicato);
5. il servizio *IDocumentMgt* recupera il documento da un repository e lo restituisce al servizio *IDocument*;
6. il servizio *IDocument* restituisce al client il documento in formato binario.

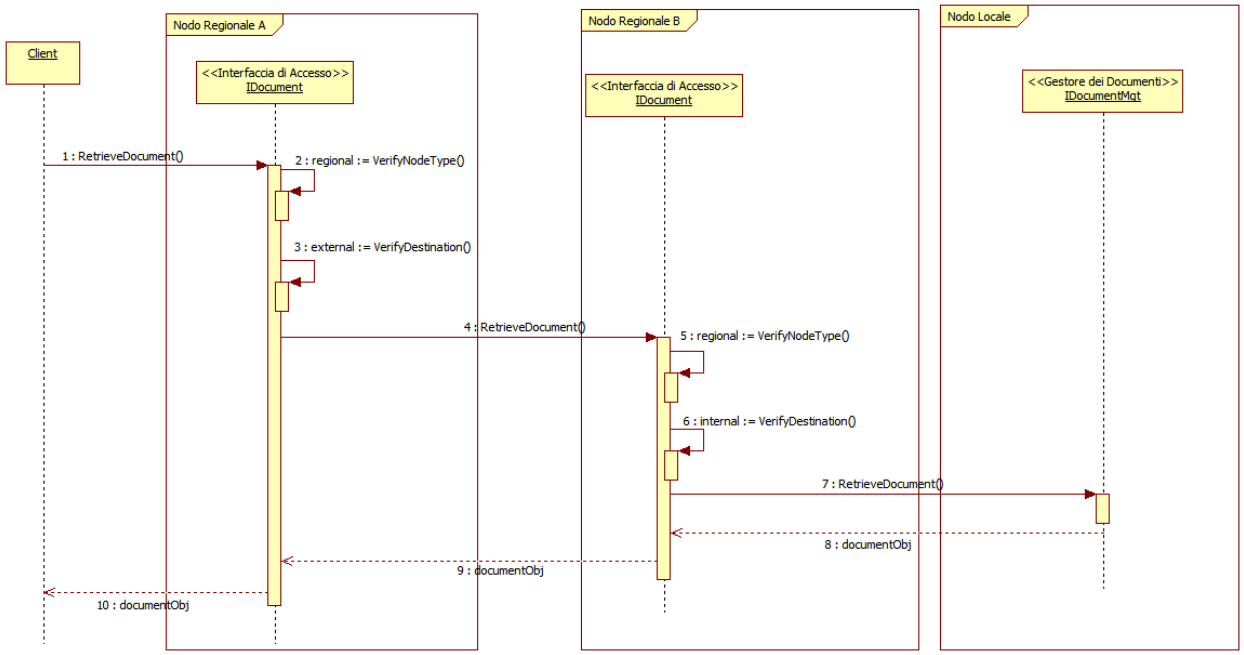
3.1.1.4 Recupero di un documento da un dominio extra-regionale

Questo scenario può essere realizzato invocando l'operazione *RetrieveDocument()*. Il recupero di un documento da un repository sito presso il Nodo Regionale B da parte di un client del Nodo Regionale A prevede due possibili scenari:

- la richiesta viene inoltrata al servizio *IDocument* dislocato presso il proprio nodo regionale;
- la richiesta è inviata al servizio *IDocument* di un nodo locale, dislocato presso una struttura sanitaria.

IDocument Regionale

RetrieveDocument()



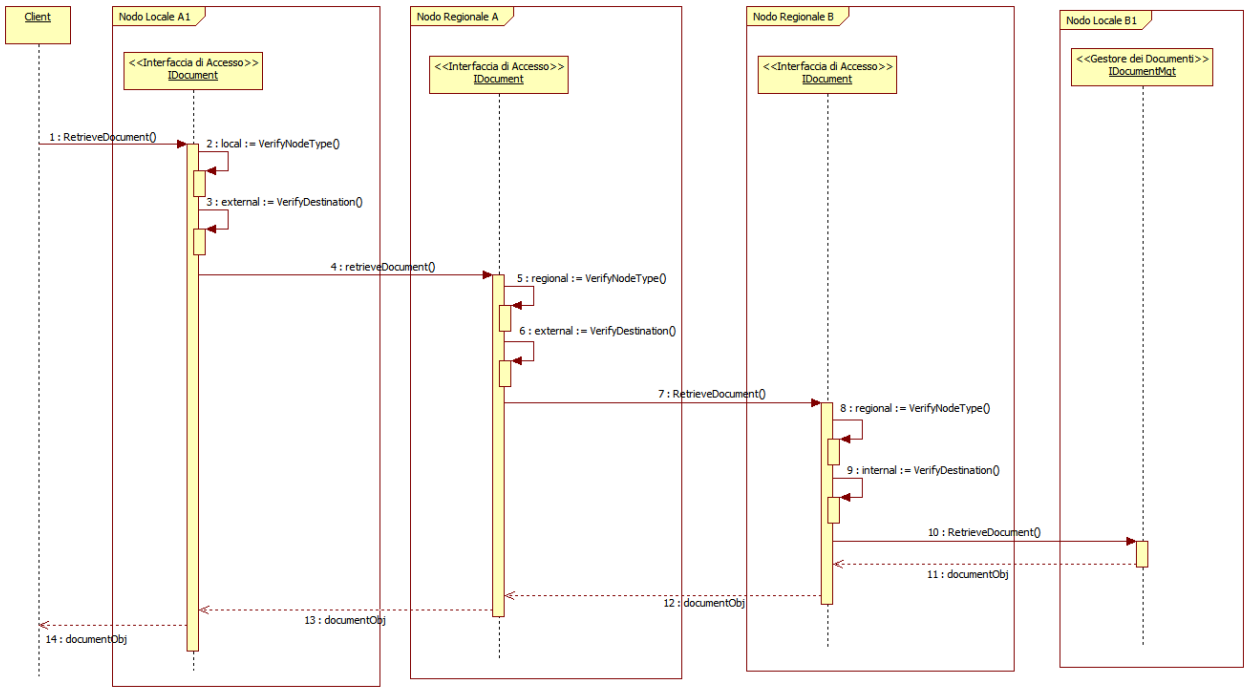
1. Il client invoca l'operazione *RetrieveDocument()* del servizio *IDocument* dislocato presso il nodo regionale A, indicando l'identificativo del documento da recuperare (comprendente il riferimento al servizio *IDocument* del nodo regionale, il riferimento al servizio *IDocumentMgt* e l'identificativo del documento), che può essere ottenuto interrogando un registro;
2. il servizio *IDocument* invocato verifica se è dislocato presso un nodo regionale o locale;
3. il servizio *IDocument* del nodo regionale A verifica se il documento è disponibile nel dominio regionale A o in un altro dominio (confrontando il riferimento al servizio *IDocument* del nodo regionale specificato con il proprio);
4. il servizio *IDocument* del nodo regionale A invoca il servizio *IDocument* del nodo regionale B specificato (la cui URL è ottenuta risolvendo il riferimento

indicato);

5. il servizio *IDocument* del nodo regionale B verifica se è dislocato presso un nodo regionale o locale;
6. il servizio *IDocument* del nodo regionale B verifica se il documento è sito nel dominio regionale B (confrontando il riferimento del servizio *IDocument* del nodo regionale specificato con il proprio);
7. il servizio *IDocument* del nodo regionale B invoca il servizio *IDocumentMgt* specificato (la cui URL è ottenuta risolvendo il riferimento indicato);
8. il servizio *IDocumentMgt* recupera il documento da un repository e lo restituisce al servizio *IDocument* del nodo regionale B;
9. il servizio *IDocument* del nodo regionale B restituisce al servizio *IDocument* del nodo regionale A il documento in formato binario;
10. il servizio *IDocument* del nodo regionale A restituisce al client il documento in formato binario.

IDocument Locale

RetrieveDocument()

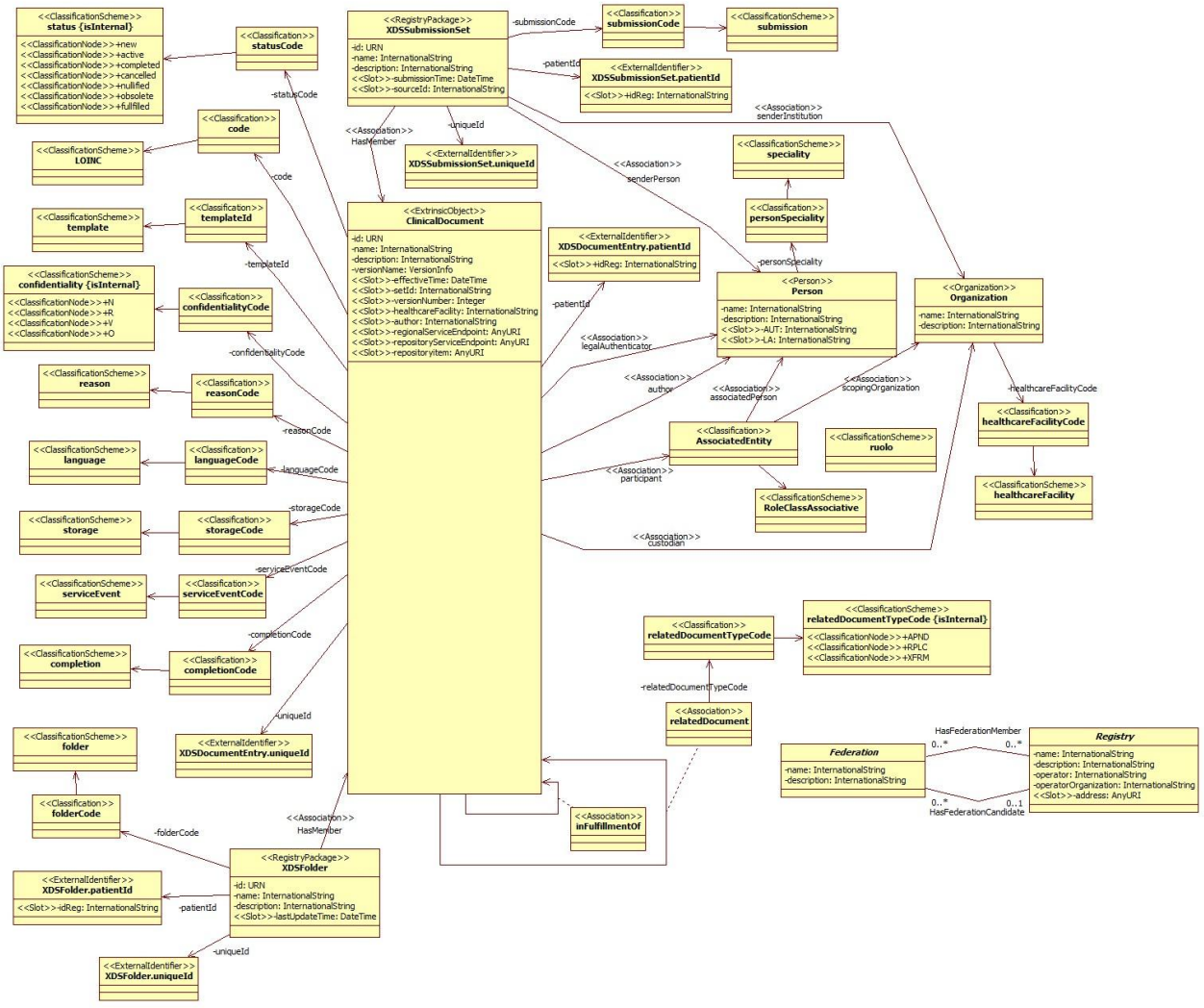


1. Il client invoca l'operazione *RetrieveDocument()* del servizio *IDocument* dislocato presso un nodo locale, indicando l'identificativo del documento da recuperare (comprendente il riferimento al servizio *IDocument* del nodo regionale, il riferimento al servizio *IDocumentMgt* e l'identificativo del documento), che può essere ottenuto interrogando un registro;
2. il servizio *IDocument* verifica se è dislocato presso un nodo regionale o locale;
3. il servizio *IDocument* del nodo locale verifica se il documento è sito nel dominio regionale o in un altro dominio (confrontando il riferimento al servizio *IDocument* del nodo regionale specificato con il riferimento al servizio *IDocument* del proprio nodo regionale);
4. il servizio *IDocument* del nodo locale invoca il servizio *IDocument* del proprio nodo regionale A;
5. il servizio *IDocument* verifica se è dislocato presso un nodo regionale o locale;
6. il servizio *IDocument* del nodo regionale A verifica se il documento è sito nel dominio regionale A o in un altro dominio (confrontando il riferimento al servizio *IDocument* del nodo regionale specificato con il proprio);
7. il servizio *IDocument* del nodo regionale A invoca il servizio *IDocument* del nodo regionale B specificato (la cui URL è ottenuta risolvendo il riferimento indicato);
8. il servizio *IDocument* invocato verifica se è dislocato presso un nodo regionale o locale;

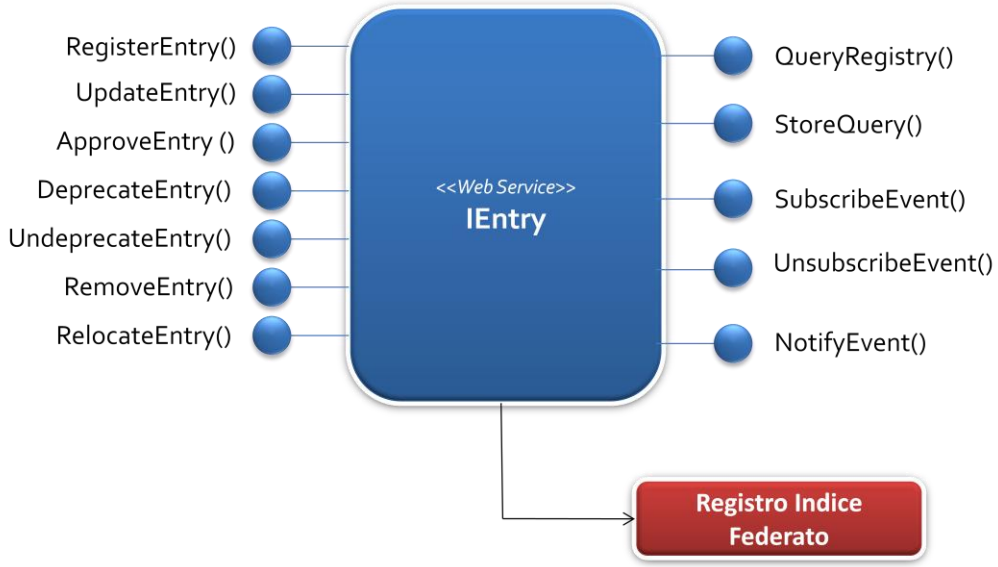
9. il servizio *IDocument* del nodo regionale B verifica se il documento è sito nel dominio regionale B (confrontando il riferimento al servizio *IDocument* del nodo regionale specificato con il proprio);
10. il servizio *IDocument* del nodo regionale B invoca il servizio *IDocumentMgt* specificato (la cui URL è ottenuta risolvendo il riferimento indicato);
11. il servizio *IDocumentMgt* recupera il documento da un repository e lo restituisce al servizio *IDocument* del nodo regionale B;
12. il servizio *IDocument* del nodo regionale B restituisce al servizio *IDocument* del nodo regionale A il documento in formato binario;
13. il servizio *IDocument* del nodo regionale A restituisce al servizio *IDocument* del nodo locale il documento in formato binario.
14. il servizio *IDocument* del nodo locale restituisce al client il documento in formato binario.

3.2 Servizio IEntry

IEntry è il servizio per la gestione dei metadati nei registri indice. Il modello dei metadati implementto è mostrato nella prossima figura. Si noti che è possibile utilizzare un dataset ridotto di metadati per l'indicizzazione dei documenti sanitari disponibili nei repository.



Il servizio *IEntry* funge da proxy rispetto ai servizi *IMetadataMgt*, *IQueryMgt* e *IEventMgt* della componente *Registro Indice Federato*.



Di seguito sono descritti tutti i metodi, con i parametri di ingresso/uscita, del servizio *IEntry*.

Servizio: **IEntry**	
Metodo: ***RegisterEntry***	É l'operazione che permette la registrazione di nuovi metadati all'interno di un registro.
Parametro IN: SubmitObjectsRequest	É il parametro che contiene i metadati da inserire all'interno del registro in maniera conforme alle specifiche ebXML RegRep 3.0.
Parametro OUT: RegistryResponse	É il parametro che contiene la risposta del registro.

Servizio: **IEntry**	
Metodo: ***UpdateEntry***	É l'operazione che consente di aggiornare i metadati esistenti in un registro.
Parametro IN: UpdateObjectsRequest	É il parametro che contiene i metadati da utilizzare per l'aggiornamento.
Parametro OUT: RegistryResponse	É il parametro che contiene la risposta del registro.

Servizio: **IEntry**	
Metodo: ***ApproveEntry***	É l'operazione che consente di approvare metadati esistenti in un registro.
Parametro IN: ApproveObjectsRequest	É il parametro che specifica i criteri da utilizzare per l'approvazione dei metadati.
Parametro OUT: RegistryResponse	É il parametro che contiene la risposta del registro.

Servizio: **IEntry**	
Metodo: ***DeprecateEntry***	É l'operazione che consente di invalidare metadati esistenti in un registro.
Parametro IN: DeprecateObjectsRequest	É il parametro che specifica i criteri da utilizzare per l'invalidazione dei metadati.
Parametro OUT: RegistryResponse	É il parametro che contiene la risposta del registro.

Servizio: **IEntry**	
Metodo: ***UndeprecateEntry***	É l'operazione che consente di validare metadati precedentemente dichiarati invalidi in un registro.
Parametro IN: UndeprecateObjectsRequest	É il parametro che specifica i criteri da utilizzare per la rivalidazione dei metadati.
Parametro OUT: RegistryResponse	É il parametro che contiene la risposta del registro.

Servizio: **IEntry**	
Metodo: ***RemoveEntry***	É l'operazione che consente di rimuovere metadati esistenti in un registro.
Parametro IN: RemoveObjectsRequest	É il parametro che specifica i criteri da utilizzare per la rimozione dei metadati.
Parametro OUT: RegistryResponse	É il parametro che contiene la risposta del registro.

Servizio: **IEntry**	
Metodo: ***RelocateEntry***	É l'operazione che consente di rilocare metadati da un registro ad un altro.
Parametro IN: RelocateObjectsRequest	É il parametro che specifica i criteri da utilizzare per la rilocazione dei metadati.
Parametro OUT: RegistryResponse	É il parametro che contiene la risposta del registro.

Servizio: **IEntry**	
Metodo: ***QueryRegistry***	É l'operazione che consente di sottoporre query locali o federate ai registri e di ottenere i risultati.
Parametro IN: AdhocQueryRequest	É il parametro che specifica i criteri da utilizzare per la query.
Parametro OUT: AdhocQueryResponse	É il parametro che contiene la risposta del registro invocato.

Servizio: **IEntry**	
Metodo: ***StoreQuery***	É l'operazione che consente di memorizzare stored query all'interno di un registro.
Parametro IN: SubmitObjectsRequest	É il parametro che specifica la query da memorizzare.
Parametro OUT: RegistryResponse	É il parametro che contiene la risposta del registro.

Servizio: **IEntry**	
Metodo: ***SubscribeEvent***	É l'operazione che consente di sottoporre sottoscrizioni ad eventi di interesse.
Parametro IN: SubmitObjectsRequest	É il parametro che specifica gli eventi di interesse sulla base di criteri di selezione.
Parametro OUT: RegistryResponse	É il parametro che contiene la risposta del registro.

Servizio: **IEntry**	
Metodo: ***UnsubscribeEvent***	É l'operazione che consente di rimuovere sottoscrizioni.
Parametro IN: RemoveObjectsRequest	É il parametro che specifica le sottoscrizioni da rimuovere.
Parametro OUT: RegistryResponse	É il parametro che contiene la risposta del registro.

Servizio: **IEntry**	
Metodo: ***NotifyEvent***	É l'operazione che consente di notificare eventi di interesse.
Parametro IN: Notification	É il parametro che trasporta il contenuto dell'evento.

3.2.1 Scenari di interazione

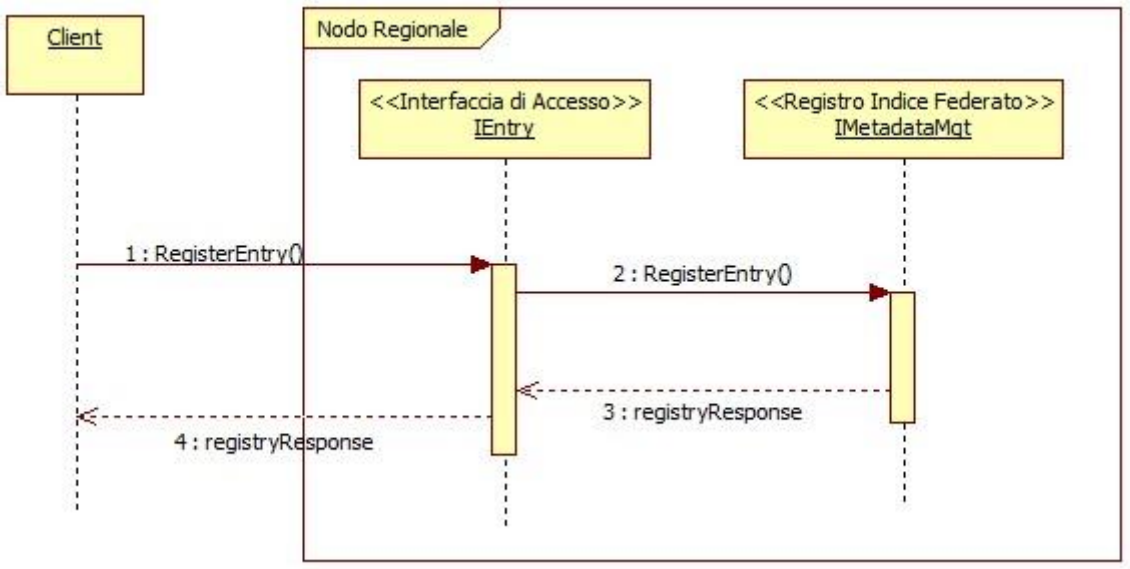
Questa sezione descrive i possibili scenari di interazione per la gestione dei metadati attraverso il servizio *IEntry*. In particolare, si distinguono i seguenti casi:

- gestione dei metadati;
- gestione delle query;
- gestione delle sottoscrizioni/notifiche di eventi.

3.2.1.1 Gestione dei metadati

La gestione dei metadati è effettuata mediante le operazioni *RegisterEntry()*, *UpdateEntry()*, *ApproveEntry()*, *DeprecateEntry()*, *UndeprecateEntry*(), *RemoveEntry()* e *RelocateEntry()*. Gli scenari di interazione tra le componenti sono simili per tutte le operazioni, eccezion fatta per l'operazione *RelocateEntry()*. Pertanto, di seguito sono mostrati due possibili scenari: la registrazione di nuovi metadati mediante l'operazione *RegisterEntry()* e la rilocazione di metadati da un registro ad un altro mediante l'operazione *RelocateEntry()*.

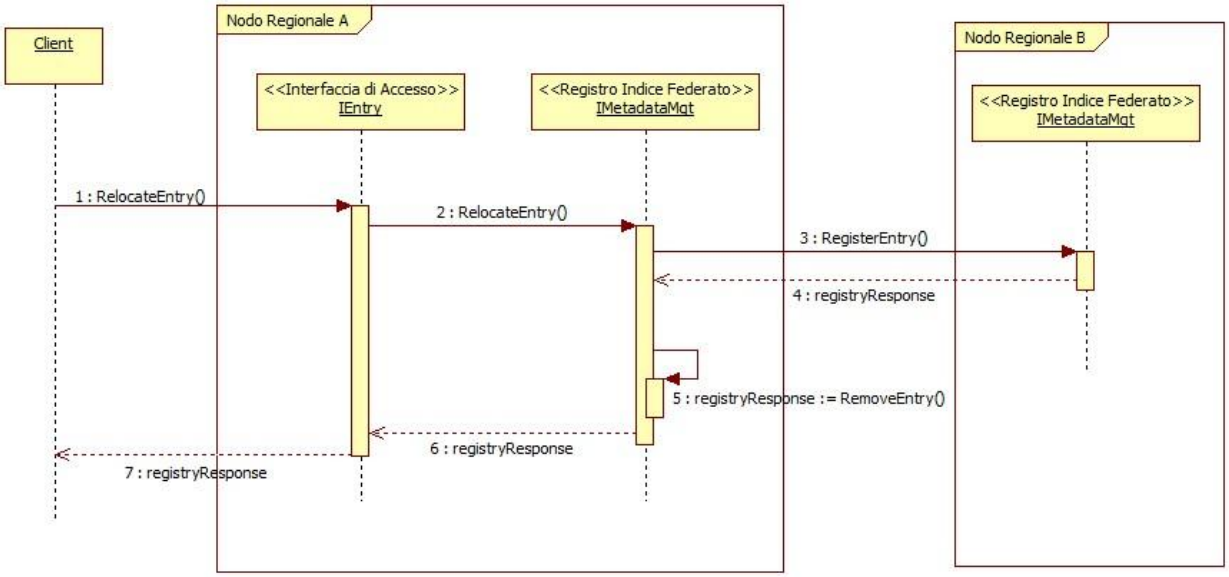
RegisterEntry()



1. Il client invoca l'operazione *RegisterEntry()*, indicando i metadati da memorizzare;
2. il servizio *IEntry* invia i metadati al servizio *IMetadataMgt* noto;
3. il servizio *IMetadataMgt* memorizza i metadati nel registro e restituisce una risposta;

4. il servizio *IEntry* restituisce la risposta al client.

RelocateEntry()

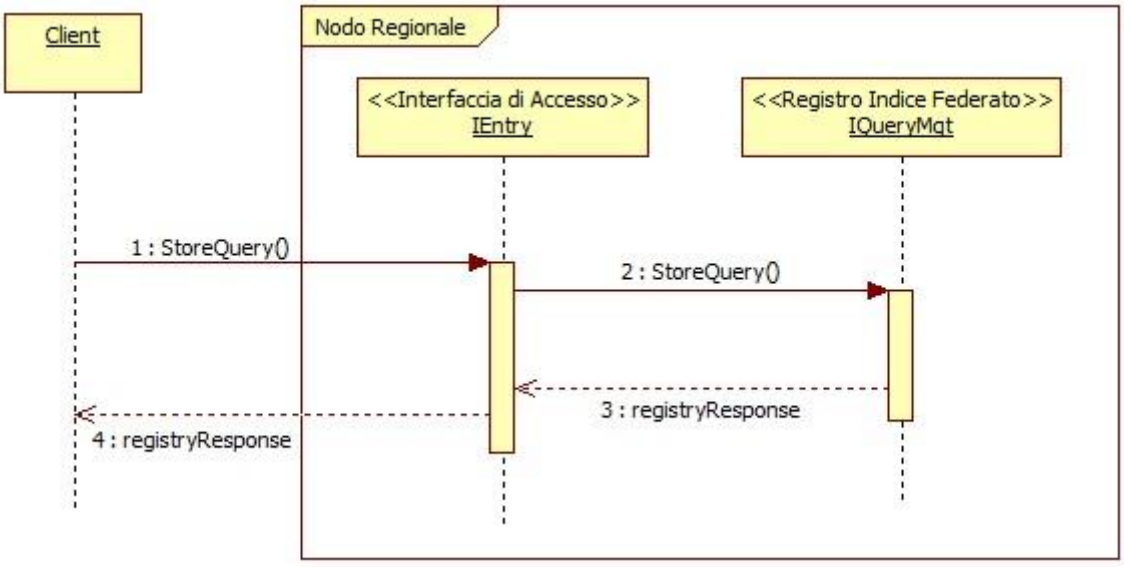


1. Il client invoca l'operazione *RelocateEntry()*, indicando i metadati da rilocare;
2. il servizio *IEntry* invia la richiesta al servizio *IMetadataMgt* noto;
3. il servizio *IMetadataMgt* recupera i metadati dal proprio registro e invoca l'operazione *RegisterEntry()* del servizio *IMetadataMgt* del Nodo Regionale B specificato nella richiesta;
4. il servizio *IMetadataMgt* del Nodo Regionale B memorizza i metadati indicati nel proprio registro e restituisce una risposta;
5. il servizio *IMetadataMgt* del Nodo Regionale A rimuove i metadati dal proprio registro (questa opzione è facoltativa);
6. il servizio *IMetadataMgt* del Nodo Regionale A restituisce una risposta al servizio *IEntry*;
7. il servizio *IEntry* restituisce la risposta al client.

3.2.1.2 Gestione delle query

Il servizio IEntry permette di sottoporre ad un registro query in linguaggio SQL e di memorizzare ed invocare stored query. In particolare, la richiesta di query in linguaggio SQL e l'invocazione di stored query possono essere effettuate in modalità federata o meno mediante l'operazione *QueryRegistry()*, mentre la memorizzazione di stored query è resa possibile dall'operazione *StoreQuery()*.

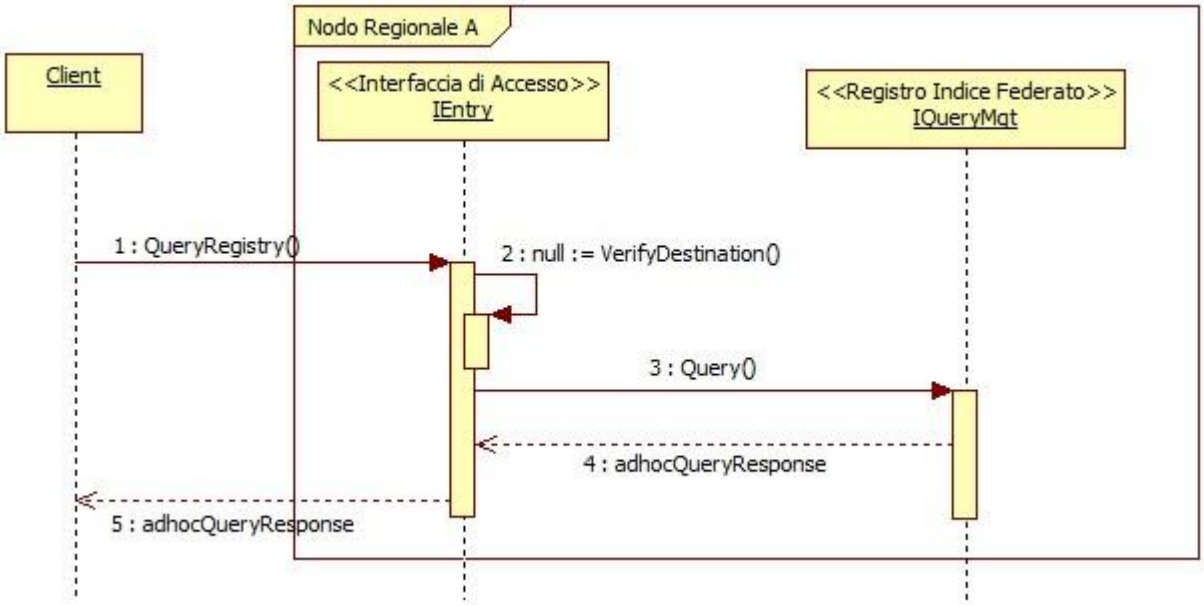
StoreQuery()



1. Il client invoca l'operazione *StoreQuery()*, indicando la stored query da memorizzare;
2. il servizio *IEntry* invia la richiesta al servizio *IQueryMgt* noto;
3. il servizio *IQueryMgt* memorizza la stored query nel registro e restituisce una risposta al servizio *IEntry*;
4. il servizio *IEntry* restituisce la risposta al client.

Query regionale non federata

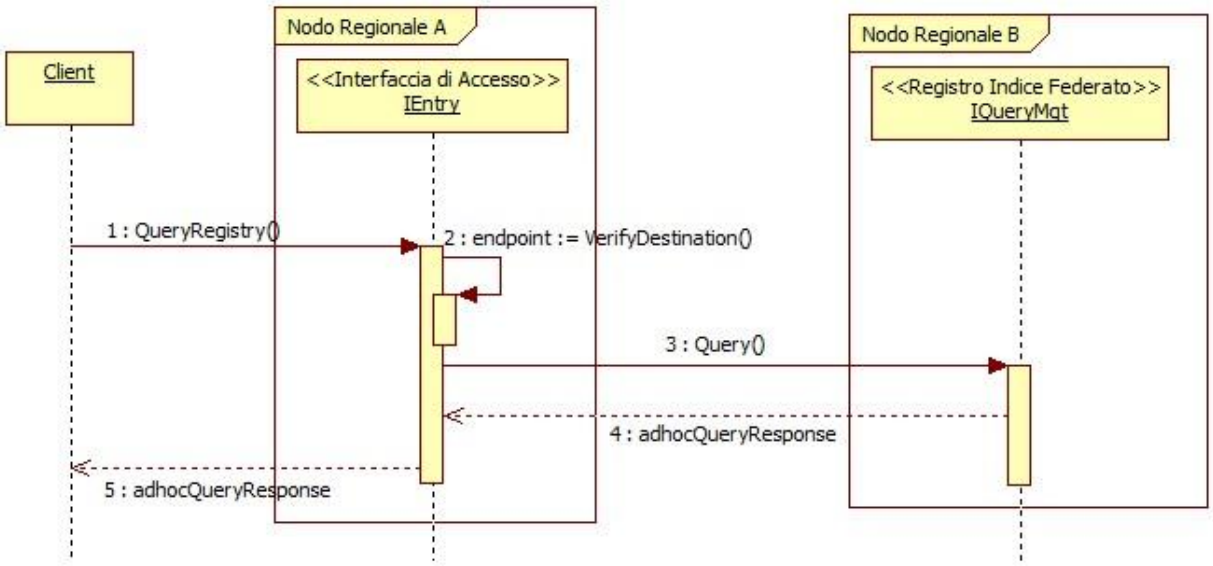
QueryRegistry()



1. Il client invoca l'operazione *QueryRegistry()*, indicando la stored query da invocare o la query in formato SQL da sottoporre;
2. il servizio *IEntry* verifica che la richiesta non contiene l'endpoint del servizio *IQueryMgt* da invocare;
3. il servizio *IEntry* invia la richiesta al servizio *IQueryMgt* noto;
4. il servizio *IQueryMgt* invoca la stored query o sottopone la query in formato SQL al registro e restituisce una risposta al servizio *IEntry*;
5. il servizio *IEntry* restituisce la risposta al client.

Query extra-regionale non federata

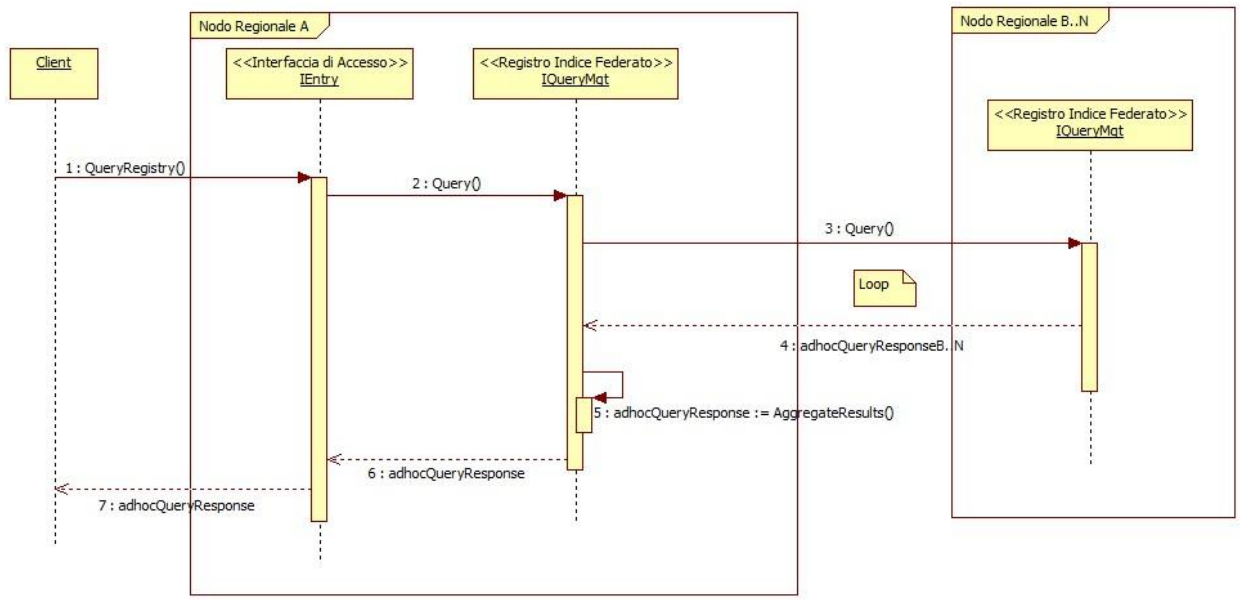
QueryRegistry()



1. Il client invoca l'operazione *QueryRegistry()*, indicando la stored query da invocare o la query in formato SQL da sottoporre;
2. il servizio *IEntry* verifica che la richiesta contiene l'endpoint del servizio *IQueryMgt* da invocare;
3. il servizio *IEntry* invia la richiesta al servizio *IQueryMgt* specificato;
4. il servizio *IQueryMgt* del nodo Regionale B invoca la stored query o sottopone la query in formato SQL al registro e restituisce una risposta al servizio *IEntry*;
5. il servizio *IEntry* restituisce la risposta al client.

Query federata

QueryRegistry()



1. Il client invoca l'operazione *QueryRegistry()* in modalità federata, indicando la stored query da invocare o la query in formato SQL da sottoporre;
2. il servizio *IEntry* invia la richiesta al servizio *IQueryMgt* noto;
3. il servizio *IQueryMgt* del nodo Regionale A recupera dal registro l'elenco degli endpoint ai servizi *IQueryMgt* che fanno parte della federazione e propaga la query a questi ultimi in modalità non federata;
4. tutti i servizi *IQueryMgt* contattati invocano la stored query o sottopongono la query in formato SQL al proprio registro e restituiscono una risposta;
5. il servizio *IQueryMgt* del nodo Regionale A invoca la stored query o sottopone la query in formato SQL al proprio registro e aggrega tutti i risultati;
6. il servizio *IQueryMgt* del nodo Regionale A restituisce una risposta complessiva al servizio *IEntry*;
7. il servizio *IEntry* restituisce la risposta al client.

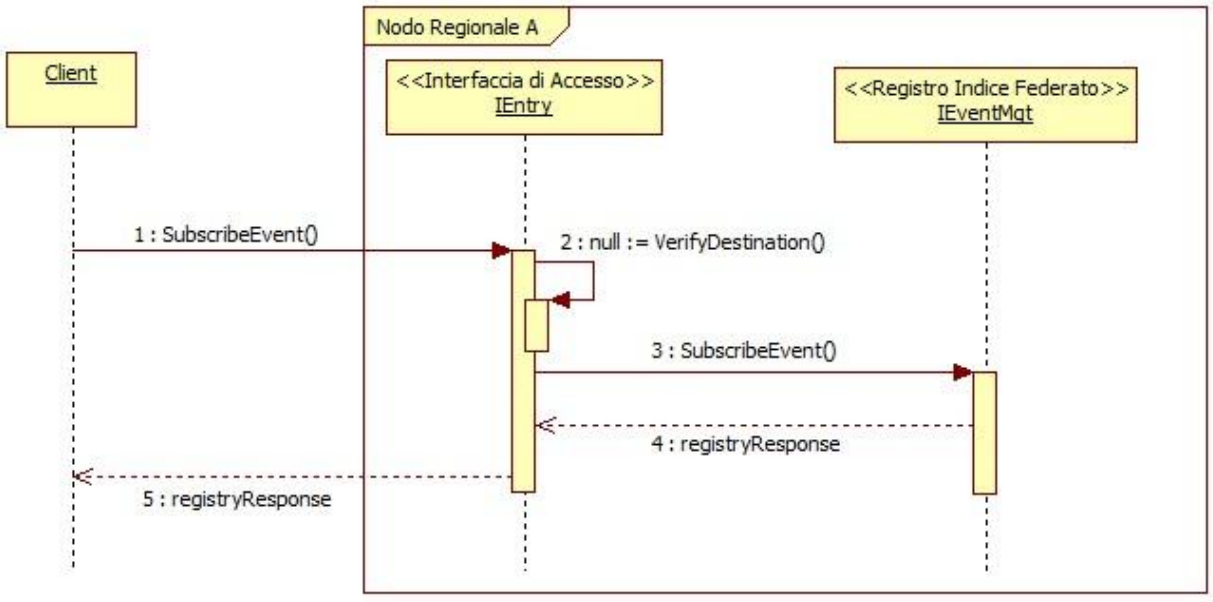
3.2.1.3 Gestione delle sottoscrizioni/notifiche di eventi

Le operazioni *SubscribeEvent()* e *UnsubscribeEvent()* permettono di memorizzare e rimuovere una sottoscrizione ad eventi di interesse che occorrono in un registro. L'obiettivo di tali funzionalità è quello di supportare la gestione delle federazioni di registri mediante meccanismi basati sulla notifica di eventi. In particolare, quest'ultima è realizzata dall'operazione *NotifyEvent()*, la quale trasmette una notifica ai consumatori sottoscritti e la memorizza nei registri.

Gli scenari per le operazioni *SubscribeEvent()* e *UnsubscribeEvent()* sono simili. Pertanto, di seguito sono mostrati due possibili scenari: la sottoscrizione ad un evento di interesse mediante l'operazione *SubscribeEvent()* e la notifica di eventi da un registro ad un altro mediante l'operazione *NotifyEvent()*.

Sottoscrizione regionale

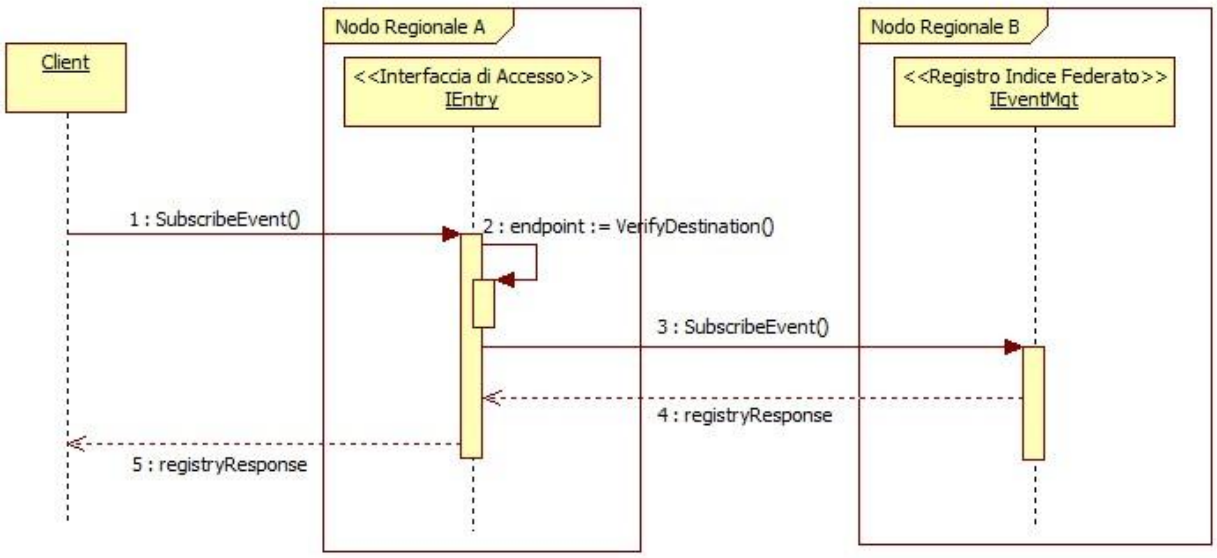
SubscribeEvent()



1. Il client invoca l'operazione *SubscribeEvent()*, specificando gli eventi di interesse dei quali intende essere notificato;
2. il servizio *IEntry* verifica che la richiesta non contiene l'endpoint del servizio *IEventMgt* da invocare;
3. il servizio *IEntry* invia la richiesta al servizio *IEventMgt* noto;
4. il servizio *IEventMgt* memorizza la sottoscrizione nel registro e restituisce una risposta al servizio *IEntry*;
5. il servizio *IEntry* restituisce la risposta al client.

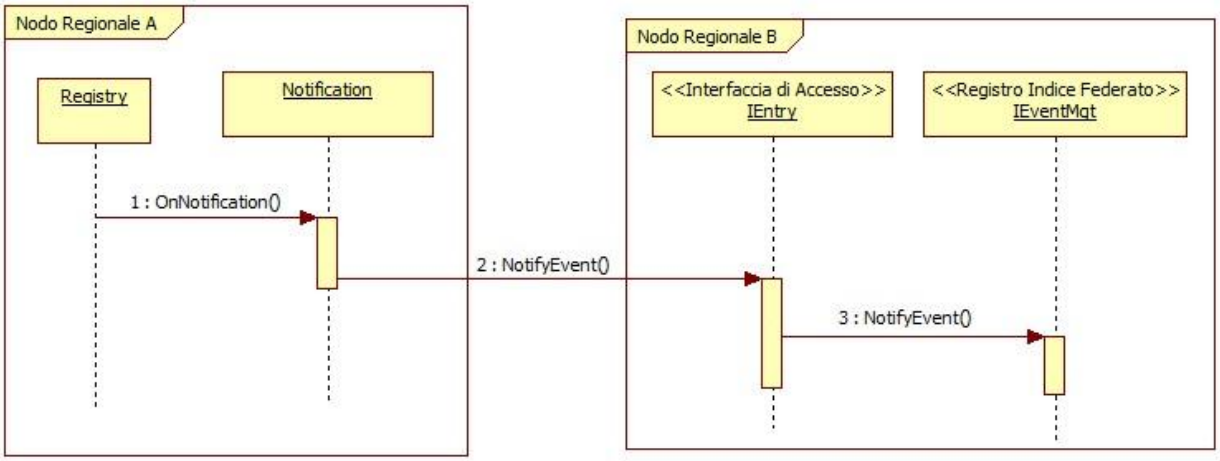
Sottoscrizione extra-regionale

SubscribeEvent()



1. Il client invoca l'operazione *SubscribeEvent()*, specificando gli eventi di interesse dei quali intende essere notificato;
2. il servizio *IEntry* verifica che la richiesta contiene l'endpoint del servizio *IEventMgt* da invocare;
3. il servizio *IEntry* invia la richiesta al servizio *IEventMgt* specificato;
4. il servizio *IEventMgt* del nodo regionale B memorizza la sottoscrizione nel registro e restituisce una risposta al servizio *IEntry*;
5. il servizio *IEntry* restituisce la risposta al client.

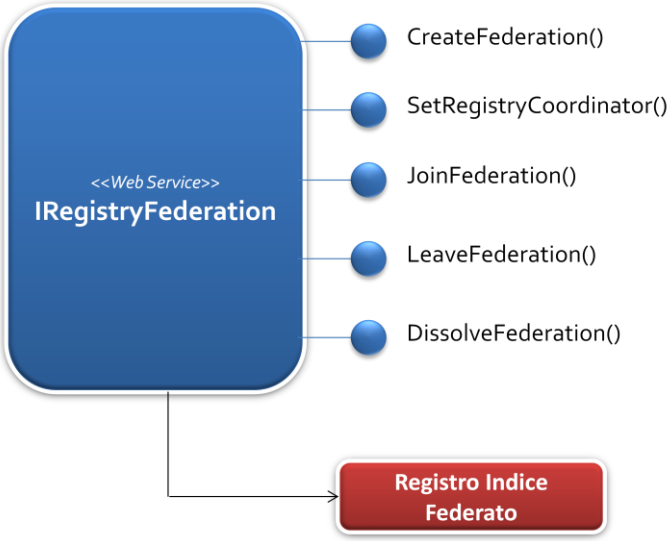
NotifyEvent()



1. All'occorrere di un evento, il registro del Nodo Regionale A invia una notifica, consistente in un insieme di metadati, al servizio di supporto *Notification*;
2. il servizio *Notification* invia la notifica al servizio *IEntry* del Nodo Regionale B, che aveva manifestato l'interesse a ricevere notifiche di questo tipo;
3. il servizio *IEntry* invia la notifica al servizio *IEventMgt* noto, il quale memorizza i metadati contenuti nella notifica nel proprio registro.

3.3 Servizio IRegistryFederation

IRegistryFederation è il servizio per la gestione delle federazioni di registri, mediante l'aggiornamento di opportuni metadati in un registro. Funge da proxy rispetto al servizio *IRegistryFederationMgt* della componente *Registro Indice Federato*.



Di seguito sono descritti tutti i metodi, con i parametri di ingresso/uscita, del servizio *IRegistryFederation*.

Servizio: **IRegistryFederation**	
Metodo: ***CreateFederation***	È il metodo che permette la creazione di una federazione.
Parametro IN: SubmitObjectsRequest	È il parametro che specifica la federazione da creare, attraverso la memorizzazione di opportuni metadati nei registri.
Parametro OUT: RegistryResponse	É il parametro che contiene la risposta del registro.

Servizio: **IRegistryFederation**	
Metodo: ***JoinFederation***	È il metodo che permette ad un nodo di connettersi logicamente ad una federazione. L'operazione è utilizzata per la prima connessione alla federazione da parte di un nuovo nodo.
Parametro IN: SubmitObjectsRequest	È il parametro che specifica il registro e la federazione da connettere.
0Parametro OUT: RegistryResponse	É il parametro che contiene la risposta del registro.

Servizio: **IRegistryFederation**	
Metodo: ***LeaveFederation***	È il metodo che permette ad un nodo di disconnettersi logicamente da una federazione.
Parametro IN: RemoveObjectsRequest	È il parametro che specifica la federazione da abbandonare attraverso la rimozione di opportuni metadati nei registri.
Parametro OUT: RegistryResponse	É il parametro che contiene la risposta del registro.

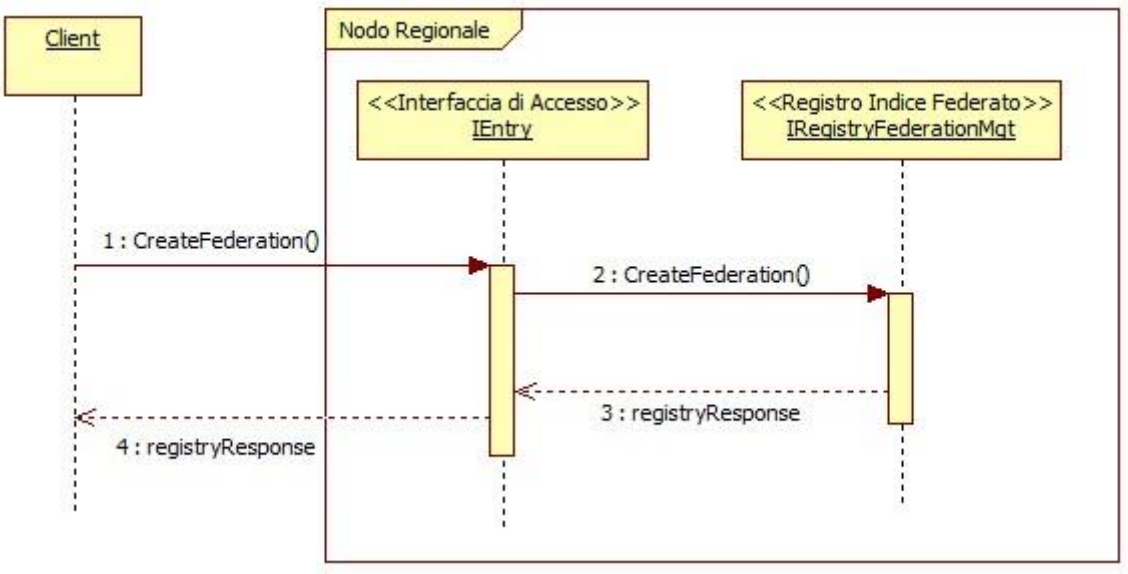
Servizio: **IRegistryFederation**	
Metodo: ***DissolveFederation***	È il metodo che permette l'eliminazione logica di una federazione.
Parametro IN: RemoveObjectsRequest	È il parametro che specifica la federazione da eliminare, attraverso la rimozione di opportuni metadati nei registri.
Parametro OUT: RegistryResponse	É il parametro che contiene la risposta del registro.

Servizio: **IRegistryFederation**	
Metodo: ***SetRegistryCoordinator***	È l'operazione che consente di specificare il registro coordinatore all'interno di una federazione locale esistente.
Parametro IN: SubmitObjectsRequest	È il parametro contenente le informazioni atte ad eleggere un registro coordinatore all'interno di una federazione locale esistente.
Parametro OUT: RegistryResponse	É il parametro che contiene la risposta del registro.

3.3.1 Scenari di interazione

Questa sezione descrive i possibili scenari di interazione per la gestione dei metadati attraverso il servizio *IRegistryFederation*. In particolare, gli scenari di interazione tra le componenti sono simili per tutte le operazioni. Pertanto, di seguito è mostrato un singono scenario esemplificativo, inerente alla creazione di una federazione.

CreateFederation()



1. Il client invoca l'operazione *CreateFederation()*, specificando le informazioni inerenti alla federazione da creare;
2. il servizio *IEntry* invia la richiesta al servizio *IRegistryFederationMgt* noto;
3. il servizio *IRegistryFederationMgt* crea una nuova federazione memorizzando opportuni metadati nel registro e restituisce una risposta al servizio *IEntry*;
4. il servizio *IEntry* restituisce la risposta al client.

3.4 Servizio IEvent

IEvent è il servizio per la pubblicazione e ricezione di eventi nel FSE. Esso invoca i servizi *IPublisherRegistrationMgt*, *INotificationBrokerMgt* e *ISubscriptionMgt*. Di seguito sono descritti tutti i metodi, con i parametri di ingresso/uscita, del servizio.

Servizio: **IEvent**	
Metodo: ***Register***	È il metodo che permette la pubblicazione di uno o più topic. Attraverso questa funzionalità un publisher comunica al gestore degli eventi la possibilità di generare nuovi eventi appartenenti ad una determinata classe.
Parametro IN: PublisherReference	Reference usata per identificare il produttore degli eventi.
Parametro IN: Topic[]	Lista di Topics.
Parametro OUT: RegistrationID	URN generato per la nuova registrazione.

Servizio: **IEvent**	
Metodo: ***RegisterHierarchy***	È il metodo che permette la pubblicazione di una o più gerarchie.
Parametro IN: PublisherReference	Reference usata per identificare il produttore degli eventi.
Parametro IN: HierachyID[]	Lista di identificativi univoci di gerarchie.
Parametro OUT: RegistrationID	URN generato per la nuova registrazione.

Servizio: **IEvent**	
Metodo: ***DestroyRegistration***	È il metodo che permette la cancellazione di una registrazione.
Parametro IN: RegistrationID	URN generato per la nuova registrazione.
Parametro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

Servizio: **IEvent**	
Metodo: ***CreateTopic***	È il metodo che permette di creare una nuova classe di eventi.
Parametro IN: Owner	Reference usata per identificare il creatore del topic.
Parametro IN: Topic	Parametro che definisce il nome della classe creata.
Parametro IN: Attribute[]	Lista opzionale che identifica una serie di attributi associati al topic.
Parametro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

Servizio: **IEvent**	
Metodo: ***ArchiveTopic***	È il metodo che permette di rimuovere una classe di eventi.
Parametro IN: Owner	Reference usata per identificare il creatore del topic.
Parametro IN: Topic	Parametro che definisce il nome della classe creata.
Parametro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

Servizio: **IEvent**	
Metodo: ***CreateHierarchy***	È il metodo che permette di creare una nuova gerarchia di eventi.
Parametro IN: Owner	Reference usata per identificare il creatore della gerarchia.
Parametro IN: RootTopic	Parametro che definisce il nome del RootTopic della gerarchia.
Parametro IN: Attribute[]	Lista opzionale che identifica una serie di attributi associati al topic.
Parametro IN: AttributeValue[]	Lista opzionale di valori associati agli attributi della gerarchia.
Parametro OUT: HierachyID	URN identificativo della nuova gerarchia.

Servizio: **IEvent**	
Metodo: ***ArchiveHierarchy***	È il metodo che permette di archiviare una gerarchia e tutti gli eventi ad essa associati.
Parametro IN: Owner	Reference usata per identificare il creatore del topic.
Parametro IN: HierachyID	URN identificativo della nuova gerarchia.
Parametro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

Servizio: **IEvent**	
Metodo: ***CreateAssociation***	È il metodo che permette di creare una nuova associazione tra topic di una gerarchia.
Parametro IN: Owner	Reference usata per identificare il creatore della gerarchia.
Parametro IN: FromTopic	Parametro che definisce il nome del Topic origine della associazione.
Parametro IN: ToTopic	Parametro che definisce il nome del Topic destinazione della associazione.
Parametro IN: HierachyID	URN identificativo della nuova gerarchia.
Parametro IN: isAnd	Valore booleano che codifica il tipo di associazione da creare: true per inidcare una AND e false per inidcare una OR.
Paremetro IN: expiration	Indica il numero di giorni di validità di un legame AND quando uno dei due topic associati ha ricevuto un evento.
Parametro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

Servizio: **IEvent**	
Metodo: ***RemoveAssociation***	È il metodo che permette di rimuovere una associazione tra topic di una gerarchia.
Parametro IN: Owner	Reference usata per identificare il creatore della gerarchia.
Parametro IN: FromTopic	Parametro che definisce il nome del Topic origine della associazione.
Parametro IN: ToTopic	Parametro che definisce il nome del Topic destinazione della associazione.
Parametro IN: HierachyID	URN identificativo della nuova gerarchia.
Paremetro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

Servizio: **IEvent**	
Metodo: ***GetCurrentMessageEvent***	È il metodo che permette di ottenere dal NotificationBroker l'ultimo messaggio notificato per un determinato topic già notificato a tutti i consumatori sottoscritti.
Parametro IN: SubscriptionID	URN identificativo di una data sottoscrizione.
Parametro IN: Topic	Identifica esattamente un Topic.
Paremetro OUT: event	Ultimo elemento pubblicato per il dato topic.

Servizio: **IEvent**	
Metodo: ***GetAllTopics***	È il metodo che permette di ottenere dal NotificationBroker la lista di tutti i topic accessibili. La lista varia in funzione dei diritti e dei ruoli dell'attore richiedente.
Parametro IN: HierachyID	URN opzionale identificativo della gerarchia dalla quale ottenere i topics. Se il parametro è {any} il metodo restituirà tutti i topics attivi.
Parametro OUT: Topic[]	Lista di topics.

Servizio: **IEvent**	
Metodo: ***GetAllHierarchies***	È il metodo che permette di ottenere dal NotificationBroker la lista di tutte le gerarchie.
Parametro IN: RootTopic	Parametro opzionale che specifica il nome del RootTopic della gerarchia richiesta.
Parametro IN: Attribute[]	Lista opzionale che identifica una serie di attributi.
Parametro IN: AttributeValue[]	Lista opzionale di valori associati agli attributi ed utilizzati come filtro.
Parametro OUT: Hierarchy[]	Lista di gerarchie.

Servizio: **IEvent**	
Metodo: ***RetrieveEvents***	È il metodo che permette di ottenere tutti gli eventi notificati ed associati ad un determinato topic.
Parametro IN: Topic	Parametro che specifica il nome del Topic di cui si voglio ottenere gli eventi.
Parametro IN: FromDate	Indica la data dell'inizio dell'intervallo temporale in cui cercare gli eventi.
Parametro IN: ToDate	Indica la data della fine dell'intervallo temporale in cui cercare gli eventi.
Parametro OUT: Event[]	Lista di eventi.

Servizio: **IEvent**	
Metodo: ***RetrieveHierarchy***	È il metodo che permette di ottenere tutti gli eventi notificati nell'ambito di una determinata gerarchia.
Parametro IN: HierachyID	URN identificativo della gerarchia dalla quale ottenere gli eventi notificati.
Parametro OUT: Event[]	Lista di eventi.

Servizio: **IEvent**	
Metodo: ***SubscribeTopics***	È il metodo che permette la sottoscrizione di uno o più topics, ossia permette la sottoscrizione di un consumatore presso il NotificationBroker per la ricezione degli eventi relativi ad un dato NotificationProducer.
Parametro IN: SubscriberReference	Una reference usata per identificare il consumatore degli eventi.
Parametro IN: Topic[]	Lista di topics da sottoscrivere.
Parametro IN: SubscriberType	Questo parametro permette di distinguere rispetto alla tipologia di sottoscrittore tra modalità push e pull.
Parametro IN: Expiration	Contiene informazioni circa la data di scadenza della sottoscrizione.
Parametro OUT: SubscriptionID	URN identificativo della nuova sottoscrizione.

Servizio: **IEvent**	
Metodo: ***SubscribeHierarchy***	È il metodo che permette la sottoscrizione di una intera gerarchia.
Parametro IN: SubscriberReference	Una reference usata per identificare il consumatore degli eventi.
Parametro IN: HierarchyID	Identificativo della gerarchia da sottoscrivere.
Parametro IN: Attribute[]	Lista opzionale di valori associati agli attributi.
Parametro IN: AttributeValue[]	Lista opzionale di valori associati agli attributi ed utilizzati come filtro.
Parametro IN: SubscriberType	Questo parametro permette di distinguere rispetto alla tipologia di sottoscrittore tra modalità push e pull.
Parametro IN: Expiration	Contiene informazioni circa la data di scadenza della sottoscrizione.
Parametro OUT: SubscriptionID	URN identificativo della nuova sottoscrizione.

Servizio: **IEvent**	
Metodo: ***RenewSubscription***	È il metodo che consente di rinnovare una sottoscrizione.
Parametro IN: SubscriptionID	URN identificativo di una data sottoscrizione.
Parametro IN: Expiration	Contiene informazioni circa la data di scadenza della sottoscrizione.
Parametro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

Servizio: **IEvent**	
Metodo: ***Unsubscribe***	È il metodo che permette la rimozione dall'elenco dei sottoscrittori.
Parametro IN: SubscriptionID	URN identificativo di una sottoscrizione.
Parametro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

Servizio: **IEvent**	
Metodo: ***NotifyEvent***	È il metodo che permette di notificare un evento ai consumers interessati.
Parametro IN: Owner	Reference usata per identificare il creatore del topic.
Parametro IN: SubscriptionID	URN identificativo di una sottoscrizione.
Parametro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

Servizio: **IEvent**	
Metodo: **AddContentFilter**	È il metodo che consente di assocare un filtro sul contenuto degli eventi ad un dato topic.
Parametro IN: Owner	Reference usata per identificare il creatore del topic.
Parametro IN: Topic	Topic su cui associare il filtro.
Parametro IN: ContentFilter	Specifica di un filtro sul contenuto degli eventi pubblicati.
Parametro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

3.5 Servizio IBrokerFederation

IBrokerFederation è il servizio per la gestione di federazioni di broker. Esso invoca il servizio *IBrokerFederationMgt*. Di seguito sono descritti tutti i metodi, con i parametri di ingresso/uscita, del servizio.

Servizio: **IBrokerFederation**	
Metodo: ***CreateFederation***	È il metodo che permette la creazione di una federazione.
Parametro IN: FederationID	URN identificativo della federazione.
Parametro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

Servizio: **IBrokerFederation**	
Metodo: ***JoinFederation***	È il metodo che permette ad un nodo di connettersi logicamente ad una federazione. Il metodo è utilizzato per la prima connessione alla federazione da parte di un nuovo nodo.
Parametro IN: FederationID	URN identificativo della federazione.
Parametro IN: NodeReference	Reference del nodo che si connette alla federazione.
Parametro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

Servizio: **IBrokerFederation**	
Metodo: ***Connect***	È il metodo che permette la creazione di una connessione logica tra due nodi della federazione.
Parametro IN: FederationID	URN identificativo della federazione.
Parametro IN: FromNodeReference	Reference del nodo origine della connessione.
Parametro IN: ToNodeReference	Reference del nodo destinazione della connessione.
Parametro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

Servizio: **IBrokerFederation**	
Metodo: ***LeaveFederation***	È il metodo che permette ad un nodo broker di disconnettersi logicamente da una federazione. Il metodo elimina tutte le connessioni logiche del nodo broker con ogni altro nodo broker della federazione.
Parametro IN: FederationID	URN identificativo della federazione.
Parametro IN: NodeReference	Reference del nodo broker che si disconnette dalla federazione.
Parametro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

Servizio: **IBrokerFederation**	
Metodo: ***DissolveFederation***	È il metodo che permette l'eliminazione logica di una federazione.
Parametro IN: FederationID	URN identificativo della federazione.
Parametro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

Servizio: **IBrokerFederation**	
Metodo: ***RouteEvent***	È il metodo che permette il routing degli eventi nella federazione di nodi broker. Il servizio analizza l'evento da propagare ed in funzione del tipo di evento può richiamare servizi del nodo broker locale prima di trasmettere l'evento ad altri nodi broker regionali.
Parametro IN: NodeReference	Reference del nodo broker a cui trasmettere l'evento.
Parametro IN: Msg	Messaggio da inoltrare.
Parametro: ReMsg	Messaggio di ritorno (opzionale).
Parametro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

4 Registro Indice Federato

In questa sezione sono descritti i seguenti servizi della componente *Registro Indice Federato*:

- *IMetadataMgt*;
- *IQueryMgt*;
- *IEventMgt*;
- *IRegistryFederationMgt*.

Tutti i servizi elencati interagiscono con un registro. In particolare, il registro utilizzato per la gestione dei metadati è basato sul sistema open source OMAR, rilasciato da freebXML.

Si noti che, allo scopo di agevolare l'integrazione dei servizi del *Registro Inidce Federato* con un generico registro, la logica di interazione con il registro è stata implementata attraverso un'interfaccia disponibile in una libreria denominata *com.cnr.infse*. L'interazione con il registro OMAR è stata implementata all'interno di un'ulteriore libreria che implementa tale interfaccia, denominata *com.cnr.infse.omar*. Le due librerie sono disponibili all'interno dei file WAR di ogni servizio.

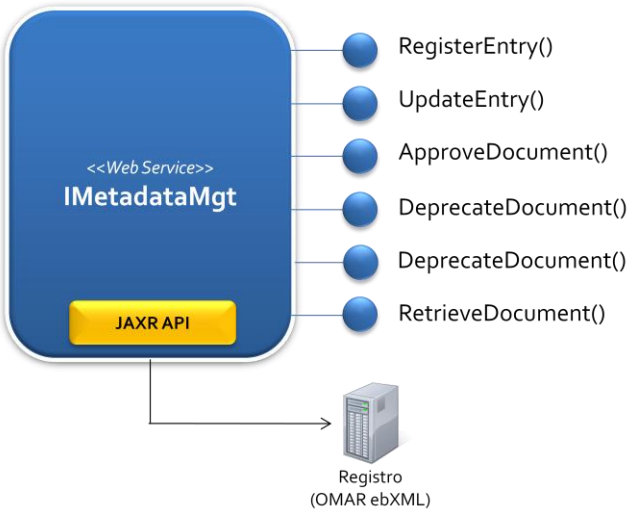
Pertanto, l'interazione con un generico registro può essere effettuata sviluppando una nuova libreria che implementa l'interfaccia *RegistryDriver*, le cui operazioni sono mostrate nella prossima tabella.

RegistryResponseType registerEntry(SubmitObjectsRequest) throws Exception
Registra i metadati specificati in un regsitro e resituisce una risposta.
RegistryResponseType updateEntry(UpdateObjectsRequest) throws Exception
Aggiorna i metadati specificati in un regsitro e resituisce una risposta.
RegistryResponseType removeEntry(RemoveObjectsRequest) throws Exception
Rimuove i metadati specificati in un regsitro e resituisce una risposta.
RegistryResponseType approveEntry(ApproveObjectsRequest) throws Exception
Approva i metadati specificati in un regsitro e resituisce una risposta.
RegistryResponseType deprecateEntry(DeprecateObjectsRequest) throws Exception
Invalida i metadati specificati in un regsitro e resituisce una risposta.
RegistryResponseType undeprecateEntry(UndeprecateObjectsRequest) throws Exception
Rivalida i metadati specificati in un regsitro e resituisce una risposta.
List relocateEntry(RelocateObjectsRequest) throws Exception

Trasferisce un elenco di metadati da un registro ad un altro e li resituisce.
Collection<RegistryError> newAdhocQuery(AdhocQueryType)
Memorizza una query in un registro e restituisce un elenco di eventuali errori.
List<String> getListRegistryAddressToPropagate()
Restituisce l'elenco dei registri che fanno parte della federazione.
AdhocQueryResponse query(AdhocQueryRequest) throws JAXRException
Sottopone una query ad un registro e restituisce una risposta.
RegistryResponseType storeQuery(SubmitObjectsRequest)
Memorizza una stored query in un registro e restituisce una risposta.
RegistryResponseType subscribeEvent(SubmitObjectsRequest)
Memorizza una sottoscrizione in un registro e restituisce una risposta.
RegistryResponseType unsubscribeEvent(RemoveObjectsRequest)
Rimuove una sottoscrizione presente in un registro e restituisce una risposta.
void notifyEvent(NotificationType)
Notifica un evento ad un registro, memorizzando i metadati specificati.
RegistryResponseType createFederation(SubmitObjectsRequest)
Crea una nuova federazione in un registro, memorizzando i metadati specificati.
RegistryResponseType joinFederation(JoinFederationType)
Associa un nuovo registro ad una federazione esistente, memorizzando i metadati specificati.
RegistryResponseType leaveFederation(LeaveFederationType)
Rimuove un registro da una federazione esistente, memorizzando i metadati specificati.
RegistryResponseType dissolveFederation(RemoveObjectsRequest)
Rimuove una federazione esistente, eliminando i metadati specificati.
RegistryResponseType setRegistryCoordinator(SetRegistryCoordinatorType)
Imposta un registro coordinatore in una federazione esistente, memorizzando i metadati specificati.

4.1 Servizio IMetadataMgt

Il servizio *IMetadataMgt* interagisce con un registro conforme alle specifiche ebXML RegRep v3.0 e supporta le operazioni per gestire il ciclo di vita dei metadati.



Di seguito sono descritti tutti i metodi, con i parametri di ingresso/uscita, del servizio *IMetadataMgt*.

Servizio: **IMetadataMgt**	
Metodo: ***RegisterEntry***	É l'operazione che permette la registrazione di nuovi metadati all'interno di un registro.
Parametro: SubmitObjectsRequest	É il parametro che contiene i metadati da inserire all'interno del registro in maniera conforme alle specifiche ebXML RegRep 3.0.
Parametro: RegistryResponse	É il parametro che contiene la risposta del registro.

Servizio: **IMetadataMgt**	
Metodo: ***UpdateEntry***	É l'operazione che consente di aggiornare metadati esistenti in un registro.
Parametro: UpdateObjectsRequest	É il parametro che contiene i metadati da utilizzare per l'aggiornamento.
Parametro: RegistryResponse	É il parametro che contiene la risposta del registro.

Servizio: **IMetadataMgt**	
Metodo: ***ApproveEntry***	É l'operazione che consente di approvare metadati esistenti in un registro.
Parametro: ApproveObjectsRequest	É il parametro che specifica i criteri da utilizzare per l'approvazione dei metadati.
Parametro: RegistryResponse	É il parametro che contiene la risposta del registro.

Servizio: **IMetadataMgt**	
Metodo: ***DeprecateEntry***	É l'operazione che consente di invalidare metadati esistenti in un registro.
Parametro: DeprecateObjectsRequest	É il parametro che specifica i criteri da utilizzare per l'invalidazione dei metadati.
Parametro: RegistryResponse	É il parametro che contiene la risposta del registro.

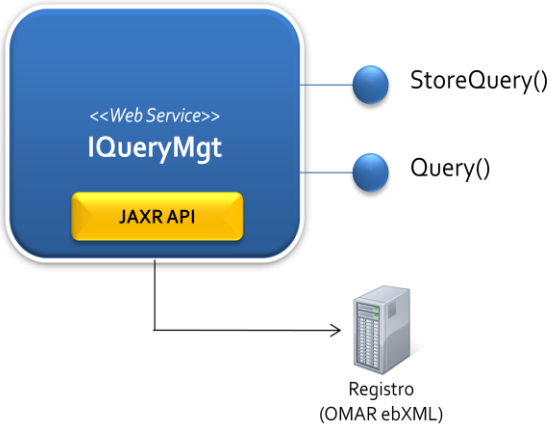
Servizio: **IMetadataMgt**	
Metodo: ***UndeprecateEntry***	É l'operazione che consente di validare metadati precedentemente dichiarati invalidi in un registro.
Parametro: UndeprecateObjectsRequest	É il parametro che specifica i criteri da utilizzare per la rivalidazione dei metadati.
Parametro:RegistryResponse	É il parametro che contiene la risposta del registro.

Servizio: **IMetadataMgt**	
Metodo: ***RemoveEntry***	É l'operazione che consente di rimuovere metadati esistenti in un registro.
Parametro: RemoveObjectsRequest	É il parametro che specifica i criteri da utilizzare per la rimozione dei metadati.
Parametro: RegistryResponse	É il parametro che contiene la risposta del registro.

Servizio: **IMetadataMgt**	
Metodo: ***RelocateEntry***	É l'operazione che consente di rilocare metadati da un registro ad un altro.
Parametro: RelocateObjectsRequest	É il parametro che specifica i criteri da utilizzare per la rilocazione dei metadati.
Parametro: RegistryResponse	É il parametro che contiene la risposta del registro.

4.2 Servizio IQueryMgt

Il servizio *IQueryMgt* interagisce con un registro conforme alle specifiche ebXML RegRep v3.0 e supporta le operazioni per sottoporre query ad uno specifico registro o ad una federazione di registri.



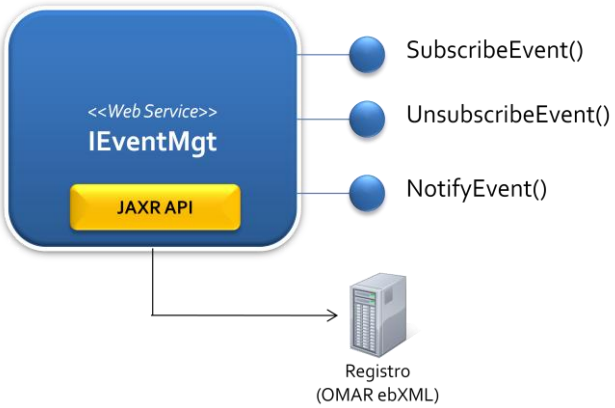
Di seguito sono descritti tutti i metodi, con i parametri di ingresso/uscita, del servizio *IQueryMgt*.

Servizio: **IQueryMgt**	
Metodo: ***Query***	É l'operazione che consente di sottoporre query locali o federate ai registri e di ottenere i risultati.
Parametro IN: AdhocQueryRequest	É il parametro che specifica i criteri da utilizzare per la query in maniera conforme alle specifiche ebXML RegRep v3.0. Per indicare una modalità di query federata, è opportuno impostare a *true* l'attributo *federated*. In questo caso, il servizio interroga il registro per determinare quali sono i nodi federati con il nodo corrente e propaga la query a tutti i servizi *IQueryMgt* della federazione.
Parametro OUT: AdhocQueryResponse	É il parametro che contiene la risposta del registro invocato.

Servizio: **IQueryMgt**	
Metodo: ***StoreQuery***	É l'operazione che consente di memorizzare stored query all'interno di un registro.
Parametro IN: SubmitObjectsRequest	É il parametro che specifica la query da memorizzare nel registro, al fine di ottimizzare l'esecuzione delle query.
Parametro OUT: RegistryResponse	É il parametro che contiene la risposta del registro.

4.3 Servizio IEventMgt

Il servizio *IEventMgt* interagisce con un registro conforme alle specifiche ebXML RegRep v3.0 e supporta le operazioni per manisfestare l'interesse a ricevere notifiche su eventi di interesse.Allo scopo di rispettare il sistema di notifica di OMAR, il rilascio comprende un servizio, denominato *Notification*, che intercetta le notifiche inviate dal registro e le propaga alle componenti software.



Di seguito sono descritti tutti i metodi, con i parametri di ingresso/uscita, del servizio *IEventMgt*.

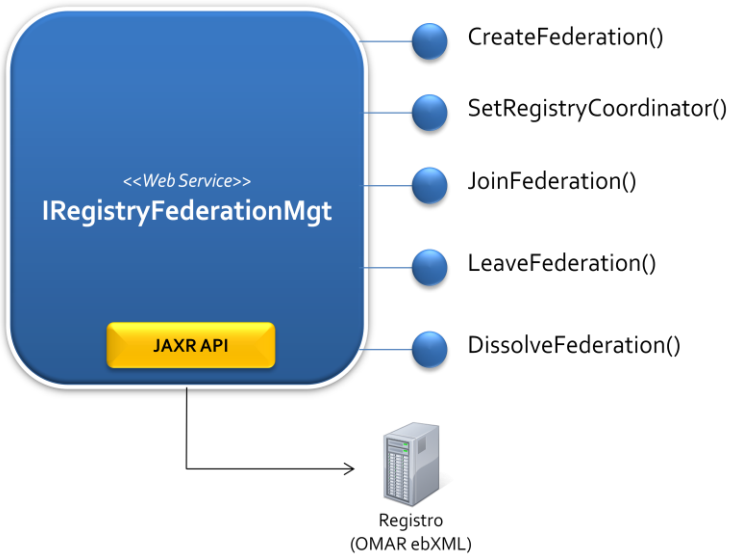
Servizio: **IEventMgt**	
Metodo: ***SubscribeEvent***	É l'operazione che consente di sottoporre sottoscrizioni ad eventi di interesse.
Parametro IN: SubmitObjectsRequest	É il parametro che specifica gli eventi di interesse sulla base di criteri di selezione.
Parametro OUT: RegistryResponse	É il parametro che contiene la risposta del registro.

Servizio: **IEventMgt**	
Metodo: ***UnsubscribeEvent***	É l'operazione che consente di rimuovere sottoscrizioni.
Parametro IN: RemoveObjectsRequest	É il parametro che specifica le sottoscrizioni da rimuovere.
Parametro OUT: RegistryResponse	É il parametro che contiene la risposta del registro.

Servizio: **IEventMgt**	
Metodo: ***NotifyEvent***	É l'operazione che consente di notificare eventi di interesse.
Parametro IN: Notification	É il parametro contenente la notifica.

4.4 Servizio IRegistryFederationMgt

Il servizio *IRegistryFederationMgt* interagisce con un registro conforme alle specifiche ebXML RegRep v3.0 e supporta le operazioni per gestire le federazioni di registri.



Di seguito sono descritti tutti i metodi, con i parametri di ingresso/uscita, del servizio *IRegistryFederationMgt*.

Servizio: **IRegistryFederationMgt**	
Metodo: ***CreateFederation***	È il metodo che permette la creazione di una federazione.
Parametro IN: SubmitObjectsRequest	È il parametro che specifica la federazione da creare, attraverso la memorizzazione nei registri di opportuni metadati di tipo *Federation*.
Parametro OUT: RegistryResponse	É il parametro che contiene la risposta del registro.

Servizio: **IRegistryFederationMgt**	
Metodo: ***JoinFederation***	È il metodo che permette ad un nodo di connettersi logicamente ad una federazione. Il metodo è utilizzato per la prima connessione alla federazione da parte di un nuovo nodo.
Parametro IN: SubmitObjectsRequest	È il parametro che specifica la federazione a cui unirsi, attraverso la memorizzazione nei registri di opportuni metadati di tipo *Association*, che legano un concetto *Federation* ad un concetto *Registry*.
Parametro OUT: RegistryResponse	É il parametro che contiene la risposta del registro.

Servizio: **IRegistryFederationMgt**	
Metodo: ***LeaveFederation***	È il metodo che permette ad un nodo di disconnettersi logicamente da una federazione.
Parametro IN: RemoveObjectsRequest	È il parametro che specifica la federazione da abbandonare attraverso la rimozione nei registri di opportuni metadati di tipo *Association*, che legano un concetto *Federation* ad un concetto *Registry*.
Parametro OUT: RegistryResponse	É il parametro che contiene la risposta del registro.

Servizio: **IRegistryFederationMgt**	
Metodo: ***SetRegistryCoordinator***	È l'operazione che consente di specificare il registro coordinatore all'interno di una federazione locale esistente.
Parametro IN: SubmitObjectsRequest	È il parametro contenente le informazioni atte ad eleggere un registro coordinatore all'interno di una federazione locale esistente.
Parametro OUT: RegistryResponse	É il parametro che contiene la risposta del registro.

Servizio: **IRegistryFederationMgt**	
Metodo: ***DissolveFederation***	È il metodo che permette l'eliminazione logica di una federazione.
Parametro IN: RemoveObjectsRequest	È il parametro che specifica la federazione da eliminare, attraverso la rimozione nei registri di opportuni metadati di tipo *Federation*.
Parametro OUT: RegistryResponse	É il parametro che contiene la risposta del registro.

5 Gestore dei Documenti

In questa sezione è descritto il seguente servizio della componente *Gestore dei Documenti*:

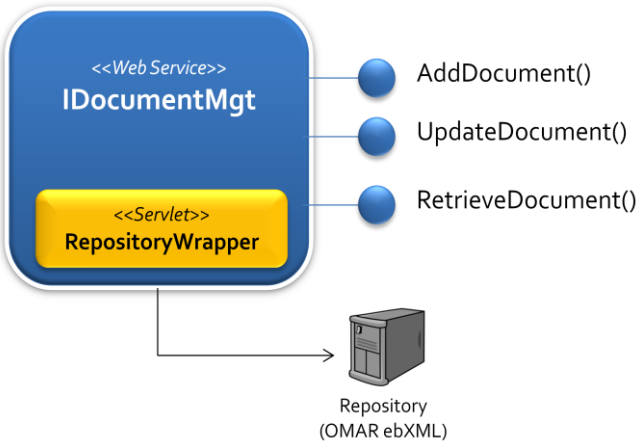
- *IDocumentMgt*.

Tale servizio interagisce con un repository per la gestione dei documenti. In particolare, il repository utilizzato per la memorizzazione dei documenti è basato sul sistema open source OMAR, rilasciato da freebXML.

Si noti che, allo scopo di agevolare l'integrazione del servizio con repository differenti da OMAR, la logica di interazione con il repository è stata implementata in una servlet denominata *Repository* ed invocata dal servizio *IDocumentMgt*.

5.1 Servizio IDocumentMgt

Il servizio *IDocumentMgt* interagisce con un repository e supporta le operazioni per la gestione dei documenti nei repository.



Sono di seguito descritti tutti i metodi, con i parametri di ingresso/uscita, del servizio *IDocumentMgt*.

Servizio: **IDocumentMgt**	
Metodo: ***AddDocument***	È il metodo che permette l'inserimento di un nuovo documento.
Parametro IN: Owner	Riferimento usato per identificare il servizio *IDocument* che ha effettuato la richiesta.
Parametro IN: Status	Parametro che permette di specificare lo stato iniziale del documento.
Parametro IN: DocumentObj	Documento in formato HL7-CDA2 con, opzionalmente, il documento in formato PDF/A o lo stylesheet in formato XLST.
Parametro OUT: DocumentID	Insieme di identificativi del documento richiesto comprendenti il riferimento al servizio *IDocument* del nodo regionale, il riferimento al servizio *IDocumentMgt* dove reperire il documento e l'identificativo del documento.

Servizio: **IDocumentMgt**	
Metodo: ***UpdateDocument***	È il metodo che permette l'aggiornamento di un documento pre-esistente.
Parametro IN: Owner	Riferimento usato per identificare il servizio *IDocument* che ha effettuato la richiesta.
Parametro IN: Status	Nuovo stato del documento.
Parametro IN: Version	Nuova versione del documento.
Parametro IN: DocumentObj	Documento da archiviare (in formato HL7-CDA2 con, opzionalmente, il documento in formato PDF/A o il documento di stylesheet in formato XLST).
Parametro IN: DocumentID	Insieme di identificativi del documento richiesto comprendenti l'URI del servizio *IDocument* del nodo regionale, l'URI del servizio *IDocumentMgt* dove reperire il documento e l'identificativo del documento.

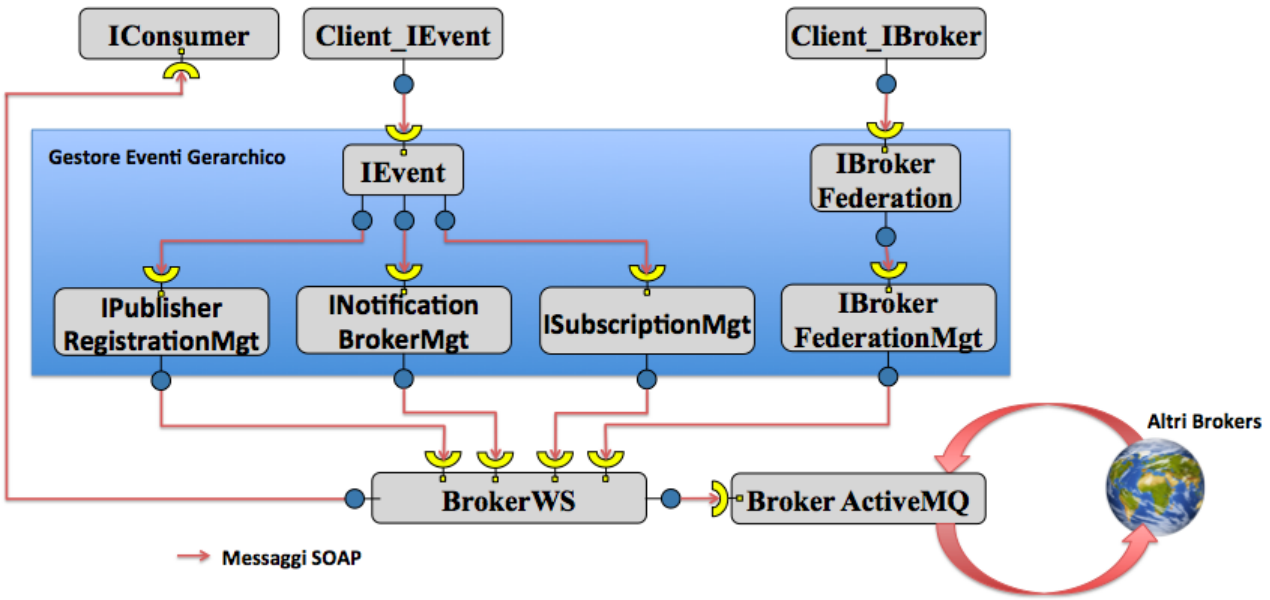
Servizio: **IDocumentMgt**	
Metodo: ***RetrieveDocument***	È il metodo che permette l'acquisizione di un documento.
Parametro IN: DocumentObj	Insieme di identificativi del documento richiesto comprendenti il riferimento al servizio *IDocument* del nodo regionale, il riferimento al servizio *IDocumentMgt* dove reperire il documento e l'identificativo del documento.
Parametro OUT: DocumentID	Documento in formato HL7-CDA2 con, opzionalmente, il documento in formato PDF/A o il documento di stylesheet in formato XLST.

6 Gestore Gerarchico degli Eventi

In questa sezione sono descritti i seguenti servizi della componente *Gestore Gerearchico degli Eventi*:

- *IPublisherRegistrationMgt* – rappresenta il componente del gestore per la creazione e distruzione di advertisements di pubblicazione di eventi su determinati topic e/o gerarchie;
- *INotificationBrokerMgt* – rappresenta il componente del gestore per la gestione di topics e gerarchie e la notifica di eventi;
- *ISubscriptionMgt* – rappresenta il componente del gestore delle sottoscrizioni ed il consumo di notifiche;
- *IBrokerFederationMgt* – rappresenta il componente del gestore per la definizione di federazioni di broker;
- *IConsumer* – rappresenta il Web Service che il client deve implementare per la gestione di notifiche push-based;
- *BrokerWS* – implementa la logica applicativa del broker alla base del gestore degli eventi in OpenINFSE.

Di seguito viene riportata un'overview architetturale delle interdipendenze tra questi servizi:



6.1 Servizio IPublisherRegistrationMgt

IPublisherRegistrationMgt è il servizio per la gestione di intenti di pubblicazione. Esso invoca il servizio *BrokerWS*. Di seguito sono descritti tutti i metodi, con i parametri di ingresso/uscita, del servizio.

Servizio: **IPublisherRegistrationMgt**	
Metodo: ***Register***	È il metodo che permette la pubblicazione di uno o più topic. Attraverso questa funzionalità un publisher comunica al gestore degli eventi la possibilità di generare nuovi eventi appartenenti ad una determinata classe.
Parametro IN: PublisherReference	Reference usata per identificare il produttore degli eventi.
Parametro IN: Topic[]	Lista di Topics.
Parametro OUT: RegistrationID	URN generato per la nuova registrazione.

Servizio: **IPublisherRegistrationMgt**	
Metodo: ***RegisterHierarchy***	*È il metodo che permette la pubblicazione di una o più gerarchie.*
Parametro IN: *PublisherReference*	*Reference usata per identificare il produttore degli eventi.*
Parametro IN: *HierachyID[]*	*Lista di identificativi univoci di gerarchie.*
Parametro OUT: *RegistrationID*	*URN generato per la nuova registrazione.*

Servizio: **IPublisherRegistrationMgt**	
Metodo: ***DestroyRegistration***	È il metodo che permette la cancellazione di una registrazione.
Parametro IN: RegistrationID	URN generato per la nuova registrazione.
Parametro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

6.2 Servizio INotificationBrokerMgt

INotificationBrokerMgt è il servizio per la gestione di topics e gerarchie. Esso invoca il servizio BrokerWS. Di seguito sono descritti tutti i metodi, con i parametri di ingresso/uscita, del servizio.

Servizio: **INotificationBrokerMgt**	
Metodo: ***CreateTopic***	È il metodo che permette di creare una nuova classe di eventi.
Parametro IN: Owner	Reference usata per identificare il creatore del topic.
Parametro IN: Topic	Parametro che definisce il nome della classe creata.
Parametro IN: Attribute[]	Lista opzionale che identifica una serie di attributi associati al topic.
Parametro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

Servizio: **INotificationBrokerMgt**	
Metodo: ***ArchiveTopic***	È il metodo che permette di archiviare una classe di eventi.
Parametro IN: Owner	Reference usata per identificare il creatore del topic.
Parametro IN: Topic	Parametro che definisce il nome della classe creata.
Parametro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

Servizio: **INotificationBrokerMgt**	
Metodo: ***CreateHierarchy***	È il metodo che permette di creare una nuova gerarchia di eventi.
Parametro IN: Owner	Reference usata per identificare il creatore della gerarchia.
Parametro IN: RootTopic	Parametro che definisce il nome del RootTopic della gerarchia.
Parametro IN: Attribute[]	Lista opzionale che identifica una serie di attributi associati al topic.
Parametro IN: AttributeValue[]	Lista opzionale di valori associati agli attributi della gerarchia.
Parametro OUT: HierachyID	URN identificativo della nuova gerarchia.

Servizio: **INotificationBrokerMgt**	
Metodo: ***ArchiveHierarchy***	È il metodo che permette di archiviare una gerarchia e tutti gli eventi ad essa associati.
Parametro IN: Owner	Reference usata per identificare il creatore del topic.
Parametro IN: HierachyID	URN identificativo della nuova gerarchia.
Parametro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

Servizio: **INotificationBrokerMgt**	
Metodo: ***CreateAssociation***	È il metodo che permette di creare una nuova associazione tra topic di una gerarchia.
Parametro IN: Owner	Reference usata per identificare il creatore della gerarchia.
Parametro IN: FromTopic	Parametro che definisce il nome del Topic origine della associazione.
Parametro IN: ToTopic	Parametro che definisce il nome del Topic destinazione della associazione.
Parametro IN: HierachyID	URN identificativo della nuova gerarchia.
Parametro IN: isAnd	Valore booleano che codifica il tipo di associazione da creare: true per inidcare una AND e false per inidcare una OR.
Paremetro IN: expiration	Indica il numero di giorni di validità di un legame AND quando uno dei due topic associati ha ricevuto un evento.
Parametro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

Servizio: **INotificationBrokerMgt**	
Metodo: ***RemoveAssociation***	È il metodo che permette di rimuovere una associazione tra topic di una gerarchia.
Parametro IN: Owner	Reference usata per identificare il creatore della gerarchia.
Parametro IN: FromTopic	Parametro che definisce il nome del Topic origine della associazione.
Parametro IN: ToTopic	Parametro che definisce il nome del Topic destinazione della associazione.
Parametro IN: HierachyID	URN identificativo della nuova gerarchia.
Parametro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

Servizio: **INotificationBrokerMgt**	
Metodo: ***GetCurrentMessageEvent***	È il metodo che permette di ottenere dal NotificationBroker l'ultimo messaggio notificato per un determinato topic già notificato a tutti i consumatori sottoscritti.
Parametro IN: SubscriptionID	URN identificativo di una data sottoscrizione.
Parametro IN: Topic	Identifica esattamente un Topic.
Parametro OUT: Event	Messaggio di notifica.

Servizio: **INotificationBrokerMgt**	
Metodo: **NotifyEvent**	È il metodo che permette la notifica ai sottoscrittori locali.
Parametro IN: Notification	Messaggio di notifica.
Parametro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

Servizio: **INotificationBrokerMgt**	
Metodo: ***GetAllTopics***	È il metodo che permette di ottenere dal NotificationBroker la lista di tutti i topic accessibili. La lista varia in funzione dei diritti e dei ruoli dell'attore richiedente.
Parametro IN: HierachyID	URN opzionale identificativo della gerarchia dalla quale ottenere i topics. Se il parametro è {any} il metodo restituirà tutti i topics attivi.
Parametro OUT: Topic[]	Lista di topics.

Servizio: **INotificationBrokerMgt**	
Metodo: ***GetAllHierarchies***	È il metodo che permette di ottenere dal NotificationBroker la lista di tutte le gerarchie.
Parametro IN: RootTopic	Parametro opzionale che specifica il nome del RootTopic della gerarchia richiesta.
Parametro IN: Attribute[]	Lista opzionale che identifica una serie di attributi.
Parametro IN: AttributeValue[]	Lista opzionale di valori associati agli attributi ed utilizzati come filtro.
Parametro OUT: Hierarchy[]	Lista di gerarchie.

Servizio: **INotificationBrokerMgt**	
Metodo: **RetrieveEvents**	È il metodo che permette di ottenere tutti gli eventi notificati ed associati ad un determinato topic.
Parametro IN: Topic	Parametro che specifica il nome del Topic di cui si vogliono ottenere gli eventi.
Parametro IN: FromDate	Parametro di tipo data per la specifica di un intervallo di ricerca degli eventi.
Parametro IN: ToDate	Parametro di tipo data per la specifica di un intervallo di ricerca degli eventi.
Parametro OUT: Event[]	Lista di eventi.

Servizio: **INotificationBrokerMgt**	
Metodo: ***RetrieveHierarchy***	È il metodo che permette di ottenere tutti gli eventi notificati nell'ambito di una determinata gerarchia.
Parametro IN: HierachyID	URN identificativo della gerarchia dalla quale ottenere gli eventi notificati.
Peremetro OUT: Event[]	Lista di eventi.

6.3 Servizio ISubscriptionMgt

ISubscriptionMgt è il servizio per la gestione di notifiche. Esso invoca il servizio *BrokerWS*. Di seguito sono descritti tutti i metodi, con i parametri di ingresso/uscita, del servizio.

Servizio: **ISubscritionMgt**	
Metodo: ***SubscribeTopics***	È il metodo che permette la sottoscrizione di uno o più topics, ossia permette la sottoscrizione di un consumatore presso il NotificationBroker per la ricezione degli eventi relativi ad un dato NotificationProducer.
Parametro IN: SubscriberReference	Una reference usata per identificare il consumatore degli eventi.
Parametro IN: Topic[]	Lista di topics da sottoscrivere.
Parametro IN: SubscriberType	Questo parametro permette di distinguere rispetto alla tipologia di sottoscrittore tra modalità push e pull.
Parametro IN: Expiration	Contiene informazioni circa la data di scadenza della sottoscrizione.
Parametro OUT: SubscriptionID	URN identificativo della nuova sottoscrizione.

Servizio: **ISubscritionMgt**	
Metodo: ***SubscribeHierarchy***	È il metodo che permette la sottoscrizione di una intera gerarchia.
Parametro IN: SubscriberReference	Una reference usata per identificare il consumatore degli eventi.
Parametro IN: HierarchyID	Identificativo della gerarchia da sottoscrivere.
Parametro IN: Attribute[]	Lista opzionale di valori associati agli attributi.
Parametro IN: AttributeValue[]	Lista opzionale di valori associati agli attributi ed utilizzati come filtro.
Parametro IN: SubscriberType	Questo parametro permette di distinguere rispetto alla tipologia di sottoscrittore tra modalità push e pull.
Parametro IN: Expiration	Contiene informazioni circa la data di scadenza della sottoscrizione.
Parametro OUT: SubscriptionID	URN identificativo della nuova sottoscrizione.

Servizio: **ISubscritionMgt**	
Metodo: ***RenewSubscription***	È la metodo che consente di rinnovare una sottoscrizione.
Parametro IN: SubscriptionID	URN identificativo della nuova sottoscrizione.
Parametro IN: Expiration	Contiene informazioni circa la data di scadenza della sottoscrizione.
Parametro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

Servizio: **ISubscritionMgt**	
Metodo: ***Unsubscribe***	È il metodo che permette la rimozione dall'elenco dei sottoscrittori.
Parametro IN: SubscriptionID	URN identificativo della nuova sottoscrizione.
Parametro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

Servizio: **ISubscritionMgt**	
Metodo: ***GetSubscribers***	È il metodo che permette di ottenere i sottoscrittori un determinato topic o gerarchia.
Parametro IN: HierarchyID	Identificativo della gerarchia.
Parametro IN: Attribute[]	Lista opzionale di valori associati agli attributi.
Parametro IN: AttributeValue[]	Lista opzionale di valori associati agli attributi ed utilizzati come filtro.
Parametro IN: Topic[]	Parametro (opzionale ed alternativo al parametro HierarchyID) che specifica il nome dei Topic di cui si vogliono ottenere i sottoscrittori.
Parametro IN: SubscriberType	Questo parametro permette di distinguere rispetto alla tipologia di sottoscrittore.
Parametro OUT: Subscribers[]	Lista di reference usata per identificare i sottoscrittori individuati.

Servizio: **IEvent**	
Metodo: ***AddContentFilter***	È il metodo che consente di assocare un filtro sul contenuto degli eventi ad un dato topic.
Parametro IN: Owner	Reference usata per identificare il creatore del topic.
Parametro IN: Topic	Topic su cui associare il filtro.
Parametro IN: ContentFilter	Specifica di un filtro sul contenuto degli eventi pubblicati.
Parametro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

6.4 Servizio IBrokerFederationMgt

IBrokerFederationMgt è il servizio per la gestione di federazioni. Esso invoca il servizio *BrokerWS*. Di seguito sono descritti tutti i metodi, con i parametri di ingresso/uscita, del servizio.

Servizio: **IBrokerFederationMgt**	
Metodo: ***CreateFederation***	È il metodo che permette la creazione di una federazione.
Parametro IN: FederationID	URN identificativo della federazione.
Parametro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

Servizio: **IBrokerFederationMgt**	
Metodo: ***JoinFederation***	È il metodo che permette ad un nodo di connettersi logicamente ad una federazione. Il metodo è utilizzato per la prima connessione alla federazione da parte di un nuovo nodo.
Parametro IN: FederationID	URN identificativo della federazione.
Parametro IN: NodeReference	Reference del nodo che si connette alla federazione.
Parametro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

Servizio: **IBrokerFederationMgt**	
Metodo: ***Connect***	È il metodo che permette la creazione di una connessione logica tra due nodi della federazione.
Parametro IN: FederationID	URN identificativo della federazione.
Parametro IN: FromNodeReference	Reference del nodo origine della connessione.
Parametro IN: ToNodeReference	Reference del nodo destinazione della connessione.
Parametro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

Servizio: **IBrokerFederationMgt**	
Metodo: ***LeaveFederation***	È il metodo che permette ad un nodo broker di disconnettersi logicamente da una federazione. Il metodo elimina tutte le connessioni logiche del nodo broker con ogni altro nodo broker della federazione.
Parametro IN: FederationID	URN identificativo della federazione.
Parametro IN: NodeReference	Reference del nodo broker che si disconnette dalla federazione.
Parametro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

Servizio: **IBrokerFederationMgt**	
Metodo: ***DissolveFederation***	È il metodo che permette l'eliminazione logica di una federazione.
Parametro IN: FederationID	URN identificativo della federazione.
Parametro OUT: esito	Valore booleano che codifica l'esito dell'invocazione del metodo.

Servizio: **IBrokerFederationMgt**	
Metodo: ***RouteEvent***	È il metodo che permette il routing degli eventi nella federazione di nodi broker. Il servizio analizza l'evento da propagare ed in funzione del tipo di evento può richiamare servizi del nodo broker locale prima di trasmettere l'evento ad altri nodi broker regionali.
Parametro IN: NodeReference	Reference del nodo broker che si disconnette dalla federazione.
Parametro IN: Msg	Messaggio da inoltrare.
Parametro IN: ReMsg	Messaggio di ritorno (opzionale).

6.5 Servizio IConsumer

IConsumer è il servizio per il consumo push-based di notifiche. Esso viene invocato dal servizio *BrokerWS*, ed è responsabilità degli utenti implementarne la logica applicativa. Di seguito sono descritti tutti i metodi, con i parametri di ingresso/uscita, del servizio.

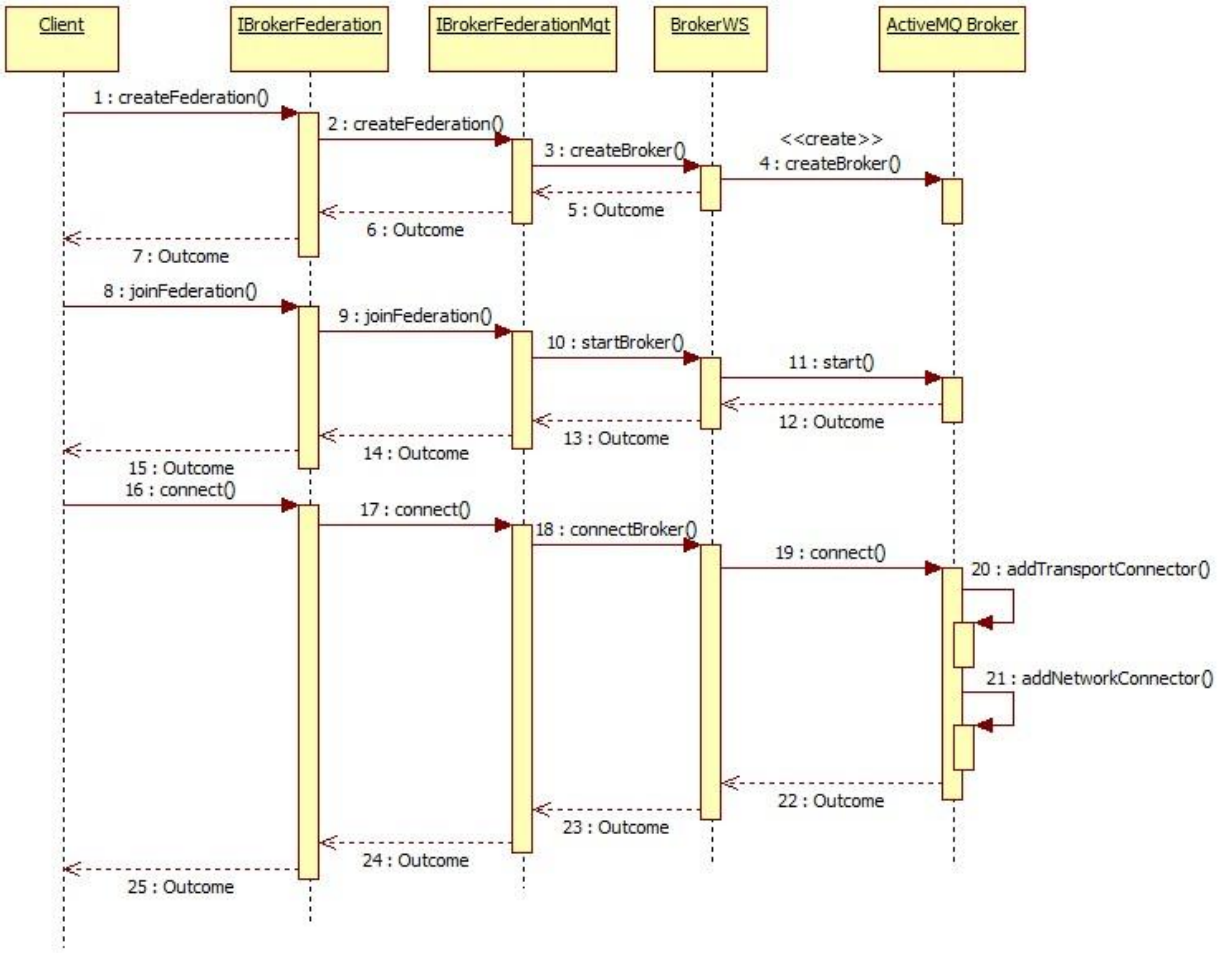
Servizio: **IConsumer**	
Metodo: ***ConsumeEvent***	È il metodo che permette di consumera in maniera push-based un dato evento
Parametro IN: Msg	Messaggio da consumare.

6.6 Servizio BrokerWS

BrokerWS rappresenta il core business del componente ed ne implementa la logica applicativa. La sua interfaccia rappresenta l'unione di tutti i metodi dei servizi sovrastanti.

6.7 Scenari di Utilizzo

6.7.1 Creazione di una federazione

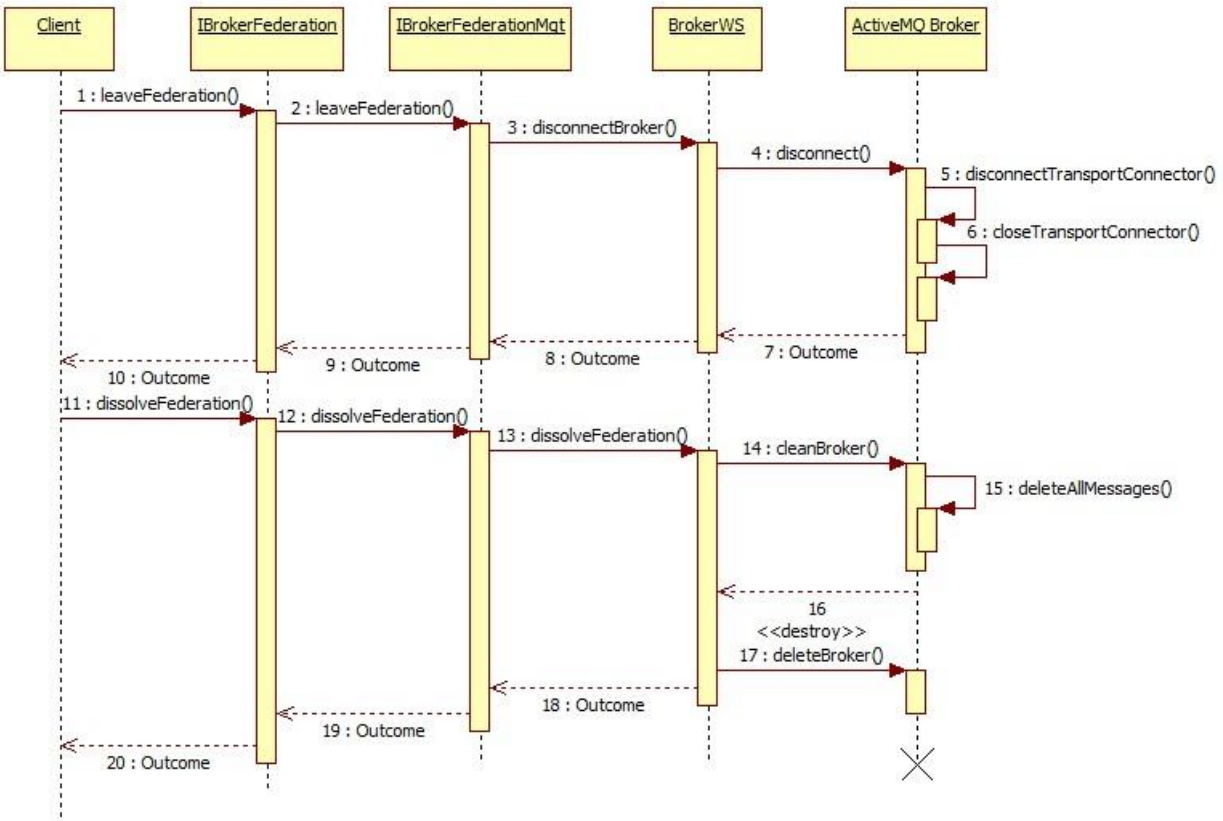


Lo scenario di creazione e connessione di una federazione si compone di una serie di tre operazioni, il cui ordine di invocazione deve essere mantenuto come in figura:

1. Creazione della federazione invocando *createFederation()*. Tale metodo arriva fino al servizio *BrokerWS*, che opera la creazione di un Broker del middleware Active MQ di Apache;
2. Collegamento della federazione creata invocando *joinFederation()*. Tale metodo viene passato a *BrokerWS* che opera lo start del Broker di Active MQ e la creazione di una nuova sessione di utilizzo;
3. Connessione della federazione creata invocando il metodo *connectFederation()*. Tale metodo invocato su *BrokerWS* comporta la definizione di un Transport Connector sul Broker di ActiveMQ che viene usato da *BrokerWS* per effettuare

la pubblicazione/ricezione di eventi verso il broker di ActiveMQ. Inoltre si ha anche la creazione di un NetworkConnector che serve per definire una rete di Broker, in funzione delle preferenze dell'utente contenuto del file di configurazione di *BrokerWS*.

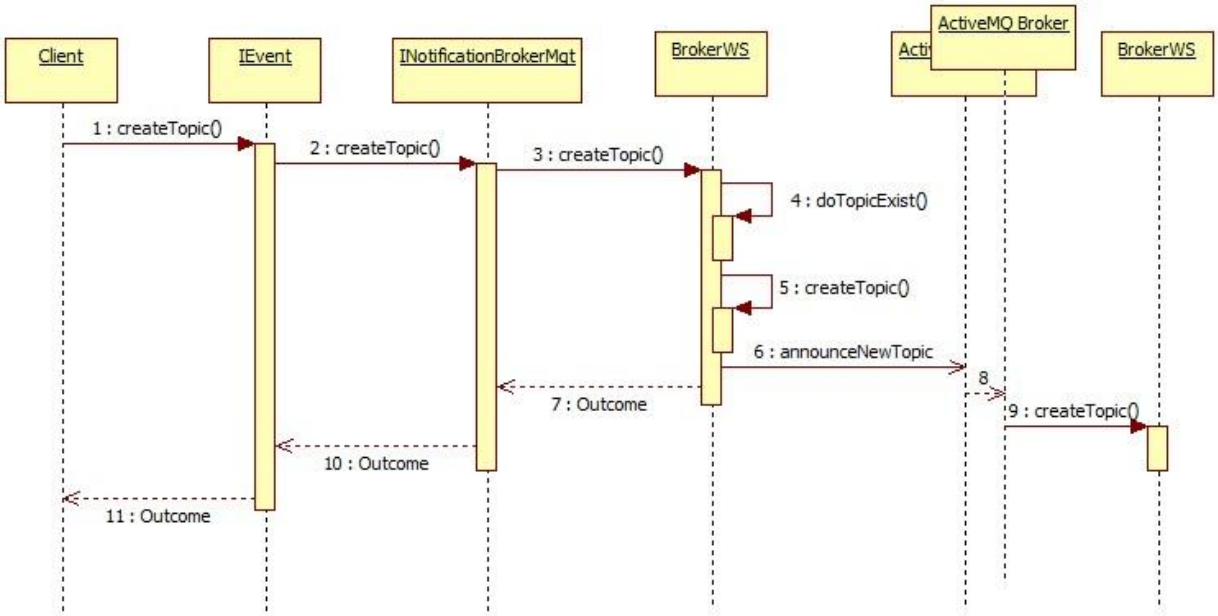
6.7.2 Distruzione di una federazione



La distruzione di una federazione si realizza con l'invocazione in successione di due metodi, come illustrato in figura:

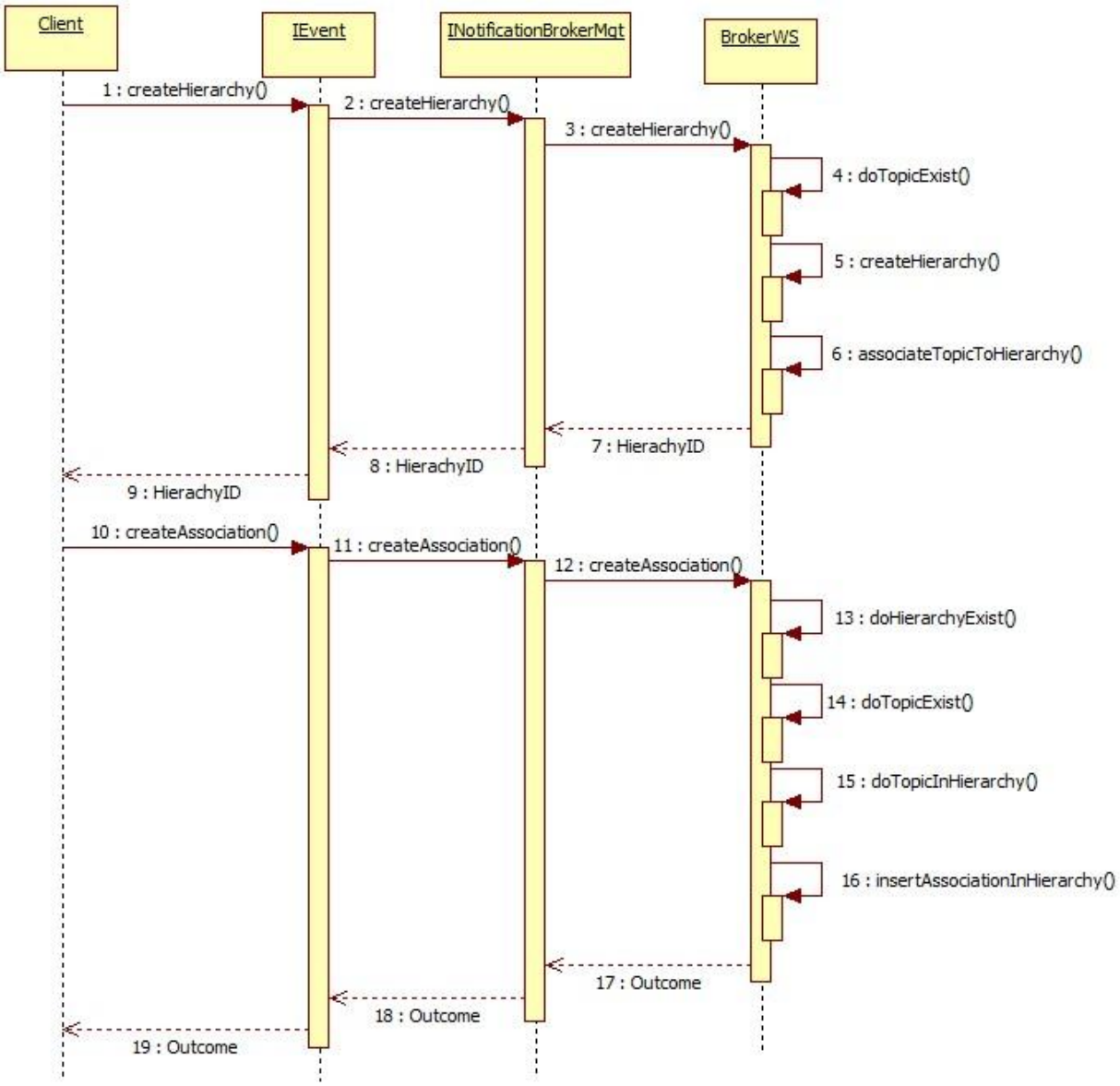
1. Scollegamento della federazione invocando *leaveFederation()*. Tale metodo arriva fino al servizio *BrokerWS*, che opera la distruzione dei connectors del Broker del middleware Active MQ di Apache;
2. Eliminazione della federazione creata invocando *dissolveFederation()*. Tale metodo viene passato a *BrokerWS* che opera la cancellazione di tutte le notifiche presenti sul Broker di Active MQ e la distruzione del broker.

6.7.3 Creazione di un topic e di una gerarchia



La creazione di un topic si realizza per mezzo del metodo *createTopic()*. *BrokerWS* controlla preventivamente se il topic richiesto sia già esistente, e in caso contrario opera la creazione. Se una federazione è attiva, allora questo topic viene annunciato sulla federazione, così che altre istanze di *BrokerWS* possono essere avvisati e creare a loro volta il nuovo topic.

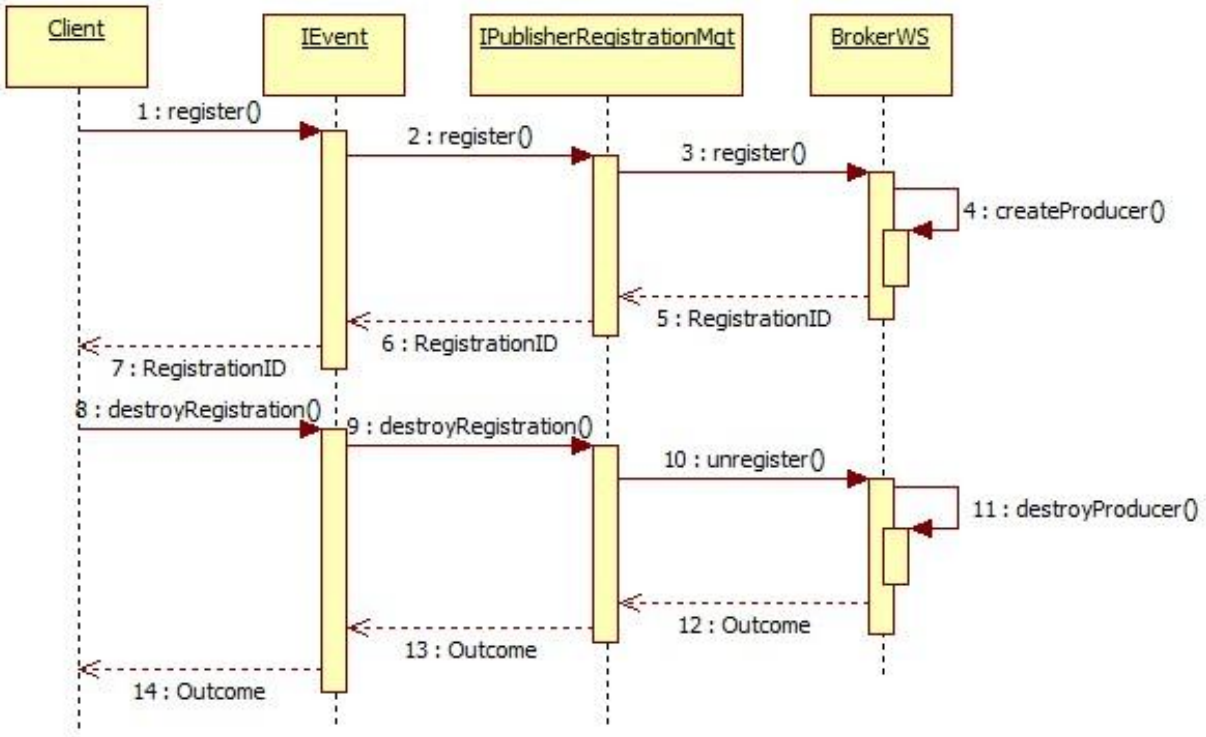
La creazione di una gerarchia si realizza con il metodo *createHierarchy()*. Tale metodo controlla se il topic radice indicato nell'invocazione esiste, solo in caso affermativo, la gerarchia viene creata e il topic associato ad esso. Il ritorno è un id univoco della gerarchia. Successivamente alla creazione di una gerarchia, bisogna definire una associazione al suo interno, invocando il metodo *createAssociation()*. *BrokerWS* controlla se il topic di partenza dell'associazione esiste nella gerarchia e se il topic di arrivo esiste, e solo in caso affermativo di entrabi i casi, l'associazione viene creata e inserita nella gerarchia, dove viene associato il topic di arrivo.



6.7.4 Annuncio di pubblicazione

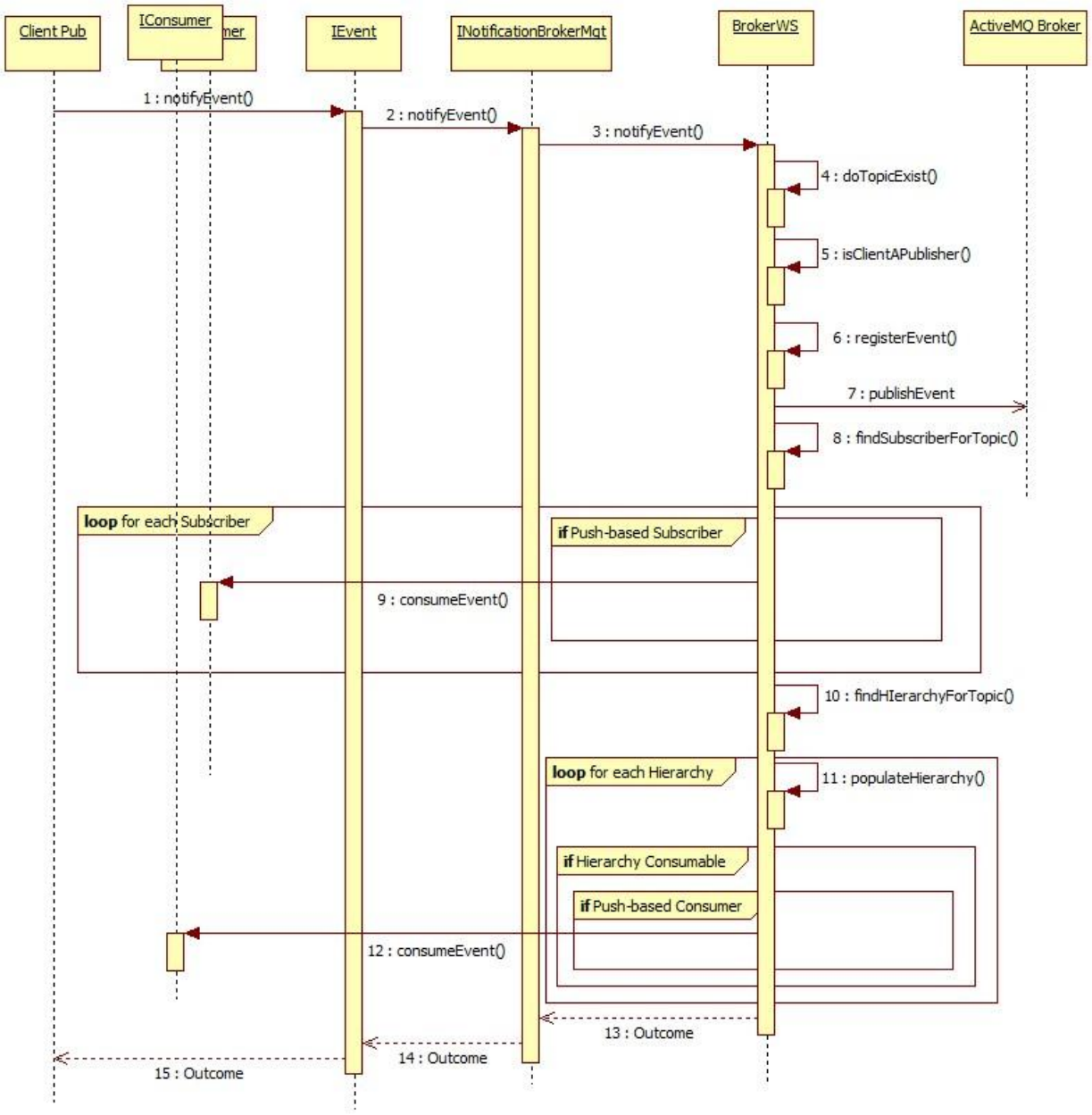
Quando un client intende assumere il ruolo di publisher per un dato topic o gerarchie deve effettuare una registrazione, invocando rispettivamente *registerTopics()* e *registerHierarchy()* (in figura è nostrato il funzionamento solo del primo visto che dal punto di vista funzionale i due metodo sono simili). Quando si invoca un metodo di registrazione, viene creato un Producer per i topic relativi (nel caso di una gerarchia per i topic che la compongono). *BrokerWS* costruisce un identificativo univoco di registrazione, che viene ritornato al client ripercorrendo la catena di invocazioni nell'architettura del *Gestore Gerarchico degli Eventi*. Tale identificativo può essere

usato per la distruzione di una registrazione con *destroyRegistration()*, che comporta la distruzione del producer.

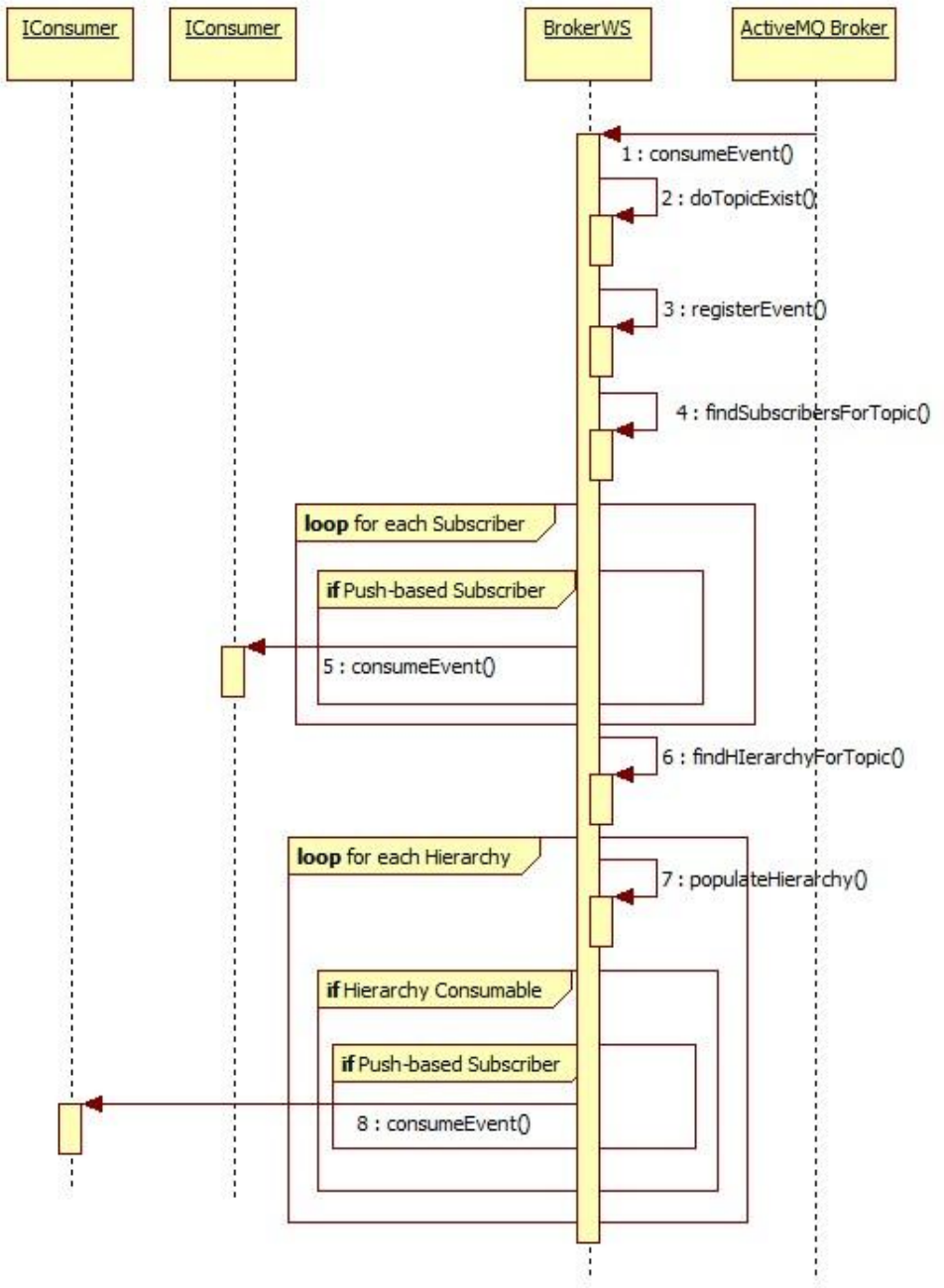


6.7.5 Pubblicazione di un evento

La pubblicazione di un evento si realizza per mezzo del metodo *notifyEvent()*. *BrokerWS* controlla preventivamente l'esistenza del topic a cui fa capo l'evento da pubblicare. Se esistente, si controlla se il richiedente è un Publisher per tale evento, ovvero ha preventivamente fatto un register. In caso positivo, l'evento siene accodato nella lista eventi del topic di appartenenza, ed eventualmente pubblicato su Active MQ se una federazione è stata definita. Successivamente si cercano i subscribers registrati per il dato topic, e se ce ne sono alcuni che hanno scelto una notifica push-based, questi vengono notificati invocando *consumeEvent()* sul loro servizio *IConsumer*.

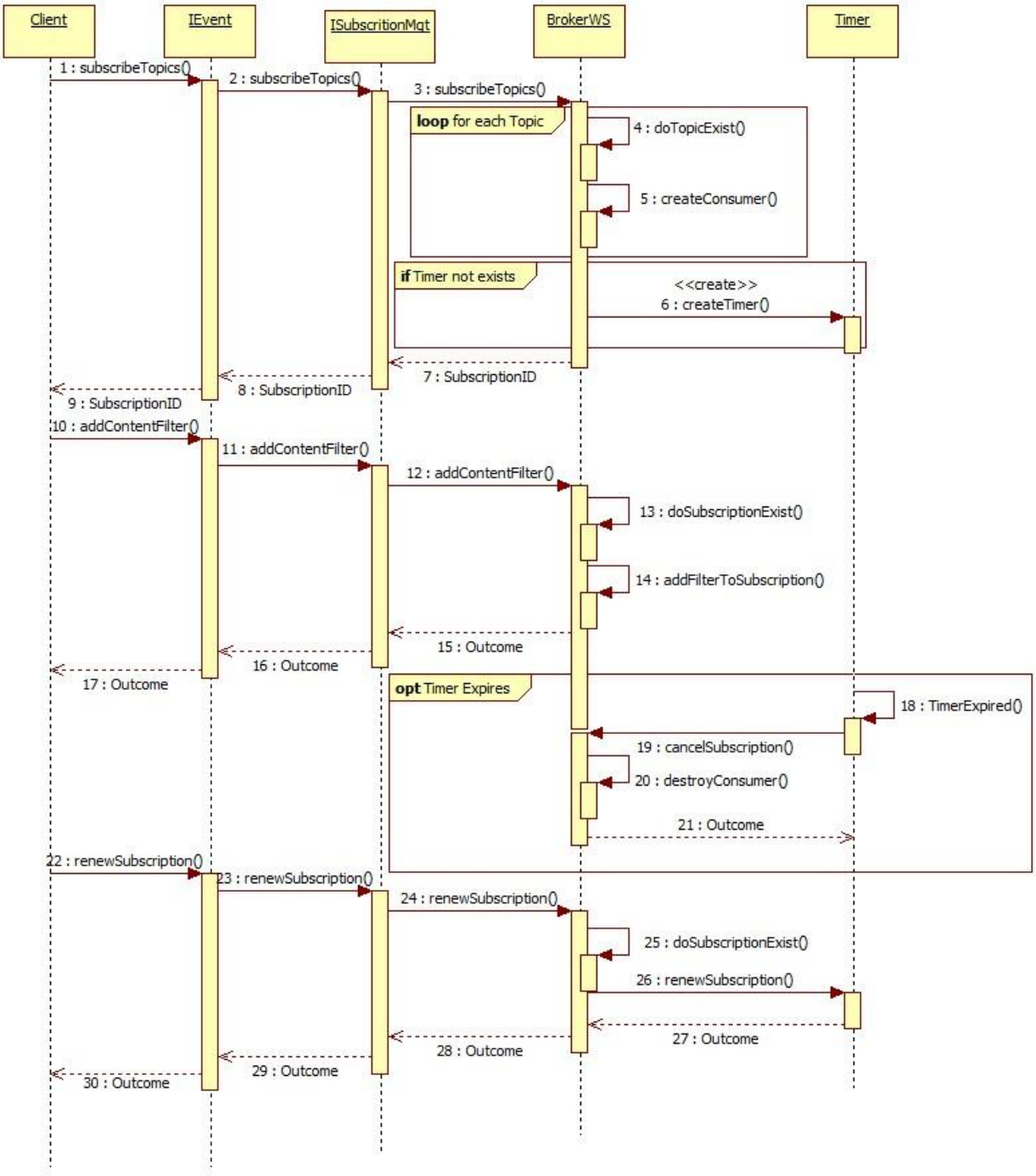


La publicazione di un evento non termina all'interno di un dato dominio, ma se una federazione è stata definita si propoaga in tutta la federazione toccando tutti i domini federati. Nello specifico, quando il Broker ActiveMQ riceve una notifica, questa viene passata a tutti i *BrokerWS* ad esso collegati e a tutti i Broker Active MQ federati che gireranno la notifica ai loro *BrokerWS*. In particolare se un Broker di Active MQ riceve una notifica, la gira a tutti i *BrokerWS* collegati, i quali ripercorrono gli stessi passi descritti precedentemente.



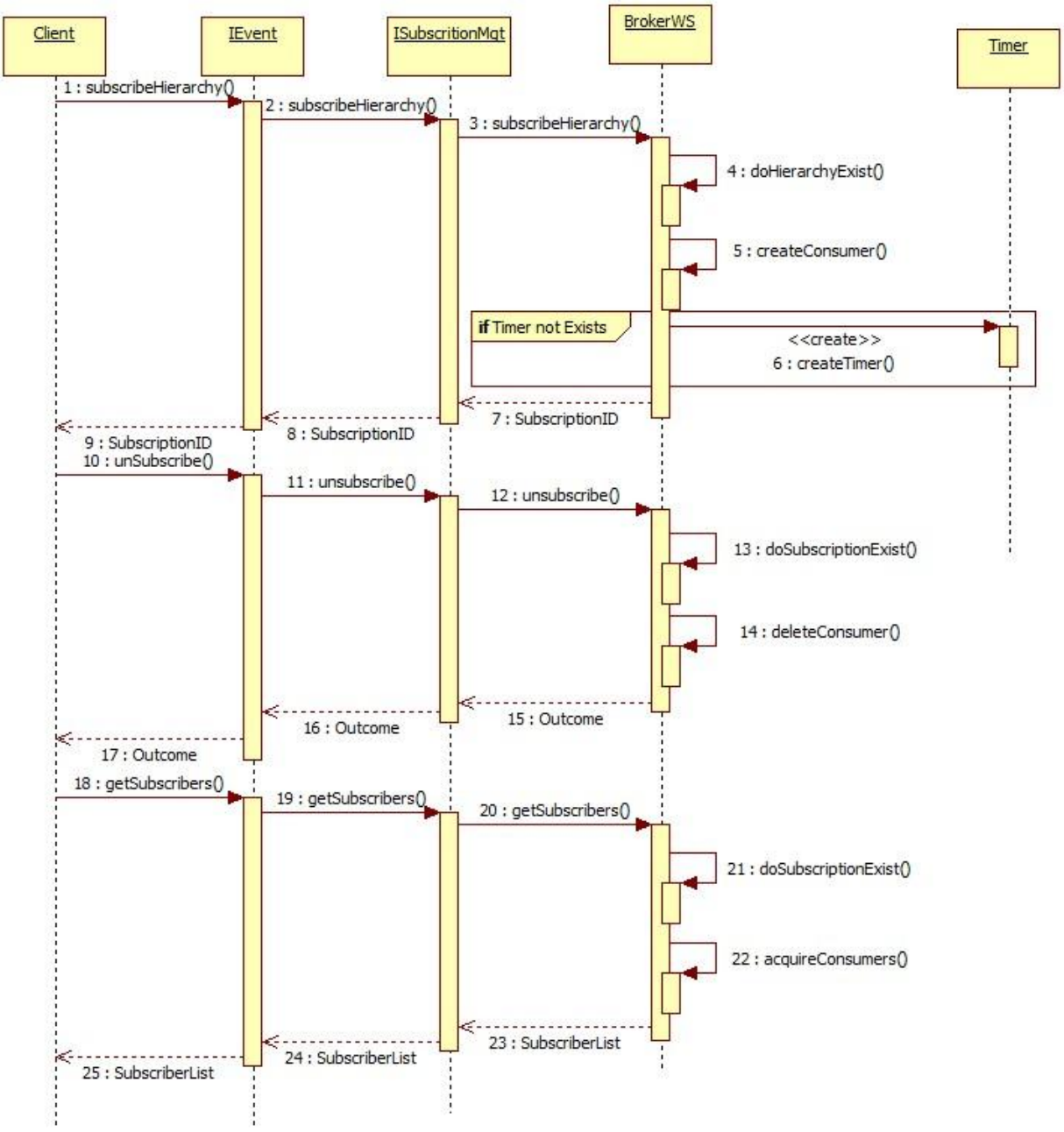
6.7.6 Sottoscrizione ad un topic o una gerarchia

La sottoscrizione a un topic viene realizzata invocando il metodo *subscribeTopics()*. *BrokerWS* per ogni topic indicato ne verifica l'esistenza, e in caso positivo crea un Consumer. Inoltre se il Timer non è esistente, ne viene creato uno e settato al tempo più basso di scadenza di una sottoscrizione. All'utente viene restituito un identificativo univoco della sottoscrizione.



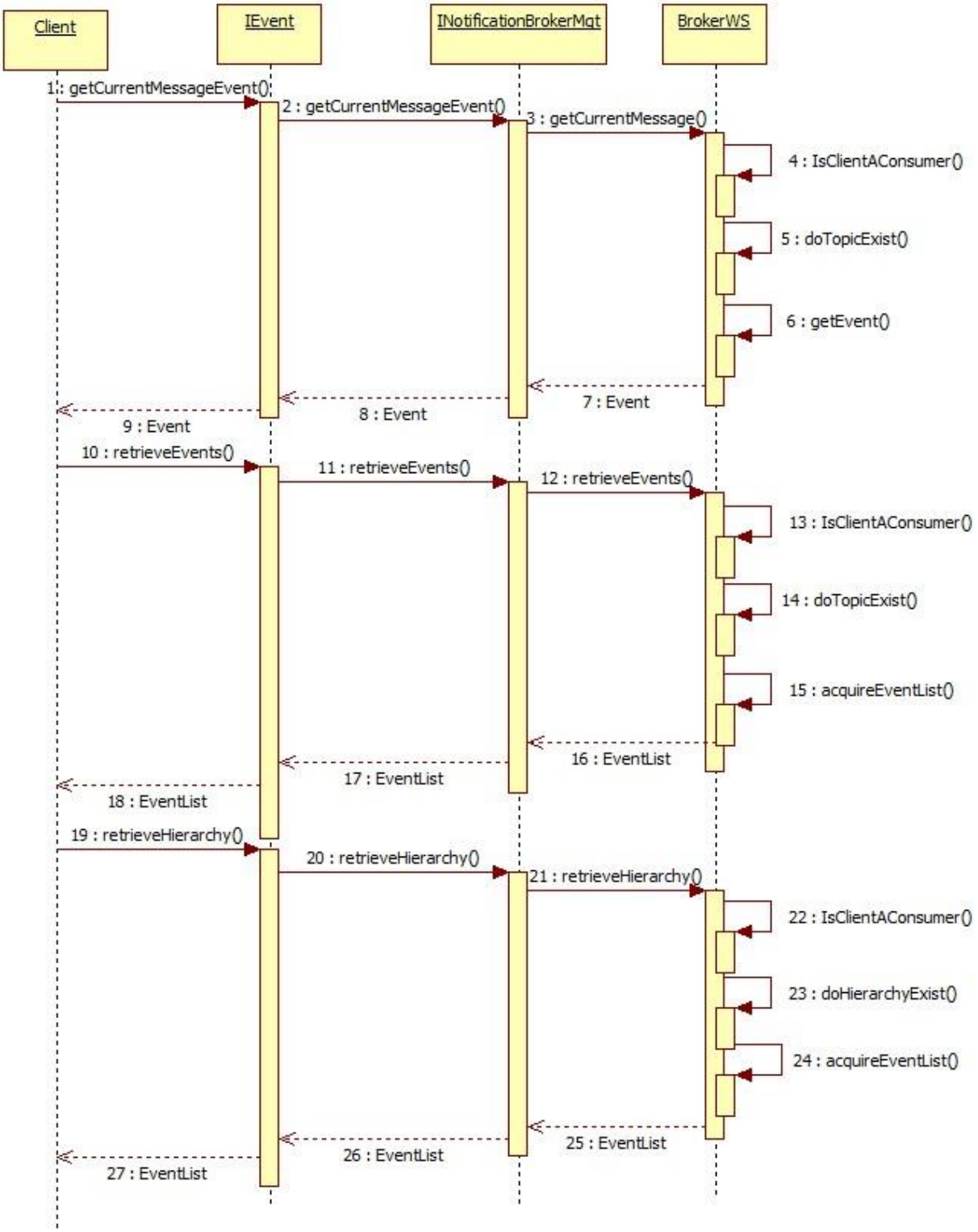
Tale identificativo viene usato per poter definire un filtro sul contenuto delle notifiche oppure per un successiva eliminazione della sottoscrizione. Nel caso in cui un Timer scade e non si è fatto una renew della sottoscrizione, allora in automatico *BrokerWS* procede alla cancellazione della sottoscrizione. L'invocazione di renewSubscription ha

l'effetto di reimpostare la scadenza della sottomisione e di riconfigurare il Timer con il valore della scadenza più bassa tra le sottoscrizioni attive.



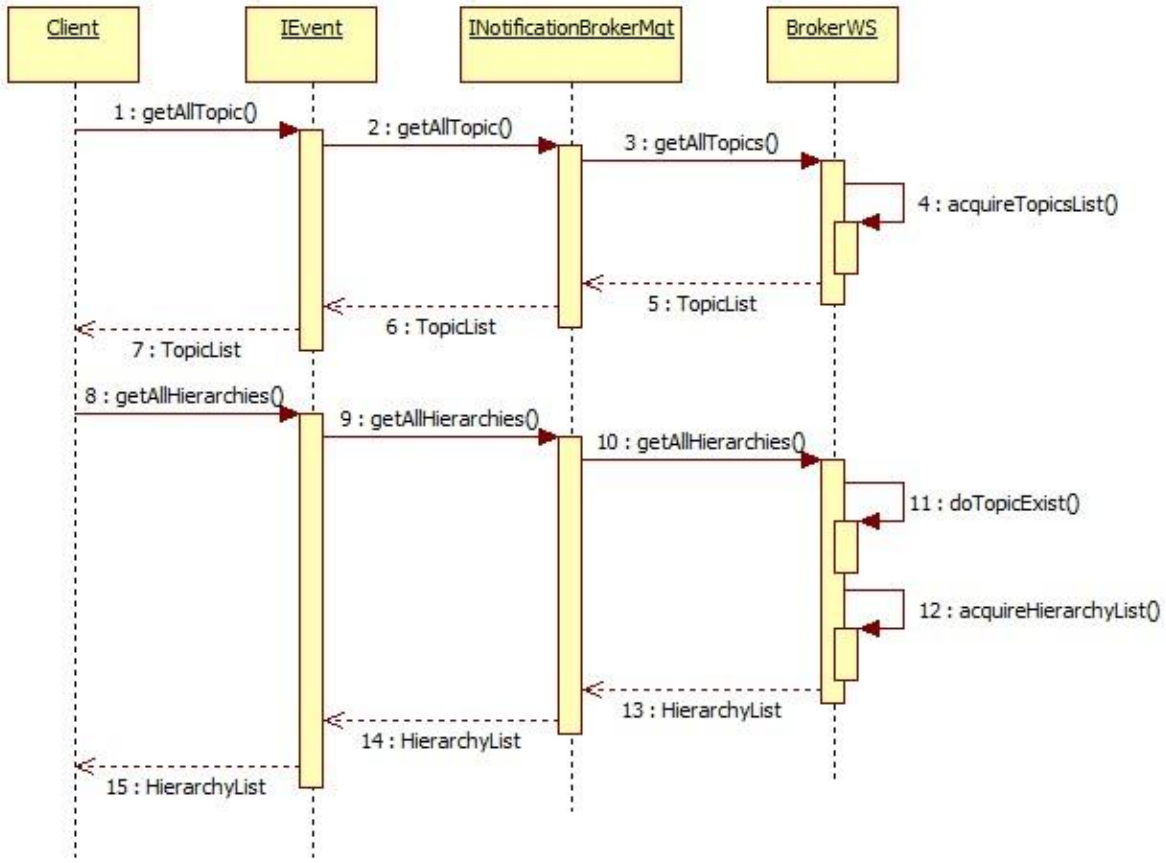
Il metodo *getSubscriber()* ha l'effetto di restituire all'utente la lista di tutti subscribers attualmente attivi su una data gerarchia.

6.7.7 Recupero di eventi



È possibile effettuare due diversi tipi di retrieval: ottenere l'ultimo evento pubblicato per un dato topic, se esistente, oppure ottenere tutti gli eventi pubblicati all'interno di un ben preciso intervallo temporale. Entrambe le operazioni sono possibili se il richiedente ha espresso un identificativo di sottoscrizione valido. La prima operazione è anche offerte per le gerarchie.

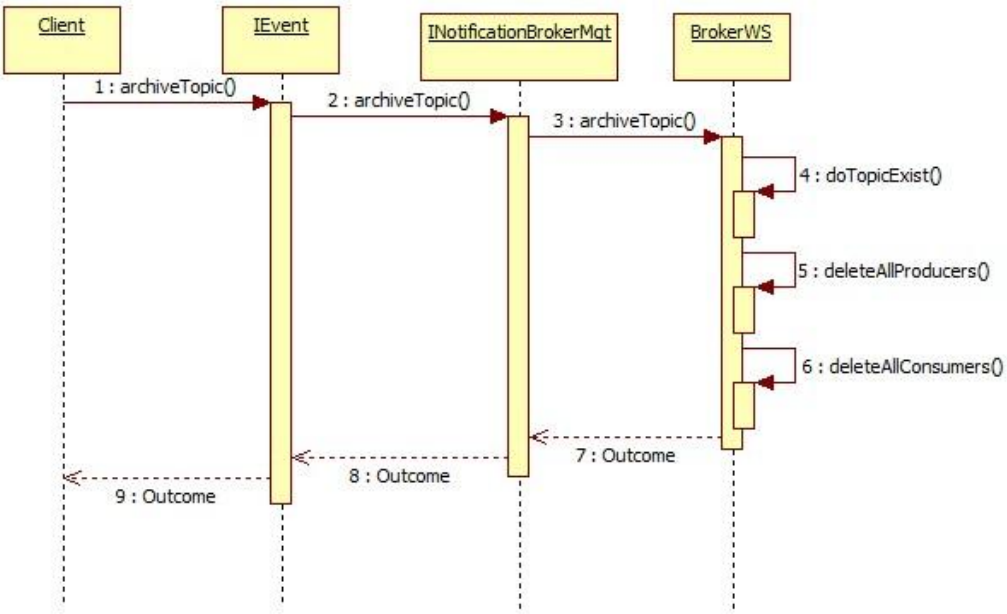
6.7.8 Ottenimento lista topics e gerarchie



L'utente può effettuare due operazioni di amministrazione: una prima per ottenere la lista di tutti i topic attualmente e una seconda per ottenere delle gerarchie esistenti sul *BrokerWS*.

6.7.9 Archiviazione di topics e gerarchie

L'archiviazione di un topic o di una gerarchia implica che tutti i produttori e i consumatori associati al topic o alla gerarchia così che nessuna successione invocazione dei metodi di pubblicazione ha un esito positivo. Il topic o la gerarchia può comunque essere consultata per accedere agli eventi precedenti all'istante di archiviazione.


Client
IEvent
INotificationBrokerMqt
BrokerWS
1 : archiveTopic()
2 : archiveTopic()
3 : archiveTopic()
4 : doTopicExist()
5 : deleteAllProducers()
6 : deleteAllConsumers()
7 : Outcome
8 : Outcome
9 : Outcome

7 Gestore delle Politiche di Accesso

La gestione delle politiche di accesso in OpenInFSE è basata sullo standard Web Services Security (WS-Security), estensione per i messaggi SOAP. Tale estensione prevede l'utilizzo delle asserzioni per la gestione degli accessi ai servizi.
Le possibili asserzioni considerate sono descritte di seguito. L'elenco degli attributi che compongono ogni asserzione è mostrato nella prossima tabella.

- L'asserzione di identità, che contiene informazioni circa l'identità dell'utente;
- L'asserzione di ruolo, che contiene informazioni inerenti al ruolo dell'utente e informazioni relative all'organizzazione a cui esso afferisce.
- L'asserzione di contesto, che contiene informazioni relative all'operazione che si vuole effettuare, ovvero lo scopo dell'accesso al servizio, il tipo di documento richiesto, e infine se per tale documento vi è o meno il consenso del paziente.
- L'asserzione di accesso, che consente di accedere ad un determinato servizio.

Attributo	Namespace	Tipo	Descrizione
Subject:subject-id	urn:oasis:names:tc:xacml:1.0:subject:subject-id"	Identità	Identificativo dell'utente
SubjectId	urn:oasis:names:tc:xacml:1.0:subject:subject-id	Ruolo	Identificativo dell'utente
Organization-id	urn:oasis:names:tc:xspa:1.0:subject:organization-id	Ruolo	Identificativo dell'azienda sanitaria o della regione presso cui l'utente è in carico
Organization	urn:oasis:names:tc:xspa:1.0:subject:organization	Ruolo	Descrizione dell'azienda sanitaria o della regione presso cui l'utente è in carico
Role	urn:oasis:names:tc:xacml:2.0:subject:role	Ruolo	Ruolo dell'utente
SubjectId	urn:oasis:names:tc:xacml:1.0:subject:subject-id	Contesto	Identificativo dell'utente
OrganizationId	urn:oasis:names:tc:xacml:1.0:resource:resource-id	Contesto	Identificativo dell'azienda sanitaria o della regione presso cui l'utente è in carico
Organization	urn:oasis:names:tc:xspa:1.0:subject:organization	Contesto	Descrizione dell'azienda sanitaria o della regione presso cui l'utente è in carico
EnvironmentLocality	urn:oasis:names:tc:xspa:1.0:environment:locality	Contesto	Posizione dalla quale l'utente opera (ospedale, studio medico, casa del paziente)
SubjectRole	urn:oasis:names:tc:xacml:2.0:subject:role	Contesto	Ruolo dell'utente
SubjectPurposeOfUse	urn:oasis:names:tc:xspa:1.0:subject:purposeofuse	Contesto	Destinazione d'uso della richiesta (es. Healthcare Treatment, Emergency Treatment)
ResourceType	urn:oasis:names:tc:xspa:1.0:resource:hl7:type	Contesto	Tipo della risorsa a cui si intende accedere (PatientSummary)

ResourceId	urn:oasis:names:tc:xacml:1.0:resource:resource-id	Contesto	Identificativo dell'assistito la cui risorsa si intende accedere (es codice fiscale dell'assistito X)
PatientConsent	urn:oasis:names:tc:xspa:1.0:resource:patient:consent	Contesto	Manifestazione del consenso puntuale ricevuto dal paziente
ResourceAction	urn:oasis:names:tc:xacml:1.0:action:action-id	Contesto	Azione che l'utente intende effettuare sulla risorsa
SubjectId	urn:oasis:names:tc:xacml:1.0:subject:subject-id	Accesso	Identificativo dell'utente
Role	urn:oasis:names:tc:xacml:2.0:subject:role	Accesso	Ruolo dell'utente

I casi d'uso tipici dell'utilizzo di tali asserzioni riguardano operazioni di query e di recupero del documento.

In particolare, la richiesta di query al servizio *IQueryMgt* (se configurato per gestire il controllo degli accessi) dovrà contenere le seguenti tre asserzioni:

- asserzione di identità
- asserzione di ruolo
- asserzione di contesto

Inoltre, il servizio *IQueryMgt* sarà caratterizzato dalle presenza della componente *PEPComponent*, che realizza i dovuti controlli sulle asserzioni prima di fornire eventualmente l'accesso al servizio. Nel caso in cui la componente PEP fornisca l'accesso al servizio, restituirà all'utente l'asserzione di accesso al servizio *IDocument*.

In caso di recupero del documento, la richiesta al servizio *IDocument* (se configurato per gestire il controllo degli accessi), dovrà comprendere l'asserzione di accesso.

La componente *Gestore delle Politiche di Accesso* si compone delle seguenti sotto-componenti:

- *PEPComponent*;
- *IHandler*;
- *IPDPComponent*.

7.1 Componente PEPComponent

È una componente del sistema, indipendente dal servizio OpenInFSE che deve interfacciare. Essa effettua il controllo sugli accessi, realizza la richiesta di decisione verso il PDP e fa rispettare tale decisione.

Il *PEPComponent*, dislocato presso ogni servizio, ha il compito di intercettare la richiesta per effettuare le seguenti verifiche:

- analizzare la correttezza della struttura delle asserzioni;
- verificare la firma digitale contenuta in ogni asserzione;
- verificare la coerenza dell'asserzione di contesto con il messaggio applicativo;
- verificare la correttezza del ruolo e del purpose of use;
- verificare il consenso del cittadino, interagendo con il sistema di gestione del consenso utilizzato nello specifico dominio regionale.

In particolare, il *PEPComponent* riceve il messaggio SOAP, costituito da un header e un body; nell'header vi è la componente WS-Security caratterizzata da una o più asserzioni.

Il *PEPComponent* controlla la struttura dell'header del messaggio e il contenuto delle singole asserzioni. Si mostrano di seguito due esempi:

- la richiesta di *query* al servizio *IQueryMgt*;
- la richiesta di *retrieveDocument* al servizio IDocument.

L'header del messaggio SOAP, in caso di query, deve contenere tre diverse asserzioni: l' asserzione di identità, l'asserzione di ruolo e l'asserzione di contesto.

In caso di richiesta di *retrieveDocument*, l'header del messaggio deve contenere unicamente l'asserzione di accesso.

7.1.1 Controlli effettuati sull'asserzione di identità

Si elencano i controlli effettuati in caso di presenza dell'asserzione di identità:

- Si effettua un controllo della struttura, ovvero si verifica la presenza dell'attributo *subject:subject-id* (ovvero l'identificativo dell'utente che vuole accedere alle informazioni relative al documento richiesto) nell'asserzione;
- Si effettua un controllo della signature associata all'asserzione, tramite le seguenti operazioni :
    - **validateXMLSignature**: con la seguente operazione si valuta dapprima che l'algoritmo usato per realizzare la firma sia non nullo e inoltre che l'algoritmo di canonicalization usato per creare la firma non è nullo.
    - **validateSignatureProfile**: tale operazione verifica che l'oggetto signature sia un oggetto XMLSignature, e che sia caratterizzato da una struttura conforme. Dopodiché valida i riferimenti, le URI dei riferimenti e l'algoritmo utilizzato per la codifica della firma. Infine valuta che la firma sia caratterizzata da un periodo di validità temporale valido.

- **validateSignatureCredential**: con questa operazione si verifica innanzitutto che sono disponibili le chiavi (pubbliche) della firma, inoltre essa preleva l'algoritmo che ha generato le chiavi e valida la firma tramite le chiavi ottenute.

- Si realizza un controllo sul valore di determinati attributi:
  - si controlla che il codice fiscale dell'utente sia ben formato e sia inoltre lo stesso nelle altre asserzioni presenti nell'header
- Si verifica la presenza del consenso da parte del paziente;
- Si effettua il controllo relativo alla validazione temporale:
  - viene preso l'istante temporale in cui si riceve il messaggio e si verifica che questo appartenga all'intervallo temporale di validità che viene calcolato opportunamente in funzione al periodo di validità temporale indicato dall'asserzione e ad un periodo massimo di validità configurabile.

7.1.2 Controlli effettuati sull'asserzione di ruolo

Si elencano i controlli effettuati in caso di presenza dell'asserzione di ruolo:

- Si effettua un controllo sulla struttura dell'asserzione, ovvero si controlla la presenza dei seguenti attributi:
  - *urn:oasis:names:tc:xacml:1.0:subject:subject-id* (identificativo dell'utente)
  - *urn:oasis:names:tc:xspa:1.0:subject:organization-id* (identificativo della organizzazione che fornisce il ruolo all'utente)
  - *urn:oasis:names:tc:xspa:1.0:subject:organization* (descrizione dell'organizzazione)
  - *urn:oasis:names:tc:xacml:2.0:subject:role* (indica il ruolo dell'utente)
- Si realizza il controllo della signature associata all'asserzione:
  - **validateXMLSignature**: con il seguente metodo si valuta innanzitutto che l'algoritmo usato per realizzare la firma sia non nullo, inoltre che l'algoritmo di canonicalization usato per creare la firma non è vuoto;
  - **validateSignatureProfile**, questa operazione verifica che l'oggetto Signature sia un oggetto XMLSignature, e che sia caratterizzato da una struttura conforme. Dopodichè si passa a validare i riferimenti, le URI dei riferimenti e l'algoritmo utilizzato per la codifica della firma. Infine verifiche che la firma sia caratterizzata da un periodo di validità temporale adeguato;

- **validateSignatureCredential**, tale opeazione verifica innanzitutto che sono disponibili le chiavi (pubbliche) della firma, dopodiché preleva l'algoritmo per la costruzione delle chiavi, valida la firma, e infine verifica la validità del certificato, verificando che il certificato con il quale si è firmato appartenga ad una CA trusted.

- Si effettua il controllo sul valore di determinati attributi, ovvero:
  - si verifica che il valore del *organizationid* è opportuno;
  - si controlla che il ruolo dell'utente indicato è tra quelli validi;
  - si confronta l'identificativo dell'utente con quello indicato nelle altre asserzioni.

- Si verifica la presenza del consenso da parte del paziente;

- Infine si effettua il controllo relativo alla validazione temporale:
  - viene preso l'istante temporale in cui si riceve il messaggio e si verifica che questo appartenga all'intervallo temporale di validità che viene calcolato opportunamente in funzione al periodo di validità temporale indicato dall'asserzione e ad un periodo massimo di validità configurabile.

7.1.3 Controlli effettuati sull'asserzione di contesto

Si elencano i controlli effettuati in caso di presenza dell'asserzione di contesto:

- Si effettua il controllo della struttura dell'asserzione, ovvero si controlla la presenza degli attributi:
  - *urn:oasis:names:tc:xacml:1.0:subject:subject-id*
  - *urn:oasis:names:tc:xspa:1.0:subject:organization-id*
  - *urn:oasis:names:tc:xspa:1.0:subject:organization*
  - *urn:oasis:names:tc:xacml:2.0:subject:role*
  - *urn:oasis:names:tc:xspa:1.0:subject:purposeofuse*
  - *urn:oasis:names:tc:xspa:1.0:resource:hl7:type*
  - *urn:oasis:names:tc:xacml:1.0:resource:resource-id*
  - *urn:oasis:names:tc:xspa:1.0:resource:patient:consent*
  - *urn:oasis:names:tc:xspa:1.0:environment:locality*
  - *urn:oasis:names:tc:xacml:1.0:action:action-id*
- Si realizza il controllo della signature associata all'asserzione, tramite le seguenti operazioni :

    - **validateXMLSignature**, con la seguente operazione si valuta dapprima che l'algoritmo usato per la firma sia non nullo, inoltre si verifica che l'algoritmo di canonicalization usato per creare la firma non è vuoto.
    - **validateSignatureProfile**, tale opearzione verifica che l'oggetto Signature sia un oggetto XMLSignature e caratterizzato da una struttura conforme. Dopodiché si passa a validare i riferimenti, le URI dei riferimenti e si verifica la presenza dell'algoritmo utilizzato per la codifica della firma. Infine si controlla che la firma sia caratterizzata da un periodo di validità temporale valdio.
    - **validateSignatureCredential**, tale opeazione verifica innanzitutto che sono disponibili le chiavi (pubbliche) della firma, dopodiché individua l'algoritmo per la generazione delle chiavi e valida la firma tramite le chiavi ottenute.
- Si effettua il controllo sul valore di determinati attributi:
    - si verifica che il valore del organizationid è opportuno
    - si controlla che il purpose of use indicato è presente tra quelli ammissibili
    - si verifica infine l'identificativo dell'utente
- Si verifica la presenza del consenso da parte del paziente;
- Si effettua il controllo relativo alla validazione temporale:
    - viene preso l'istante temporale in cui si riceve il messaggio e si verifica che questo appartenga all'intervallo temporale di validità che viene calcolato opportunamente in funzione al periodo di validità temporale indicato dall'asserzione e ad un periodo massimo di validità configurabile.
- Infine si realizza il confronto tra dati presenti nell'header e quelli del body:
    - si controlla che l'identificativo del paziente indicato nell'asserzione di contesto sia effettivamente l'identificativo univoco richiesto per la query dei documenti, ovvero quello presente nel body del messaggio.

7.1.4 Controlli effettuati sull'asserzione di accesso

Si elencano i controlli effettuati in caso di presenza dell'asserzione di contesto:

- Si realizza il controllo della struttura dell'asserzione, ovvero si controlla la validità temporale dell'asserzione;
- Si effettua la verifica della signature associata all'asserzione tramite le seguenti operazioni:
    - **validateXMLSignature**, con il seguente metodo si valuta in primo luogo che l'algoritmo usato per effettuare la firma sia non nullo e inoltre che l'algoritmo di canonicalization usato per creare la firma non sia

vuoto.
    - **validateSignatureProfile**, tale operazione verifica che l'oggetto Signature sia un oggetto XMLSignature e caratterizzato da una struttura conforme. Dopodiché si validano i riferimenti, le URI dei riferimenti e l'algoritmo utilizzato per la codifica della firma. Infine si verifica la validità temporale della firma
    - **validateSignatureCredential**, con questa operazione si verifica se sono disponibili le chiavi (pubbliche) della firma, dopodiché si individua l'algoritmo per la costruzione delle chiavi e si valida la firma tramite tali chiavi.

- Si realizzano controlli sui valori di determinati attributi, vale a dire sul :
    - ruolo dell'utente
    - identificativo univoco del medico

- Si effettua la validazione temporale dell'asserzione, ovvero:
    - Viene preso l'istante temporale in cui si riceve il messaggio e si verifica che questo appartenga all'intervallo temporale di validità che viene calcolato opportunamente in funzione al periodo di validità temporale indicato dall'asserzione e ad un periodo massimo di validità configurabile.

- Infine si realizza il confronto tra determinati dati presenti nell'header e quelli del body, vale a dire:
    - si verifica che l'identificativo del documento sul quale si riceve l'autorizzazione di accesso è lo stesso di quello contenuto nel body del messaggio;
    - si controlla che la decisione comunicata dal PDP sull'asserzione di accesso sia PERMIT;
    - Si accerta che l'autorizzazione è fornita al servizio opportuno per la realizzazione della retrieve (*IDocument*);
    - Si controlla che IDOCUMENT_ENDPOINT a cui è stata fornita l'autorizzazione corrisponda al valore dell'elemento IDOCUMENT_ENDPOINT presente sul nodo *IDocument*.

7.2 Componente IHandler

Si tratta di una libreria di handler SOAP realizzata per la gestione della propagazione dell'header SOAP tra un servizio OpenInFSE ed un altro. Questi handler rappresentano dei filtri per tutti i messaggi in ingresso/uscita dal servizio su cui sono definiti.

7.3 Servizio IPDPComponent

Questo Web Service riceve una richiesta di accesso dalla componente *PEPComponent*, e ha l'obiettivo di autorizzare o meno l'accesso al servizio consultando opportune politiche di accesso basate sul ruolo. Tale componente inizializza il motore di access control acquisendo regole XACML da una serie di file xml e, in base ad esse e alla richiesta, il servizio prende una decisione di tipo permit o deny.

7.4 Esempi di configurazione

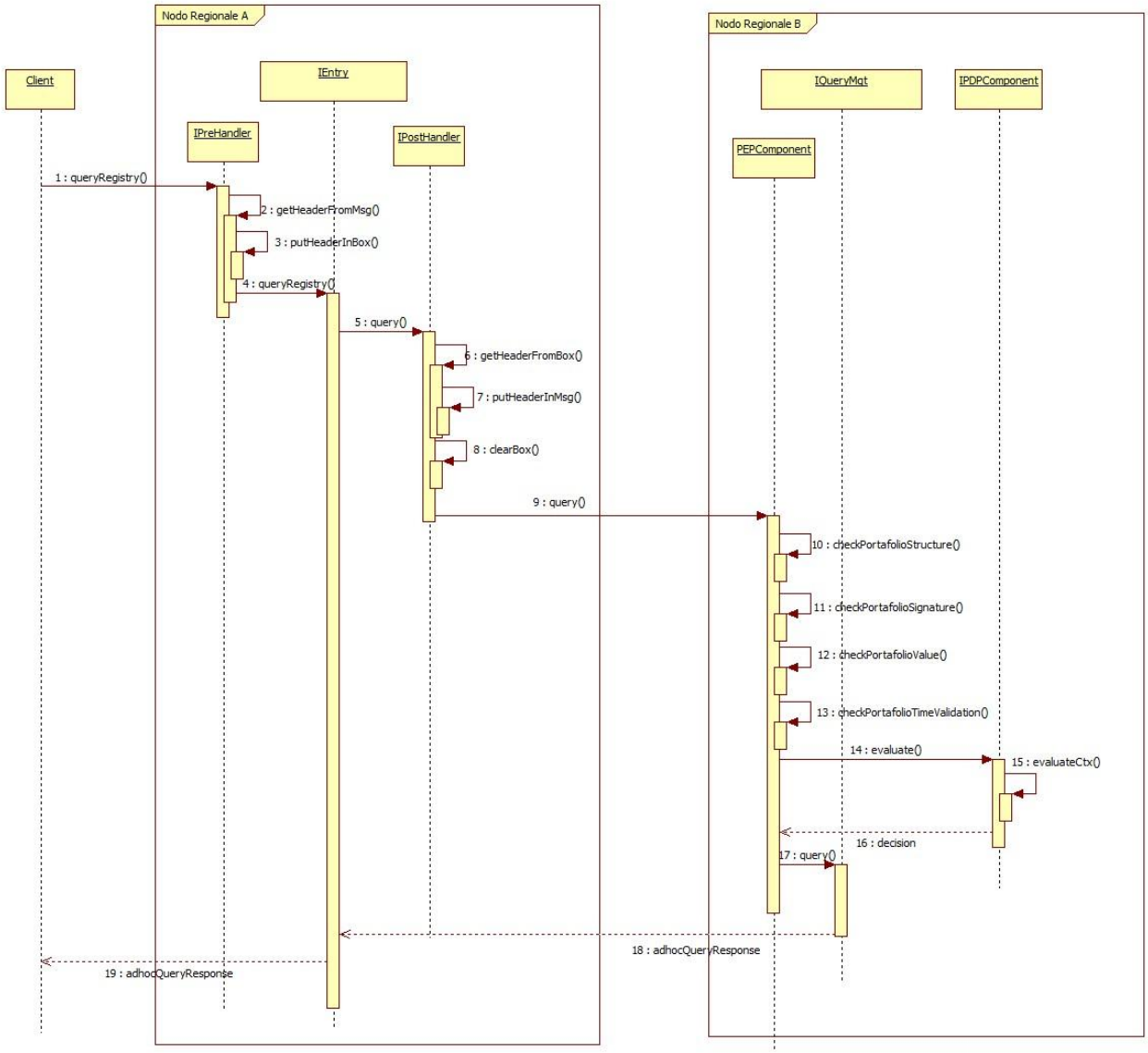
7.4.1 Esempio di query

Si mostra un esempio di configurazione per l'operazione di query extra-dominio non federata.

Come è mostrato nel sequence diagram della prossima figura, il servizio *IEntry* è configurato in modo tale da propagare le asserzioni ricevute nei messaggi di ingresso mediante due handler: IPreHandler e IPostHandler. Infatti questo servizio dell'*Interfaccia di Accesso* deve inoltrare il messaggio di richiesta, contenente le asserzioni (ricevute dal client) al servizio *IQueryMgt*.

La componente *IPreHandler* si occupa di memorizzare l'header del messaggio (contenente appunto le asserzioni), che verrà poi recuperato dalla componente *IPostHandler* che provvede ad aggiungerlo al messaggio da inviare a *IQueryMgt*.

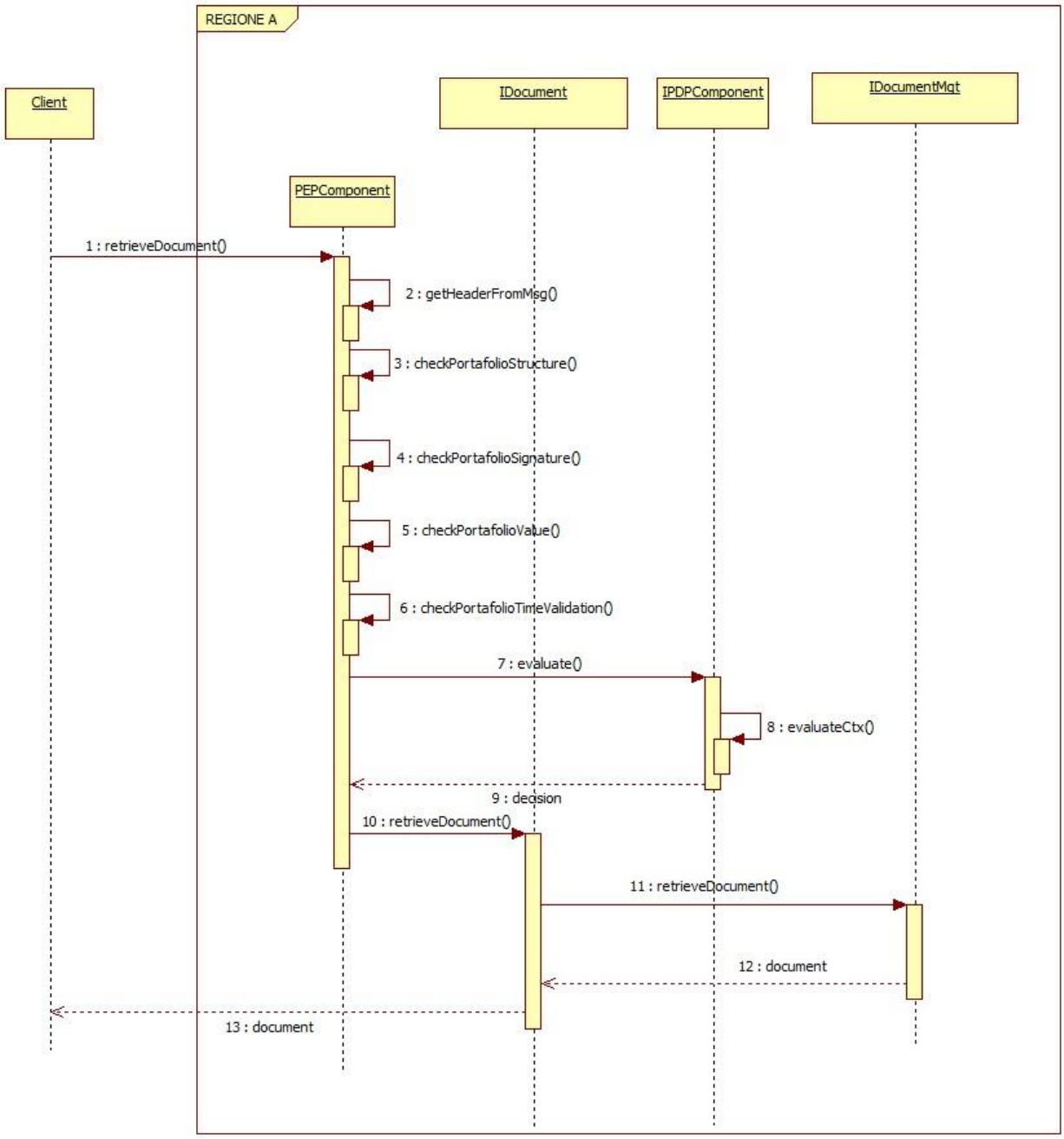
Sul servizio *IQueryMgt* è presente invece la componente *PEPComponent* che provvede ad autorizzare o meno l'accesso al servizio tramite i controlli evidenziati in precedenza e alla risposta del servizio *IPDPComponent*.


Nodo Regionale A
Nodo Regionale B
Client
IEntry
IPreHandler
IPostHandler
IQueryMgt
IPDPComponent
PEPComponent
1 : queryRegistry()
2 : getHeaderFromMsg()
3 : putHeaderInBox()
4 : queryRegistry()
5 : query()
6 : getHeaderFromBox()
7 : putHeaderInMsg()
8 : clearBox()
9 : query()
10 : checkPortafolioStructure()
11 : checkPortafolioSignature()
12 : checkPortafolioValue()
13 : checkPortafolioTimeValidation()
14 : evaluate()
15 : evaluateCtx()
16 : decision
17 : query()
18 : adhocQueryResponse
19 : adhocQueryResponse

7.4.2 Esempio di recupero documento

Si mostra un ulteriore esempio di configurazione per quanto concerne l'opeazione di recupero documento.

Come è mostrato nella prossima figura, il servizio *IDocument* è configurato in modo da utilizzare la componente *IPEPComponent*. Infatti, il servizio *IDocument* prima di fornire l'accesso ai servizi deve verificare se l'utente ha l'autorizzazione, tramite la componente *IPEPComponent* ed il servizio *IPDPComponent*.


REGIONE A
Client
IDocument
IPDPComponent
IDocumentMgt
PEPComponent
1 : retrieveDocument()
2 : getHeaderFromMsg()
3 : checkPortafolioStructure()
4 : checkPortafolioSignature()
5 : checkPortafolioValue()
6 : checkPortafolioTimeValidation()
7 : evaluate()
8 : evaluateCtx()
9 : decision
10 : retrieveDocument()
11 : retrieveDocument()
12 : document
13 : document