Manuale SDK di OpenSPCoop2

Copyright © 2005-2013 *Link.it srl*

Indice

Elenco delle figure

Elenco delle tabelle

1 Introduzione

OpenSPCoop2 è un prodotto software, discendente diretto della Porta di Dominio OpenSPCoop, in grado di gestire, trasformare e smistare messaggi SOAP in base ad informazioni presenti in un registro.

Tra le principali novità troviamo un'architettura applicativa completamente reingegnerizzata, nuove funzionalità tramite supporto nativo di standard quali WS-Security per la gestione della sicurezza, estensione della compatibilità con ulteriori piattaforme RDBMS (HSQL e SQLServer) e totale personalizzazione del protocollo di cooperazione. Proprio su quest'ultima funzionalità è incentrato il presente documento.

A differenza di OpenSPCoop, nato e pensato nell'ottica del protocollo SPCoop, OpenSPCoop2 prevede che il protocollo di gestione del flusso di cooperazione sia modificabile tramite l'installazione di un plugin.

Nel seguito il manuale affronterà i seguenti temi:

- Presentazione dei concetti sui quali si basa l'astrazione del protocollo di cooperazione;
- Documentazione del Software Development Kit (brevemente SDK), incluso in OpenSPCoop2, per la realizzazione dei plugin relativi ai protocolli di cooperazione;
- Descrizione della procedura di installazione e configurazione di un plugin in OpenSPCoop2;
- Presentazione di un esempio relativo all'implementazione di un plugin con relativa installazione.

2 La Personalizzazione del Protocollo di Cooperazione

L'adozione di un protocollo di cooperazione ad-hoc si traduce tipicamente, per ogni messaggio gestito dalla Porta di Dominio, nella gestione personalizzata di un *header SOAP* o di trasporto http e della tracciatura delle comunicazioni.

Il processo di adozione di un nuovo protocollo di cooperazione in OpenSPCoop2 prevede essenzialmente i seguenti passi:

1. Implementazione della logica del protocollo di cooperazione che consiste nella realizzazione di un insieme di classi Java aderenti ad interfacce standard definite dal SDK di OpenSPCoop2 e raggruppate in un archivio jar che, nel suo insieme, rappresenta il *plugin* che implementa il nuovo protocollo;
2. Installazione del nuovo plugin in OpenSPCoop2 e sua configurazione;
3. Configurazione di nuovi accordi di servizio e di servizi da fruire o erogare in accordo al nuovo protocollo.

Ogni protocollo di cooperazione si distingue dagli altri per i comportamenti assunti relativamente ai seguenti aspetti di processamento delle richieste di servizio:

1. La modalità con cui vengono veicolate le informazioni riguardanti il servizio e i soggetti all'interno del messaggio di cooperazione. Le scelte, come si vedrà negli esempi seguenti, possono riguardare la modifica del messaggio (ad es. l'aggiunta di un Header SOAP), la modifica dell'header di trasporto (ad es. l'aggiunta di header HTTP proprietari) o la modifica della URL usata per l'invocazione dei servizi. È anche possibile, come caso particolare, utilizzare un protocollo completamente trasparente che non richiede quindi nessuna informazione aggiuntiva nei messaggi scambiati.
2. La serializzazione xml dei dati di tracciamento. OpenSPCoop2 fornisce un'implementazione di default definita da uno schema xsd. Un plugin di protocollo può eventualmente decidere di re-implementare la serializzazione in base alle proprie esigenze.
3. La serializzazione xml dei messaggi diagnostici. OpenSPCoop2 fornisce anche per i diagnostici un'implementazione di default definita da uno schema xsd. Un plugin di protocollo può eventualmente decidere di re-implementare la serializzazione in base alle proprie esigenze.
4. La generazione di messaggi di errore applicativi da restituire ai servizi applicativi. Come per le tracce e i diagnostici viene fornita un'implementazione di default e resta a carico dello sviluppatore del plugin di protocollo ridefinirne una ad hoc.

3 Il Software Development Kit

Lo sviluppo di un plugin di protocollo è possibile tramite il Software Development Kit di OpenSPCoop2 costituito dal package java *org.openspcoop2.protocol.sdk*.

Accanto al SDK viene fornito, con il package *org.openspcoop2.protocol.basic*, un kit di classi che contengono implementazioni di default basate sul SDK. Grazie a questo package lo sviluppatore può semplificare il proprio lavoro adottando o estendendo tali classi per la realizzazione del proprio plugin.

Per meglio chiarire, prendiamo come esempio la realizzazione del meccanismo di generazione delle tracce su database. Si hanno essenzialmente due strade da percorrere:

1. Fornire un'implementazione da zero realizzando una classe che implementi l'interfaccia *org.openspcoop2.protocol.sdk.tracciamen*
2. Fornire un'implementazione estendendo la classe *org.openspcoop2.protocol.basic.tracciamento.TracciamentoOpenSPCoopAppen* facente parte del pacchetto di utilità di OpenSPCoop2.

Il primo passo che logicamente il programmatore deve svolgere per avviare l'implementazione del plugin è quello di realizzare una classe *Factory* alla quale dovranno essere agganciati tutti gli elementi che compongono la logica del protocollo. Tale factory dovrà, in fase di confezionamento del plugin, essere riferita nel file di configurazione da associare alla libreria jar.

Per la realizzazione della Factory, analogamente all'esempio descritto poc'anzi, si possono percorrere due strade:

1. Fornire un'implementazione da zero realizzando una classe che implementi l'interfaccia *org.openspcoop2.protocol.sdk.IProtocolF*
2. Fornire un'implementazione estendendo la classe astratta *org.openspcoop2.protocol.basic.BasicFactory*.

Partendo dall'interfaccia *IProtocolFactory*, lo sviluppatore potrà fornire una propria implementazione di tutte o di alcune delle classi necessarie per la gestione del protocollo. La factory contiene i metodi per istanziare i gestori di tutte le funzionalità di OpenSPCoop2 e quindi:

- Gestione dei payload (Imbustamento/Sbustamento)
- Validazione dei payload
- Emissione messaggi diagnostici
- Tracciatura
- Gestione della configurazione

Vediamo adesso una descrizione dettagliata di quanto contenuto, sezione per sezione, nella factory da realizzare.

3.1 Gestione dei payload

La factory consente alla Porta di Dominio di istanziare le classi per intervenire sui messaggi in transito al fine di supportare le funzionalità previste dal protocollo.

Alcuni protocolli potrebbero gestire un header SOAP aggiuntivo, altri invece utilizzare l'header di trasporto o altri ancora non prevedere alcun header. La gestione dei payload prende il nome di *Imbustamento*, quando la Porta di Dominio ha ruolo di fruitore, e di *Sbustamento* quando la Porta di Dominio ha ruolo di erogatore.

La Factory prevede i seguenti metodi:

- ```
  public IBustaBuilder createBustaBuilder()
  ```

  Consente di ottenere un'istanza per gestire imbustamento/sbustamento dei messaggi in transito sulla Porta di Dominio. Si deve fornire un'implementazione dell'interfaccia *IBustaBuilder* o in alternativa estendere la classe *BustaBuilder* del package basic.
  ```

- ```
  public IErroreApplicativoBuilder createErroreApplicativoBuilder()
  ```

  Consente di ottenere un'istanza per generare il messaggio di errore applicativo da inviare ai servizi applicativi nei casi di errore. Si deve fornire un'implementazione dell'interfaccia *IErroreApplicativoBuilder* o in alternativa estendere la classe *ErroreApplicativoBuilder* del package basic.

- ```
  public IEsitoBuilder createEsitoBuilder()
  ```

  Consente di ottenere un'istanza per valutare l'esito di una transazione di cooperazione tramite l'analisi del messaggio in transito. Si deve fornire un'implementazione dell'interfaccia *IEsitoBuilder* o in alternativa estendere la classe *EsitoBuilder* del package basic.

Vediamo adesso in dettaglio come sono strutturate le istanze restituite dai metodi sopra elencati.

3.1.1 org.openspcoop2.protocol.sdk.builder.IBustaBuilder

Contiene i metodi per gestire l'imbustamento e lo sbustamento dei messaggi in transito sulla Porta di Dominio. I metodi rilevanti contenuti sono:

- ```
  SOAPElement imbustamento(org.openspcoop2.message.OpenSPCoop2Message msg,
                                    Busta busta,
                                    boolean isRichiesta,
                                    ProprietaManifestAttachments  ↩
                                        proprietaManifestAttachments)
  ```

  elabora il messaggio ricevuto in input al fine di aggiungere l'intestazione di protocollo. I parametri:

  - *msg* - Messaggio in cui inserire le informazioni di cooperazione.
  - *busta* - I metadati di cooperazione, che sono:
    - azione
    - azioneRichiedenteBustaDiServizio
    - collaborazione
    - confermaRicezione
    - destinatario
    - id
    - identificativoPortaDestinatario
    - identificativoPortaMittente
    - indirizzoDestinatario
    - indirizzoMittente
    - inoltroValue
    - listaEccezioni
    - listaRiscontri
    - listaTrasmissioni
    - mittente
    - oraRegistrazione
    - profiloDiCollaborazioneValue
    - properties
    - riferimentoMessaggio
    - riferimentoMsgBustaRichiedenteServizio
    - scadenza
    - sequenza
    - servizio
  ```

    - servizioCorrelato
    - servizioRichiedenteBustaDiServizio
    - tipoDestinatario
    - tipoMittente
    - tipoOraRegistrazioneValue
    - tipoServizio
    - tipoServizioCorrelato
    - tipoServizioRichiedenteBustaDiServizio
    - versioneServizio
  - *isRichiesta* - Indicazione se il messaggio da modificare è di richiesta o di risposta
  - *proprietaManifestAttachments* - Proprietà necessarie per la generazione del manifest degli attachments, che sono:
    - *GestioneManifest* - booleano che indica se la PdD deve gestire il manifest degli attachments.
    - *ScartaBody* - booleano che indica se, in fase di gestione del manifest, la PdD deve scartare il body del messaggio.
    - *readQualifiedAttribute* - booleano che indica se, in fase di gestione del manifest, la PdD deve leggere da esso solo elementi aventi un namespace associato.

- ```
  SOAPElement addTrasmissione(org.openspcoop2.message.OpenSPCoop2Message message,
                                  Trasmissione trasmissione)
  ```

  Modifica il messaggio di cooperazione (quindi già imbustato) aggiungendo le informazioni di trasmissione, che sono:

  - destinazione
  - identificativoPortaDestinazione
  - identificativoPortaOrigine
  - indirizzoDestinazione
  - indirizzoOrigine
  - oraRegistrazione
  - origine
  - tempoValue
  - tipoDestinazione
  - tipoOrigine

- ```
  SOAPHeaderElement sbustamento(org.openspcoop2.message.OpenSPCoop2Message msg,
                                  ProprietaManifestAttachments  ↩
                                      proprietaManifestAttachments)
  ```

  Rimuove le informazioni di cooperazione dal messaggio ricevuto in input. Il messaggio così ottenuto sarà quello consegnato al servizio applicativo di destinazione.

3.1.2 org.openspcoop2.protocol.sdk.builder.IErroreApplicativoBuilder

Gestisce la generazione dei messaggi in caso di errore applicativo. Se durante il processamento dei messaggi la Porta di Dominio riscontra un errore, questo viene inserito nel messaggio di risposta applicativa che viene inoltrata al servizio applicativo fruitore. I metodi rilevanti contenuti sono:

- ```
  void insertInSOAPFault(EccezioneProtocolloBuilderParameters parameters,
                              org.openspcoop2.message.OpenSPCoop2Message msg)
  ```

  Aggiunge ad un messaggio contenente un SOAPFault, destinato al servizio applicativo, i dettagli dell'errore prelevati dall'oggetto parameters, che in questo caso descrive un errore di protocollo.
  ```

```
void insertInSOAPFault(EccezioneIntegrazioneBuilderParameters parameters,
                               org.openspcoop2.message.OpenSPCoop2Message msg)
```

Aggiunge ad un messaggio contenente un SOAPFault, destinato al servizio applicativo, i dettagli dell'errore prelevati dall'oggetto parameter, che in questo caso descrive un errore di integrazione.

- ```
  void insertRoutingErrorInSOAPFault(org.openspcoop2.core.id.IDSoggetto identitaRouter,
                               java.lang.String idFunzione,
                               java.lang.String msgErrore,
                               org.openspcoop2.message.OpenSPCoop2Message msg)
  ```

  Aggiunge ad un messaggio contenente un SOAPFault le informazioni di un errore di routing. I parametri sono:

  – *identitaRouter* - nome del soggetto che identifica la PdD Router che ha sollevato l'errore.
  – *idFunzione* - nome del modulo funzionale in cui si è verificato l'errore.
  – *msgErrore* - testo del messaggio di errore da inserire nel fault.
  – *msg* - messaggio di fault che deve essere modificato.

- ```
  AbstractEccezioneBuilderParameter readErroreApplicativo(byte[] xml,
                                       java.lang.String  ↩
                                          prefixCodiceErroreApplicativoIntegrazione)
  ```

  ```
  AbstractEccezioneBuilderParameter readErroreApplicativo(org.w3c.dom.Node xml,
                                       java.lang.String  ↩
                                          prefixCodiceErroreApplicativoIntegrazione)
  ```

  ```
  AbstractEccezioneBuilderParameter readErroreApplicativo(java.lang.String xml,
                                       java.lang.String  ↩
                                          prefixCodiceErroreApplicativoIntegrazione)
  ```

  Si occupano di interpretare l'errore applicativo e di mapparne il codice eccezione e la descrizione nell'oggetto da ritornare. L'istanza restituita, del tipo *AbstractEccezioneBuilderParameter*, contiene le seguenti informazioni:

  – *Dettaglio Eccezione PdD*
  – *Dominio Porta*
  – *Identificativo Funzione*
  – *Mittente*
  – *Ora Registrazione*
  – *Proprietà Errore Applicativo*
  – *Servizio*
  – *Servizio Applicativo*
  – *Tipo Porta*
  – *Versione SOAP*

3.1.3 org.openspcoop2.protocol.sdk.builder.IEsitoBuilder

Fornisce i metodi per ottenere l'esito di una transazione di cooperazione applicativa. Un'implementazione di questa interfaccia consente ad esempio di specificare con quale logica la Porta di Dominio è in grado di individuare gli errori applicativi tramite l'analisi del contenuto del messaggio in transito. I metodi rilevanti contenuti sono:

- ```
  Esito getEsito(org.openspcoop2.message.OpenSPCoop2Message message)
  ```
  ```

```
Esito getEsito(org.openspcoop2.message.OpenSPCoop2Message message,
                                ProprietaErroreApplicativo erroreApplicativo)
```

Tramite l'analisi del messaggio ricevuto in input forniscono l'esito della transazione di cooperazione applicativa. L'esito restituito è un enumeration che contiene i seguenti valori:

- ERRORE_APPLICATIVO
- ERRORE_GENERICO
- ERRORE_PROCESSAMENTO_PDD_4XX
- ERRORE_PROCESSAMENTO_PDD_5XX
- ERRORE_PROTOCOLLO
- OK

3.2 Validazione dei payload

I metodi di quest'area consentono di istanziare i gestori per effettuare la validazione delle informazioni di protocollo a corredo dei messaggi.

La Factory prevede i seguenti metodi:

- ```
  public IValidatoreErrori createValidatoreErrori()
  ```

  L'implementazione fornita dell'interfaccia *IValidatoreErrori* restituita consente alla Porta di Dominio di effettuare la validazione dei dati di protocollo relativi ai messaggi di errore in transito.

- ```
  public IValidazioneSintattica createValidazioneSintattica()
  ```

  L'implementazione fornita dell'interfaccia *IValidazioneSintattica* restituita consente di specificare la logica con cui la Porta di Dominio è in grado di effettuare la validazione sintattica dei dati di protocollo contenuti nei messaggi in transito.

- ```
  public IValidazioneSemantica createValidazioneSemantica()
  ```

  L'implementazione fornita dell'interfaccia *IValidazioneSemantica* restituita consente di specificare la logica con cui la Porta di Dominio è in grado di effettuare la validazione semantica dei dati di protocollo contenuti nei messaggi in transito.

- ```
  public IValidazioneConSchema createValidazioneConSchema()
  ```

  L'implementazione fornita dell'interfaccia *IValidazioneConSchema* consente di gestire la validazione sintattica tramite XSD.

Vediamo adesso in dettaglio come sono strutturate le istanze restituite dai metodi sopra elencati.

3.2.1 org.openspcoop2.protocol.sdk.validator.IValidatoreErrori

Fornisce metodi booleani per distinguere i casi di errore applicativo da quelli di protocollo.

Il comportamento dei metodi di validazione può essere modificato tramite l'elemento *ProprietaValidazioneErrori* che consente di fare switch tra due diversi livelli di severità nell'individuazione degli errori. I metodi rilevanti contenuti sono:

- ```
  boolean isBustaErrore(Busta busta,
                              org.openspcoop2.message.OpenSPCoop2Message msg,
                              ProprietaValidazioneErrori proprietaValidazioneErrori)
  ```

  Restituisce *true* se e solo se dall'analisi del messaggio ricevuto in input scaturisce che si è verificato un errore a livello del protocollo di cooperazione (errore di intestazione o di processamento).
  ```

- ```
  boolean isBustaErroreProcessamento(Busta busta,
                                org.openspcoop2.message.OpenSPCoop2Message msg,
                                ProprietaValidazioneErrori proprietaValidazioneErrori)
  ```

  Restituisce *true* se e solo se dall'analisi del messaggio ricevuto in input scaturisce che si è verificato un errore di processamento (ad esempio l'impossibilità di comunicare con un servizio applicativo).

- ```
  boolean isBustaErroreIntestazione(Busta busta,
                                org.openspcoop2.message.OpenSPCoop2Message msg,
                                ProprietaValidazioneErrori proprietaValidazioneErrori)
  ```

  Restituisce *true* se e solo se dall'analisi del messaggio ricevuto in input scaturisce che si è verificato un errore di intestazione (ad esempio il riscontro di un mittente inesistente).

ProprietaValidazioneErrori è un oggetto finalizzato alla definizione dei seguenti parametri:

- *Versione Protocollo*
- *Ignora Eccezioni Non Gravi*

3.2.2 org.openspcoop2.protocol.sdk.validator.IValidazioneSintattica

Fornisce i metodi per consentire alla Porta di Dominio di verificare che i dati di protocollo contenuti nei messaggi in transito siano corretti sintatticamente.

Per il ritorno dei risultati di validazione sintattica viene usata la classe *ValidazioneSintatticaResult* che è composta logicamente dai seguenti elementi:

- *erroriValidazione* - Vettore con gli errori di validazione riscontrati
- *erroriProcessamento* - Vettore con gli errori di processamento riscontrati durante la validazione
- *erroriTrovatiSullaListaEccezioni* - Vettore di eccezioni riscontrate tra le informazioni di cooperazione
- *busta* - Struttura dati con i dati di protocollo estratti dal messaggio
- *msgErrore* - Messaggio di errore da inserire nella risposta
- *codiceErrore* - Codice dell'errore da inserire nella risposta
- *bustaErrore* - Generata solo quando la busta arrivata non contiene gli elementi principali
- *isValido* - Test booleano che indica se il messaggio può considerarsi valido.

Il comportamento dei metodi di validazione può essere modificato tramite l'elemento *ProprietaValidazioneErrori* che consente di fare switch tra due diversi livelli di severità nell'individuazione degli errori.

I metodi rilevanti contenuti sono:

- ```
  ValidazioneSintatticaResult validaRichiesta(org.openspcoop2.message.OpenSPCoop2Message ↩
      msg,
                                  ProprietaValidazioneErrori proprietaValidazioneErrori)
  ```

  Esegue la validazione sintattica dei dati di protocollo presenti nel messaggio ricevuto in input. La logica di validazione è quella applicabile ai messaggi di richiesta.

- ```
  ValidazioneSintatticaResult validaRisposta(org.openspcoop2.message.OpenSPCoop2Message msg,
                                  Busta bustaRichiesta, ProprietaValidazioneErrori ↩
                                      proprietaValidazioneErrori)
  ```

Esegue la validazione sintattica dei dati di protocollo presenti nel messaggio ricevuto in input. La logica di validazione è quella applicabile ai messaggi di risposta.

- ```
  ValidazioneSintatticaResult validazioneFault(javax.xml.soap.SOAPBody body)
  ```

  Esegue la validazione sintattica del Fault presente in un messaggio contenente un errore.

- ```
  ValidazioneSintatticaResult validazioneManifestAttachments(
                                  org.openspcoop2.message.OpenSPCoop2Message msg,
                                  ProprietaManifestAttachments proprietaManifestAttachments)
  ```

  Esegue la validazione sintattica del manifest degli attachments per i casi in cui è previsto che questi venga gestito dalla Porta di Dominio.

3.2.3 org.openspcoop2.protocol.sdk.validator.IValidazioneSemantica

Fornisce i metodi per consentire alla Porta di Dominio di verificare che i dati di protocollo contenuti nei messaggi in transito siano corretti semanticamente e quindi coerenti con quanto previsto nel Registro dei Servizi.

Per il ritorno dei risultati di validazione semantica viene usata la classe *ValidazioneSemanticaResult* che è composta logicamente dai seguenti elementi:

- *erroriValidazione* - Vettore con gli errori di validazione riscontrati
- *erroriProcessamento* - Vettore con gli errori di processamento incorsi durante la validazione
- *servizioCorrelato* - Eventuale servizio correlato a quello richiesto nella busta
- *tipoServizioCorrelato* - Eventuale tipo del servizio correlato a quello richiesto nella busta
- *infoServizio* - Informazioni sul servizio

Il comportamento dei metodi di validazione può essere modificato tramite l'elemento *ProprietaValidazioneErrori* che consente di fare switch tra due diversi livelli di severità nell'individuazione degli errori.

I metodi rilevanti contenuti sono:

- ```
  boolean validazioneID(java.lang.String id,
                                  org.openspcoop2.core.id.IDSoggetto dominio,
                                  ProprietaValidazione proprietaValidazione)
  ```

  Verifica che l'identificativo del messaggio, ricevuto in input, rispetti le specifiche di protocollo.

  Il parametro *dominio* deve contenere l'identificativo parte del mittente nei casi in cui debba essere validato un identificativo del messaggio o una collaborazione di una richiesta (oneWay, richiestaSincrona, richiesta/ricevutaRisposta AsincronaSimmetrica, richiesta/richiestaStato AsincronaAsimmetrica), poiché sono i casi in cui è proprio il mittente che ha creato l'identificatore. Deve invece contenere l'identificativo parte del destinatario, quando deve essere validato un *RiferimentoMessaggio*, od una collaborazione di una risposta (Sincrona, ricevutaRichiesta/Risposta AsincronaSimmetrica, ricevutaRichiesta/ricevutaRichiestaStato AsincronaAsimmetrica), poiché sono i casi in cui la creazione dell'identificatore è stata fatta dal destinatario al momento della creazione della richiesta.

  Restituisce *true* se l'identificatore è valido.

- ```
  ValidazioneSemanticaResult valida(Busta busta,
                                  IState state,
                                  ProprietaValidazione proprietaValidazione,
                                  RuoloBusta tipoBusta)
  ```

  Metodo che verifica la validità semantica dei dati di protocollo presenti nel messaggio, controllandone la compatibilità con la configurazione del Registro dei Servizi. I parametri sono:

- *busta* - Busta con i dati di cooperazione da validare.
- *state* - Rappresentazione dello stato della busta.
- *proprietaValidazione* - Contiene le informazioni sul tipo di validazione da effettuare. Il tipo *ProprietaValidazione* consente di specificare le proprietà della validazione ed in particolare:
  * *ValidazioneConSchema* - Flag per indicare se deve essere effettuata una validazione secondo schema XSD
  * *ValidazioneIDCompleta* - Flag per indicare che l'ID deve essere validato anche semanticamente oltreché sintatticamente
  * *ValidazioneManifestAttachments* - Flag per indicare se deve essere validato o meno il manifesto degli attachments
  * *ValidazioneMessaggioRispostaConnectionReply* - Flag per indicare se deve essere validato o meno il messaggio ottenuto sulla reply della connessione
  * *ValidazioneProfiloCollaborazione* - Flag che indica se deve essere validato il profilo consultando il registro servizi
- *tipoBusta* - Ruolo della busta da validare. Possibili valori: *BUSTA_DI_SERVIZIO, RICEVUTA_RICHIESTA, RICEVUTA_RISPOSTA, RICHIESTA, RISPOSTA*

3.2.4 org.openspcoop2.protocol.sdk.validator.IValidazioneConSchema

Fornisce la possibilità di effettuare una validazione basata su schema XSD. I metodi rilevanti contenuti sono:

- ```
  void valida(javax.xml.soap.SOAPEnvelope soapEnvelope,
                                  javax.xml.soap.SOAPHeaderElement header,
                                  javax.xml.soap.SOAPBody soapBody,
                                  boolean isErroreProcessamento,
                                  boolean isErroreIntestazione,
                                  boolean isMessaggioConAttachments,
                                  boolean validazioneManifestAttachments)
  ```

  Metodo che effettua la validazione dei soggetti di una busta, controllando la loro registrazione nel registro dei servizi. Gli eventuali errori riscontrati vengono memorizzati e quindi successivamente recuperati tramite opportuni metodi get.

3.3 Emissione messaggi diagnostici

I metodi di quest'area contengono tutto ciò che consente alla Porta di Dominio di produrre i messaggi diagnostici.

La Factory prevede i seguenti metodi:

- ```
  public IDriverMsgDiagnostici createDriverMSGDiagnostici()
  ```

  L'implementazione fornita dell'interfaccia IDriverMsgDiagnostici supporta le funzionalità di ricerca e consultazione del repository dei messaggi diagnostici.

- ```
  public IMsgDiagnosticoOpenSPCoopAppender createMsgDiagnosticoOpenSPCoopAppender()
  ```

  L'implementazione fornita dell'interfaccia *IMsgDiagnosticoOpenSPCoopAppender* supporta le funzionalità per la produzione ed inserimento nel repository di nuovi messaggi diagnostici.

- ```
  public IXMLDiagnosticoBuilder createXMLDiagnosticoBuilder()
  ```

  L'implementazione fornita dell'interfaccia *IXMLDiagnosticoBuilder* supporta i meccanismi di serializzazione XML dei messaggi diagnostici. Questa funzionalità trova applicazione ad esempio nei casi in cui sia necessaria l'esportazione dei messaggi diagnostici dal repository della Porta di Dominio.

Vediamo adesso in dettaglio come sono strutturate le istanze restituite dai metodi sopra elencati.

FiltroRicercaDiagnostici	
busta (azione, erogatore, fruitore, servizio, tipo servizio, versione servizio)	Elementi dell'intestazione di protocollo
correlazioneApplicativa	Criterio di filtro sul dato di correlazione applicativa estratto dalla richiesta
correlazioneApplicativaOrMatch	Criterio di filtro sul dato di correlazione applicativa estratto dalla richiesta o (OR) dalla risposta
correlazioneApplicativaRisposta	Criterio di filtro sul dato di correlazione applicativa estratto dalla risposta
dataFine	Estremo iniziale dell'intervallo di appartenenza del timestamp del diagnostico
dataInizio	Estremo finale dell'intervallo di appartenenza del timestamp del diagnostico
flag delegata	Booleano. Se true, si filtrano solo i diagnostici generati dalla PdD con ruolo di mittente; con ruolo destinataria altrimenti.
dominio	Nome del soggetto cui appartiene la PdD che ha generato il diagnostico
filtroSoggetti	Lista di soggetti coinvolti nella comunicazione riferita dal diagnostico
idBustaRichiesta	Identificativo della busta di richiesta
idBustaRisposta	Identificativo della busta di risposta
idFunzione	Modulo funzionale che ha generato il diagnostico
nomePorta	Nome della PdD che ha generato il diagnostico
properties	Filtro basato su collezione di proprietà personalizzate
flag ricercaSoloMessaggiCorrelatiInformazioniProtocollo	Se true restringe la ricerca dei diagnostici che sono correlati a dati di intestazione del protocollo. Consente di escludere diagnostici riferiti ad eventi di sistema o per i quali non si sia potuto estrarre i dati di intestazione.
servizioApplicativo	Servizio applicativo che ha generato il messaggio applicativo
severità	Livello minimo di severità del diagnostico. Consente di escludere i diagnostici con severità inferiore alla soglia impostata.

Tabella 1: FiltroRicercaDiagnostici

3.3.1 org.openspcoop2.protocol.sdk.diagnostica.IDriverMsgDiagnostici

L'implementazione di questa interfaccia consente di poter consultare il repository dei messaggi diagnostici.

Per impostare i filtri di ricerca si utilizza la classe *FiltroRicercaDiagnostici* che contiene i seguenti elementi logici:

La sottoclasse di quest'ultima, *FiltroRicercaDiagnosticiConPaginazione*, contiene anche gli elementi per impostare i criteri di paginazione. Per la restituzione del messaggio diagnostico viene utilizzata la classe *MsgDiagnostico* che è così strutturata logicamente:

MsgDiagnostico	
codice	codice del diagnostico
gdo	timestamp
id	identificativo
idBusta	identificativo della busta di origine
idBustaRisposta	identificativo della busta di risposta
idFunzione	modulo funzionale che ha generato il messaggio
idSoggetto	identificativo della porta di dominio che ha generato il messaggio
messaggio	testo del diagnostico
properties	hashmap con proprietà generiche personalizzabili
severità	livello di severità

Tabella 2: MsgDiagnostico

I metodi rilevanti sono:

- ```
  int countMessaggiDiagnostici(FiltroRicercaDiagnostici filtro)
  ```

  Restituisce il numero di messaggi diagnostici che soddisfano il filtro di ricerca ricevuto in input.

- ```
  java.util.List<MsgDiagnostico> getMessaggiDiagnostici( ↩
      FiltroRicercaDiagnosticiConPaginazione filtro)
  ```

  Restituisce i messaggi diagnostici che soddisfano il filtro di ricerca ricevuto in input.

- ```
  int deleteMessaggiDiagnostici(FiltroRicercaDiagnostici filter)
  ```

  Elimina dal repository i messaggi diagnostici che soddisfano il filtro di ricerca ricevuto in input.

Accanto ai tre metodi sopra descritti ve ne sono tre analoghi che riguardano le informazioni di correlazione, vale a dire i dati di protocollo associati a ciascun messaggio diagnostico:

- ```
  int countInfoCorrelazioniMessaggiDiagnostici(FiltroRicercaDiagnostici filtro)
  ```

- ```
  int deleteInfoCorrelazioniMessaggiDiagnostici(FiltroRicercaDiagnostici filter)
  ```

- ```
  java.util.List<MsgDiagnosticoCorrelazione> getInfoCorrelazioniMessaggiDiagnostici( ↩
      FiltroRicercaDiagnosticiConPaginazione filtro)
  ```

3.3.2 org.openspcoop2.protocol.sdk.diagnostica.IMsgDiagnosticoOpenSPCoopAppender

L'implementazione di questa interfaccia consente di definire la logica di generazione dei messaggi diagnostici. I metodi rilevanti sono:

- `void log(MsgDiagnostico msgDiagnostico)`

  Registra un messaggio diagnostico emesso dalla porta di dominio.

- `void logCorrelazione(MsgDiagnosticoCorrelazione msgDiagCorrelazione)`

  Registra i dati di correlazione (informazioni di protocollo) per un dato messaggio diagnostico.

- `void logCorrelazioneApplicativaRisposta(MsgDiagnosticoCorrelazioneApplicativa msgDiagCorrelazioneApplicativa)`

  Registrazione dell'identificativo di correlazione applicativa della risposta.

- `void logCorrelazioneServizioApplicativo(MsgDiagnosticoCorrelazioneServizioApplicativo msgDiagCorrelazioneSA)`

  Creazione di una correlazione applicativa tra messaggi diagnostici e servizi applicativi.

3.3.3 org.openspcoop2.protocol.sdk.diagnostica.IXMLDiagnosticoBuilder

L'implementazione di questa interfaccia consente di gestire la logica di serializzazione xml dei messaggi diagnostici. Questa funzionalità consente tipicamente di effettuare esportazioni di messaggi diagnostici dal repository della Porta di Dominio.

I metodi per la serializzazione si differenziano per il formato dell'elemento restituito e sono:

- `byte[] toByteArray(MsgDiagnostico msgDiag)`
- `org.w3c.dom.Element toElement(MsgDiagnostico msgDiag)`
- `java.lang.String toString(MsgDiagnostico msgDiag)`

3.4 Tracciatura

I metodi di quest'area contengono tutto ciò che consente alla Porta di Dominio di produrre le tracce.

La Factory prevede i seguenti metodi:

- `public IDriverTracciamento createDriverTracciamento()`

  L'implementazione fornita dell'interfaccia *IDriverTracciamento* supporta le funzionalità di ricerca e consultazione del repository delle tracce.

- `public ITracciamentoOpenSPCoopAppender createTracciamentoOpenSPCoopAppender()`

  L'implementazione fornita dell'interfaccia *ITracciamentoOpenSPCoopAppender* supporta le funzionalità per la produzione ed inserimento nel repository di nuove tracce.

- `public IXMLTracciaBuilder createXMLTracciaBuilder()`

  L'implementazione fornita dell'interfaccia *IXMLTracciaBuilder* supporta i meccanismi di serializzazione XML delle tracce. Questa funzionalità trova applicazione ad esempio nei casi in cui sia necessaria l'esportazione dal repository delle tracce.

Vediamo adesso in dettaglio come sono strutturate le istanze restituite dai metodi sopra elencati.

3.4.1 org.openspcoop2.protocol.sdk.tracciamento.IDriverTracciamento

L'implementazione di questa interfaccia consente di poter consultare il repository delle tracce.

Per impostare i filtri di ricerca si utilizza la classe *FiltroRicercaTracce* che contiene i seguenti elementi logici: La sottoclasse di

FiltroRicercaTracce	
dominio	Identificativo del soggetto gestito dalla PdD che ha emesso la traccia
filtroSoggetti	Lista di soggetti coinvolti nelle tracce da restituire
idBusta	Identificativo della busta correlata alla traccia ricercata
idCorrelazioneApplicativa	Identificativo di correlazione applicativa estratto dal messaggio di richiesta
idCorrelazioneApplicativaOrMatch	Filtra le tracce che hanno una corrispondenza con la correlazione applicativa della richiesta o (OR) della risposta
idCorrelazioneApplicativaRisposta	Identificativo di correlazione applicativa estratto dal messaggio di risposta
informazioniProtocollo	Filtro sui dati di intestazione dei messaggi associati alle tracce
maxDate	Estremo iniziale dell'intervallo temporale di ricerca delle tracce basato sul timestamp di emissione
minDate	Estremo finale dell'intervallo temporale di ricerca delle tracce basato sul timestamp di emissione
properties	Filtro basato su collezione di proprietà personalizzate
ricercaSoloBusteErrore	Include solo tracce riferite a comunicazioni che hanno prodotto buste errore (errori di protocollo)
riferimentoMessaggio	Identificativo del messaggio correlato a quello riferito dalla traccia
tipoTraccia	Consente di filtrare le tracce in base alla tipologia del messaggio (Richiesta o Risposta) cui si riferiscono

Tabella 3: FiltroRicercaTracce

quest'ultima, *FiltroRicercaTracceConPaginazione*, contiene anche gli elementi per impostare i criteri di paginazione dei risultati.

Per la restituzione della traccia viene utilizzata la classe *Traccia* che è così strutturata logicamente: I metodi rilevanti sono:

Traccia	
busta (azione, erogatore, fruitore, servizio, tipo servizio, versione servizio)	dati di protocollo
correlazioneApplicativa	informazioni di correlazione applicativa sulla richiesta
correlazioneApplicativaRisposta	Informazioni di correlazione applicativa sulla risposta
gdo	timestamp della traccia
idSoggetto	identificativo della porta di dominio che ha emesso la traccia
listaAllegati	lista allegati presenti nella transazione
location	indirizzo di provenienza o destinazione
properties	collezione di properties personalizzabili
tipoMessaggio	tipo della traccia: richiesta o risposta

Tabella 4: Traccia

- `int countTracce(FiltroRicercaTracce filtro)`

  conta le tracce che soddisfano il filtro di ricerca.

- ```
  int deleteTracce(FiltroRicercaTracce filter)
  ```

  elimina le tracce che soddisfano il filtro di ricerca.

- ```
  java.util.List<Traccia> getTracce(FiltroRicercaTracceConPaginazione filtro)
  ```

  restituisce le tracce che soddisfano il filtro di ricerca.

- ```
  Traccia getTraccia(java.lang.String idBusta, org.openspcoop2.core.id.IDSoggetto ↩
      codicePorta)
  ```

  restituisce la traccia che soddisfa i parametri in input.

- ```
  Traccia getTraccia(java.lang.String idBusta, org.openspcoop2.core.id.IDSoggetto ↩
      codicePorta,
                  boolean ricercaIdBustaComeRiferimentoMessaggio)
  ```

  restituisce la traccia che soddisfa i parametri in input.

- ```
  Traccia getTraccia(TipoTraccia tipoTraccia,
                  java.util.Hashtable<java.lang.String, java.lang.String> propertiesRicerca)
  ```

  restituisce la traccia che soddisfa il criterio costituito dai parametri in input.

3.4.2 org.openspcoop2.protocol.sdk.tracciamento.ITracciamentoOpenSPCoopAppender

L'implementazione di questa interfaccia consente di definire la logica di generazione delle tracce.

I metodi rilevanti sono:

- ```
  void log(java.sql.Connection conOpenSPCoopPdD, Traccia traccia)
  ```

  registra una traccia emessa dalla Porta di Dominio.

3.4.3 org.openspcoop2.protocol.sdk.tracciamento.IXMLTracciaBuilder

L'implementazione di questa interfaccia consente di gestire la logica di serializzazione xml delle tracce. Questa funzionalità consente tipicamente di effettuare esportazioni di tracce dal repository della Porta di Dominio.

I metodi per la serializzazione si differenziano per il formato dell'elemento restituito e sono:

- ```
  byte[] toByteArray(Traccia traccia)
  ```

- ```
  javax.xml.soap.SOAPElement toElement(Traccia traccia)
  ```

- ```
  java.lang.String toString(Traccia traccia)
  ```
  ```

3.5 Gestione della configurazione

I metodi di quest'area hanno lo scopo di consentire alla Porta di Dominio di accedere ai dati di configurazione relativi al protocollo.

La Factory prevede i seguenti metodi:

- ```
  public IProtocolManager createProtocolManager()
  ```

  L'implementazione fornita dell'interfaccia *IProtocolManager* consente alla Porta di Dominio di recuperare numerosi dettagli e parametri necessari alla gestione del protocollo.

- ```
  public IProtocolVersionManager createProtocolVersionManager(String version)
  ```

  L'interfaccia *IProtocolVersionManager* estende *IProtocolManager* aggiungendo ulteriori metodi per recuperare dettagli di configurazione nei casi in cui sussistono diverse versioni di protocollo.

- ```
  public ITraduttore createTraduttore()
  ```

  L'implementazione dell'interfaccia *ITraduttore* consente di fornire un mapping tra entità che rappresentano generiche funzionalità della Porta di Dominio (ad es.: codici di errore, nomi dei profili di collaborazione, ecc.) e il naming adottato dal nuovo protocollo.

- ```
  public IProtocolConfiguration createProtocolConfiguration()
  ```

  L'implementazione fornita dell'interfaccia *IProtocolConfiguration* contiene la logica per il recupero dei dati di configurazione di base: Lista Tipi Soggetti, Lista Tipi Servizi e Versioni Protocollo. L'implementazione di default prevede che tali dati vengano letti dal file *openspcoop2-manifest.xml* presente nel file jar del plugin.

Vediamo adesso in dettaglio come sono strutturate le istanze restituite dai metodi sopra elencati.

3.5.1 org.openspcoop2.protocol.sdk.config.IProtocolManager

L'implementazione di questa interfaccia consente alla Porta di Dominio di recuperare numerosi dettagli di configurazione necessari al funzionamento del protocollo di cooperazione. Si rimanda al *javadoc* per il dettaglio di tali metodi.

3.5.2 org.openspcoop2.protocol.sdk.config.IProtocolVersionManager

Questa interfaccia è una specializzazione della precedente con la gestione in aggiunta della versione del protocollo. In tal modo una Porta di Dominio che supporti diverse versioni del medesimo protocollo può recuperare i parametri di configurazione distinti per versione.

3.5.3 org.openspcoop2.protocol.sdk.config.ITraduttore

L'implementazione di questa interfaccia consente di creare un mapping personalizzato delle keyword del protocollo rispetto al naming standard adottato da OpenSPCoop2. Per la descrizione di tutti i metodi di traduzione presenti in questa interfaccia si rimanda al *javadoc*.

3.5.4 org.openspcoop2.protocol.sdk.config.IProtocolConfiguration

L'implementazione di questa interfaccia consente di specificare la logica per il recupero dei dati di configurazione di base: Lista Tipi Soggetti, Lista Tipi Servizi e Versioni Protocollo.

I metodi rilevanti sono:

- `java.util.List<java.lang.String> getTipiServizi()`

  Restituisce la lista dei tipi associabili ai servizi.

- `java.util.List<java.lang.String> getTipiSoggetti()`

  Restituisce la lista dei tipi associabili ai soggetti.

- `java.util.List<java.lang.String> getVersioni()`

  Restituisce la lista delle versioni del protocollo.

4 Confezionamento del plugin relativo al nuovo protocollo

I sorgenti che implementano il nuovo protocollo devono essere racchiusi in una libreria *JAR* che dovrà essere consegnata al gestore della Porta di Dominio per l'installazione.

Affinché il plugin venga correttamente riconosciuto da OpenSPCoop2, e quindi agganciato ai meccanismi di comunicazione supportati, devono essere prodotte alcune entità di configurazione tutte da includere nella libreria JAR:

- nuovo-protocollo.jar
  - META-INF/
  - <sources-root>/
  - openspcoop2-manifest.xml
  - <protocolName>-url-mapping.properties

4.1 Il file di configurazione openspcoop2-manifest.xml

Il file di configurazione principale per consentire ad OpenSPCoop2 di riconoscere il plugin è *openspcoop2-manifest.xml*.

La figura seguente mostra la struttura del file:

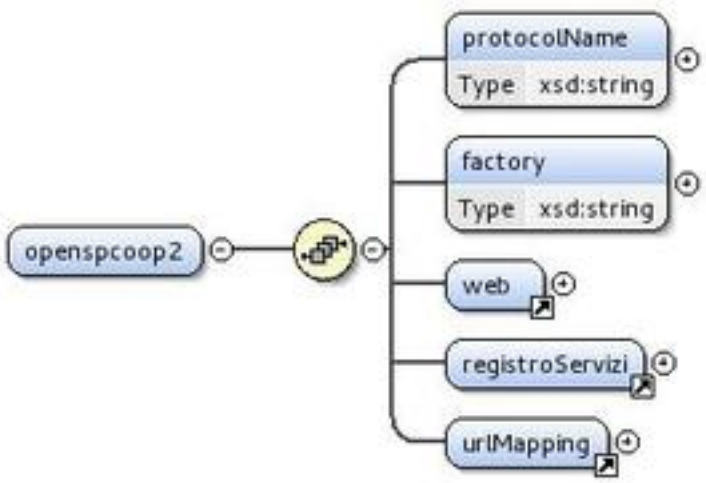

Figura 1: Struttura del file openspcoop2-manifest.xml

I primi due elementi racchiudono delle stringhe e sono:

- *protocolName* = contiene il nome del protocollo
- *factory* = riporta il *fully qualified classname* della factory, classe principale per istanziare tutti i gestori implementati dal protocollo

La loro struttura sarà quindi del tipo:

```
<protocolName>nome-protocollo</protocolName>
<factory>org.openspcoop2.protocol.nome-protocollo.NewProtocolFactory</factory>
```

L'elemento *web* consente di stabilire il contesto dell'application server con cui verrà effettuato il deploy dei servizi della Porta di Dominio dedicati al nuovo protocollo. L'elemento ha la struttura mostrata nella figura seguente:

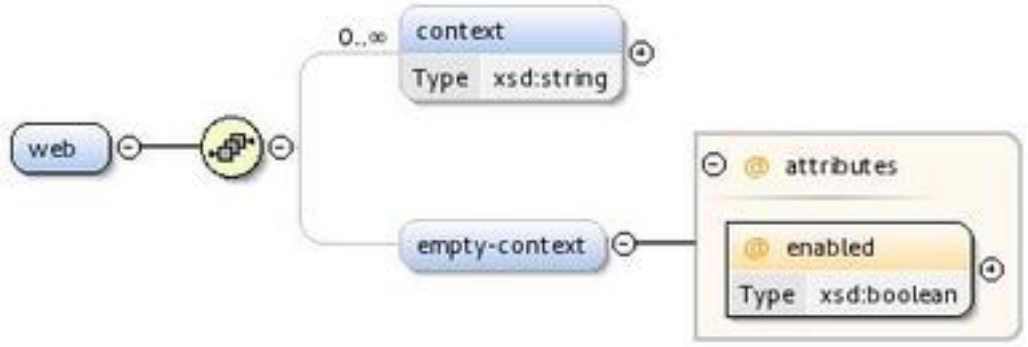

Figura 2: Struttura dell'elemento web del file openspcoop2-manifest.xml

I sotto-elementi da fornire sono i seguenti:

- *context* = nome del contesto di deploy. Tale nome dovrà risultare unico nell'ambito dell'application server che ospita OpenSPCoop2.
- *empty-context* = attributo enabled *(true|false)*. Se impostato a *true* i servizi saranno resi disponibili sul *root context* dell'application server.

Specificando un contesto saranno resi disponibili i seguenti servizi:

- http://<hostname>:<port>/openspcoop2/<context>/PD
- http://<hostname>:<port>/openspcoop2/<context>/PA
- http://<hostname>:<port>/openspcoop2/<context>/IntegrationManager

Abilitando il root context (<empty-context enabled=true/>):

- http://<hostname>:<port>/openspcoop2/PD
- http://<hostname>:<port>/openspcoop2/PA
- http://<hostname>:<port>/openspcoop2/IntegrationManager

Questo elemento avrà quindi una struttura del tipo:

```
<web>
  <context>protocol-context</context>
  <empty-context enabled="false"/>
</web>
```

L'elemento *registroServizi* fornisce gli elementi per l'identificazione delle entità del registro appartenenti al nuovo protocollo in modo che la Porta di Dominio possa distinguerle. L'elemento ha la struttura mostrata nella figura seguente:

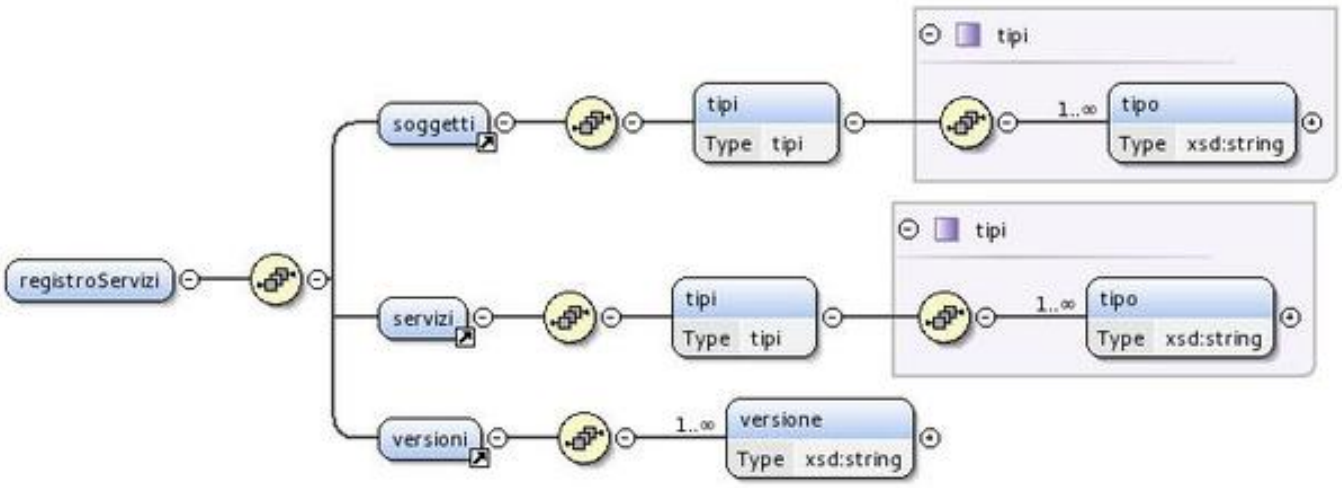

Figura 3: Struttura dell'elemento registroServizi del file openspcoop2-manifest.xml

Il sotto-elemento *soggetti* contiene la lista dei tipi che possono essere assegnati ai soggetti gestiti dalla Porta di Dominio che utilizzano il nuovo protocollo.

L'elemento ha la seguente struttura:

```
<soggetti>
  <tipi>
    <tipo>tipo1</tipo>
    <tipo>tipo2</tipo>
    ...
    <tipo>tipoN</tipo>
  </tipi>
</soggetti>
```

Il sotto-elemento *servizi* contiene la lista dei tipi che possono essere assegnati ai servizi gestiti dalla Porta di Dominio che utilizzano il nuovo protocollo.

L'elemento ha la seguente struttura:

```
<servizi>
<tipi>
<tipo>tipo1</tipo>
<tipo>tipo2</tipo>
...
<tipo>tipoN</tipo>
</tipi>
</servizi>
```

Il sotto-elemento *versioni* contiene la lista delle versioni del nuovo protocollo che saranno supportate dalla Porta di Dominio tramite il plugin. L'elemento ha la seguente struttura:

```
<versioni>
  <versione>versione1</versione>
  <versione>versione2</versione>
  ...
  <versione>versioneN</versione>
</versioni>
```

L'elemento *urlMapping* contiene il riferimento al file che contiene i mapping per risalire dalla url di invocazione della porta applicativa ai dati di indirizzamento necessari alla gestione del flusso della Porta di Dominio. La struttura dell'elemento è quella riportata nella seguente figura:

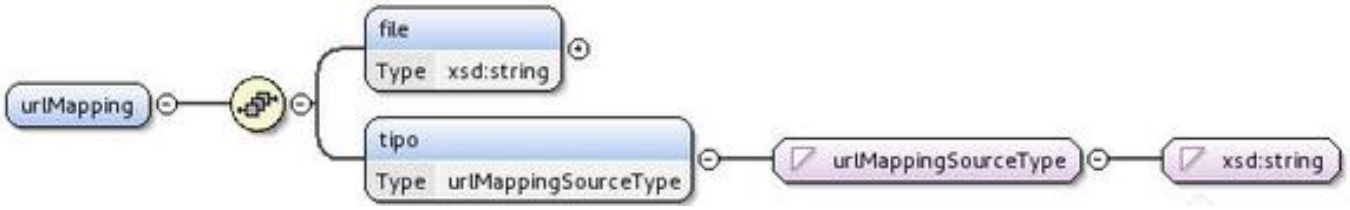

Figura 4: Struttura dell'elemento urlMapping del file openspcoop2-manifest.xml

Il sotto-elemento *file* contiene il path del file di mapping relativamente alla radice della libreria jar del protocollo.

Il sotto-elemento *tipo* contiene una stringa che rappresenta il formato del file di mapping. I valori ammessi per questo elemento sono *XML* e *PROPERTIES*. Attualmente l'unico formato supportato è *PROPERTIES*. Vediamo un possibile esempio del file openspcoop2-manifest.xml relativo al caso del protocollo SPCoop:

```
<?xml version="1.0" encoding="UTF-8"?>
<openspcoop2 xmlns="http://www.openspcoop2.org/protocol/manifest">

  <protocolName>spcoop</protocolName>

  <factory>org.openspcoop2.protocol.spcoop.SPCoopFactory</factory>

  <web>
    <context>spcoop</context>
    <empty-context enabled="true"/>
  </web>

  <registroServizi>

    <soggetti>
      <tipi>
        <tipo>SPC</tipo>
        <tipo>TEST</tipo>
        <tipo>AOO</tipo>
      </tipi>
    </soggetti>

    <servizi>
      <tipi>
        <tipo>SPC</tipo>
        <tipo>TEST</tipo>
        <tipo>URL</tipo>
        <tipo>WSDL</tipo>
        <tipo>LDAP</tipo>
        <tipo>UDDI</tipo>
        <tipo>ebXMLRegistry</tipo>
      </tipi>
    </servizi>

    <versioni>
      <versione>eGov1.1</versione>
      <versione>eGov1.1-lineeGuida1.1</versione>
    </versioni>

  </registroServizi>
```

```
    <urlMapping>
      <file>/spcoop-url-mapping.properties</file>
      <tipo>PROPERTIES</tipo>
    </urlMapping>

  </openspcoop2>
```

4.2 Il file di mapping delle url

Il file di mapping delle url descrive la logica con cui la Porta di Dominio può ricavare, per ciascun flusso in ingresso, i dati di indirizzamento (soggetti in gioco e porta applicativa) del contesto di cooperazione. Il mapping descritto riguarda le possibili url con cui viene invocato il servizio di porta applicativa della Porta di Dominio.

Il formato di questo file di configurazione può essere *XML* o *PROPERTIES*. Allo stato attuale la Porta di Dominio supporta solo il formato *PROPERTIES* ed è quindi questo formato che andiamo a descrivere.

Tutte le entry contenute nel file seguono il seguente formato:

```
<protocolName>.pa.<clusterName>.<keyword>=<value>
```

<protocolName> deve essere sostituito con il nome assegnato al nuovo protocollo e quindi il valore dell'elemento *protocolName* nel file *openspcoop2-manifest.xml*.

<clusterName> è una stringa a scelta dell'utente che funge da elemento di raggruppamento e consente quindi di creare nomi distinti per i vari flussi gestiti dalla Porta di Dominio.

<keyword> e **<value>** sono il nome ed il valore della chiave che si vuole definire. I nomi e i valori devono essere forniti in accordo alla logica che andiamo a descrivere.

url Il valore assegnato a questa chiave consente di identificare, analizzando la url di invocazione della porta applicativa, la regola da applicare tra quelle definite nel file di url-mapping. L'identificazione avviene confrontando la parte della url che segue */openspcoop2/<protocolName>/PA*.

Quindi nel caso venga invocata la url:

```
http://server-porta-di-dominio/openspcoop2/nuovo-protocollo/PA/invoke
```

Il valore da confrontare con il pattern definito dalla chiave url è *invoke*. È possibile utilizzare il carattere '*' come wildcard solo nei seguenti modi:

- * - Assegnando questo valore alla chiave url si indica che la regola di mapping vale per qualunque url pervenuta al servizio PA del nuovo protocollo.
- *pattern** - Assegnando questo valore alla chiave url si indica che la regola di mapping vale per tutte le url pervenute al servizio PA del nuovo protocollo che iniziano per *pattern*.

identificazione-pa Il valore di questa chiave stabilisce il metodo di identificazione della porta applicativa che la Porta di Dominio deve utilizzare per gestire il flusso. I valori ammessi sono:

- *static* - il nome della porta applicativa è contenuto nel medesimo file di url-mapping e corrisponde al valore della chiave *identificazione-pa.valore*
- *protocol* - il nome della porta applicativa sarà determinato dal protocollo tramite una logica ad hoc definita nell'implementazione.

identificazione-pa.valore Chiave presente solo quando *identificazione-pa* = *static*. Contiene il nome della porta applicativa.

identificazione-tipo-mittente Il valore di questa chiave stabilisce il metodo di identificazione del tipo del soggetto mittente che la Porta di Dominio deve utilizzare per gestire il flusso. I valori ammessi sono:

- *static* - il tipo del soggetto mittente è contenuto nel medesimo file di url-mapping e corrisponde al valore della chiave *identificazione-tipo-mittente.valore*.
- *protocol* - il tipo del soggetto mittente sarà determinato dal protocollo tramite una logica ad hoc definita nell'implementazione.

identificazione-tipo-mittente.valore Chiave presente solo quando *identificazione-tipo-mittente* = *static*. Contiene il tipo del soggetto mittente.

identificazione-nome-mittente Il valore di questa chiave stabilisce il metodo di identificazione del nome del soggetto mittente che la Porta di Dominio deve utilizzare per gestire il flusso. I valori ammessi sono:

- *static* - il nome del soggetto mittente è contenuto nel medesimo file di url-mapping e corrisponde al valore della chiave *identificazione-nome-mittente.valore*.
- *protocol* - il nome del soggetto mittente sarà determinato dal protocollo tramite una logica ad hoc definita nell'implementazione.

identificazione-nome-mittente.valore Chiave presente solo quando *identificazione-nome-mittente* = *static*. Contiene il nome del soggetto mittente.

identificazione-tipo-proprietario Il valore di questa chiave stabilisce il metodo di identificazione del tipo del soggetto destinatario che la Porta di Dominio deve utilizzare per gestire il flusso. I valori ammessi sono:

- *static* - il tipo del soggetto destinatario è contenuto nel medesimo file di url-mapping e corrisponde al valore della chiave *identificazione-tipo-destinatario.valore*
- *protocol* - il tipo del soggetto destinatario sarà determinato dal protocollo tramite una logica ad hoc definita nell'implementazione.

identificazione-tipo-proprietario.valore Chiave presente solo quando *identificazione-tipo-proprietario* = *static*. Contiene il tipo del soggetto destinatario.

identificazione-nome-proprietario Il valore di questa chiave stabilisce il metodo di identificazione del nome del soggetto destinatario che la Porta di Dominio deve utilizzare per gestire il flusso. I valori ammessi sono:

- *static* - il nome del soggetto destinatario è contenuto nel medesimo file di url-mapping e corrisponde al valore della chiave *identificazione-nome-destinatario.valore*.
- *protocol* - il nome del soggetto destinatario sarà determinato dal protocollo tramite una logica ad hoc definita nell'implementazione.

identificazione-nome-proprietario.valore Chiave presente solo quando *identificazione-nome-proprietario* = *static*. Contiene il nome del soggetto destinatario.

4.2.1 Esempio 1 - File spcoop-url-mapping.properties

Esempio di file di url-mapping valido per il protocollo *SPCoop*:

```
spcoop.pa.SPCOOP-MAPPING.url=*
spcoop.pa.SPCOOP-MAPPING.identificazione-pa=protocol
spcoop.pa.SPCOOP-MAPPING.identificazione-tipo-mittente=protocol
spcoop.pa.SPCOOP-MAPPING.identificazione-nome-mittente=protocol
spcoop.pa.SPCOOP-MAPPING.identificazione-tipo-proprietario=protocol
spcoop.pa.SPCOOP-MAPPING.identificazione-nome-proprietario=protocol
```

Questa configurazione prevede che per tutte le url in ingresso al servizio PA, i dati di indirizzamento vengano prelevati con la logica specificata nell'implementazione del protocollo. Infatti in questo caso i dati vengono ricavati interpretando l'header di protocollo presente nel messaggio ricevuto.

4.2.2 Esempio 2 - File trasparente-url-mapping.properties

Esempio di file di url-mapping per il caso del protocollo *Trasparente*:

```
trasparente.pa.profili.1.url=oneway
trasparente.pa.profili.1.identificazione-pa=static
trasparente.pa.profili.1.identificazione-pa.valore=PA_onewayTrasparente
trasparente.pa.profili.1.identificazione-tipo-mittente=static
trasparente.pa.profili.1.identificazione-tipo-mittente.valore=TRASP
trasparente.pa.profili.1.identificazione-nome-mittente=static
trasparente.pa.profili.1.identificazione-nome-mittente.valore=MinisteroFruitore
trasparente.pa.profili.1.identificazione-tipo-proprietario=static
trasparente.pa.profili.1.identificazione-tipo-proprietario.valore=TRASP
trasparente.pa.profili.1.identificazione-nome-proprietario=static
trasparente.pa.profili.1.identificazione-nome-proprietario.valore= ↩
    MinisteroErogatore

trasparente.pa.profili.2.url=onewayStateless
trasparente.pa.profili.2.identificazione-pa=static
trasparente.pa.profili.2.identificazione-pa.valore= ↩
    PA_onewayStatelessTrasparente
trasparente.pa.profili.2.identificazione-tipo-mittente=static
trasparente.pa.profili.2.identificazione-tipo-mittente.valore=TRASP
trasparente.pa.profili.2.identificazione-nome-mittente=static
trasparente.pa.profili.2.identificazione-nome-mittente.valore=MinisteroFruitore
trasparente.pa.profili.2.identificazione-tipo-proprietario=static
trasparente.pa.profili.2.identificazione-tipo-proprietario.valore=TRASP
trasparente.pa.profili.2.identificazione-nome-proprietario=static
trasparente.pa.profili.2.identificazione-nome-proprietario.valore= ↩
    MinisteroErogatore
```

Questa configurazione prevede due possibili url in ricezione per il protocollo Trasparente. I dati Porta Applicativa, Tipo/Nome Mittente e Tipo/Nome Proprietario sono determinati staticamente tramite i valori forniti nella configurazione stessa.