Stoppa Andrea 504313-IF

Università di Padova
Facoltà di Ingegneria

Relazione di Tirocinio: Progettazione e Sviluppo di un' Applicazione per l'Interoperabilità tra Pubbliche Amministrazioni con Architettura SOA

Tirocinante: Andrea Stoppa
Relatore: Ing. Massimo Rumor

Azienda Ospitante: AS2 S.r.l.
con sede a Rovigo (RO), Italy

Indice

1 - Obiettivi 2

2 - Situazione di partenza 3

2.1 - Contesto aziendale 4

2.2 - Pubblicazione e consultazione degli atti amministrativi prima di MyPortal 2.5 4

3 – MyPortal e il progetto SIRV-Interop 7

3.1 – Il progetto SIRV-Interop 7

3.2 – Il progetto MyPortal 11

4 – Tecnologie utilizzate 13

4.1 – Web Service ed XML 13

4.2 – Service Oriented Architecture (SOA) 17

4.3 – Porte Di Dominio e Web Service 18

4.4 – ColdFusion 20

4.5 – Eclipse 21

4.6 – Apache Tomcat 21

5 – Progettazione e sviluppo dei moduli applicativi 22

5.1 – Analisi delle strutture dati 24

5.1.1 – Analisi delle strutture dati usate dal sistema legacy del Comune di Rovigo 24

5.1.2 – Analisi delle strutture dati usate dai web service di MyPortal 2.5 27

5.2 – Sviluppo del modulo di export dati per raffinazioni successive 30

5.3 – Sviluppo del modulo di invio atti tramite WS 35

6 – Risultati 49

7 – Conclusioni 52

7.1 – Obiettivi conseguiti e punti a favore delle SOA 52

7.2 – Aspetti delicati e sconvenienza delle SOA 54

1 - Obiettivi

La presente relazione ha per oggetto un tirocinio ospitato presso l'azienda AS2 S.r.l. di Rovigo, atto a completare un percorso di studi di laurea triennale in Ingegneria Informatica presso l'Università di Padova. Il sottoscritto Andrea Stoppa ha effettuato tale tirocinio di oltre cinquecento ore nel periodo che va da Novembre 2010 a Giugno 2011 compresi, presso la sede operativa di AS2 S.r.l. all'interno dell'edificio della Provincia di Rovigo situato a Rovigo in Viale Della Pace 5.

Il progetto proposto come oggetto del tirocinio riguarda la creazione di un canale di comunicazione che sia in grado di trasferire nella maniera il più possibile automatica informazioni di pubblica utilità prodotte dal Comune di Rovigo (memorizzate su database di proprietà dell'ente, quindi *legacy*) verso il portale web multiente di nuova generazione MyPortal 2.5, attualmente in fase di collaudo (al Luglio 2011), e che verrà successivamente adottato in via ufficiale e definitiva dall'ente per la pubblicazione delle informazioni destinate alla cittadinanza e altri servizi online.

Il Comune di Rovigo è quindi il committente del progetto qui descritto, e chiede come risultato finale di poter avere un sistema automatico di pubblicazione degli atti amministrativi sul moderno MyPortal 2.5 senza che vengano apportate modifiche a sistemi e procedure preesistenti.

Gli obiettivi parziali e sequenziali del progetto possono quindi essere così riassunti:

- analisi delle strutture dati e delle procedure con cui gli atti amministrativi di interesse vengono memorizzati nel sistema informativo dell'ente Comune di Rovigo;
- analisi delle strutture dati e delle procedure con cui gli atti amministrativi di interesse verranno archiviati e pubblicati sulla nuova piattaforma online MyPortal 2.5 (messa a disposizione da Regione Veneto);
- progettazione e realizzazione di un canale per lo scambio di dati online tra i due enti sopracitati;
- progettazione e realizzazione di un modulo applicativo in grado di prelevare i dati dal sistema informativo del committente, adeguarli alle strutture dati richieste da MyPortal, ed inviarli online tramite il canale dati ideato al punto precedente;
- progettazione e realizzazione di un metodo per la schedulazione dell'esecuzione del modulo applicativo realizzato al punto precedente;

2 - Situazione di Partenza

Le informazioni di pubblica utilità su cui si è incentrato il lavoro svolto sono le delibere di giunta, le delibere di consiglio e le determinazioni del Comune di Rovigo. Tali atti amministrativi sono periodicamente registrati in formato digitale dal personale addetto tramite l'applicativo E-Praxi in uso all'ente stesso. E-Praxi è una soluzione web per la gestione documentale e procedurale degli atti amministrativi e per la conservazione digitale di documenti. Si basa su DB Oracle ed è un prodotto di Engineering Ingegneria Informatica S.p.A., partner tecnologico designato dall'ente per protocollo, atti amministrativi e gestione tributaria. La frequenza con cui i nuovi atti sono registrati a DB è giornaliera, tranne ovviamente nei giorni in cui non viene prodotto nessun nuovo atto.

Il portale di nuova generazione MyPortal 2.5 è invece basato su licenza *open source*, ma viene fortemente personalizzato da Engineering per conto della Regione Veneto, al fine di adattarlo alle esigenze dei vari enti che vogliano aderire al progetto di adozione della nuova piattaforma. Il progetto di adozione di MyPortal 2.5 è stato avviato dalla Regione Veneto, che offre adesione gratuita (tranne per quanto concerne gli ovvi costi indiretti) ai vari enti, quali Comuni e Province, per incentivare l'interoperabilità informatica tra enti e l'uniformazione dei servizi erogati a tutti i cittadini della Regione. I contenuti di MyPortal 2.5 sono gestiti dal CMS Alfresco, al quale i redattori possono accedere da interfaccia di back-end per la pubblicazione di contenuti che possono essere fortemente strutturati (sezioni classiche quali bandi di concorso) oppure no (pagine Html di varia natura, dai contenuti di struttura non prevedibile).

Allo stato iniziale, cioè nel momento in cui è iniziato il tirocinio qui descritto, non esisteva nessun punto di contatto tra le soluzioni informatiche in uso al Comune di Rovigo e MyPortal 2.5. Non esisteva cioè nessun tipo di canale dati, né di procedura di export/import per trasferire verso MyPortal le informazioni di pubblica utilità già precedentemente archiviate, redatte o pubblicati sui sistemi dell'ente stesso.

Al sottoscritto tirocinante è stata offerta come postazione di lavoro un PC desktop, collegato alla LAN aziendale di AS2, da cui poter accedere al dominio aziendale attraverso un proprio account. Su tale workstation sono stati realizzati i prototipi della parte applicativa e i collaudi che hanno preceduto la fase di funzionamento. Dalla stessa postazione è inoltre disponibile un accesso ad Internet, limitato dal firewall aziendale alle sole comunicazioni Http (e Https) standard. E' stato inoltre concesso un accesso in sola lettura al DB contenente gli atti amministrativi tramite LAN, con la possibilità di scegliere liberamente modalità e i software per la formulazione delle query, previa autorizzazione da parte del DBA (DataBase Administrator). Si è inoltre ipotizzato fin dall'inizio che il canale dati per lo scambio di informazioni tra Comune di Rovigo e MyPortal sarebbe passato attraverso Internet e si sarebbe basato su architettura SOA (Service-Oriented Architecture), ma le specifiche e le metodologie sono state via via concordate in corso d'opera con il personale di Engineering ed i sistemisti di AS2, lasciando al tirocinante la parte di implementazione.

2.1 - Contesto aziendale

AS2 S.r.l. è una società pubblica che si occupa di servizi strumentali per Pubbliche Amministrazioni. Il Comune di Rovigo ne è socio unico iniziale, e la società può essere partecipata solo da amministrazioni quali Comuni, Province, Regioni, Consorzi, Unioni, che le affidano i servizi da svolgere esclusivamente a favore degli stessi enti partecipanti ed affidanti. Nasce da una scissione di ASM Rovigo S.p.A., società a totale partecipazione del Comune di Rovigo, da cui AS2 riceve il personale, le infrastrutture, le esperienze ed i contratti attivi e passivi. AS2 eroga servizi strumentali alla Pubblica Amministrazione Locale, secondo la normativa specifica (Legge N° 248 del 4 Agosto 2006, cosiddetta "Legge Bersani"). Tali servizi vengono erogati secondo modalità concordate sulle specifiche esigenze dell'ente locale (con presenza, anche continuativa, presso la sede dell'ente, se richiesto, o a distanza, dal Centro Servizi di AS2), spesso in collaborazione con partner leader di mercato per gli specifici servizi richiesti. La partecipazione alla società consente quindi di affidare i servizi ad una propria azienda, controllata secondo le normative vigenti (cosiddetto *in-house providing*), ottenendo un risultato a costi contenuti, con modalità operative concordate e personalizzate, a cui si somma il valore aggiunto della presenza in loco. L'azienda attualmente conta più di trenta dipendenti ed offre servizi a più di venti PA.

Si noti dunque che essendo il Comune di Rovigo il committente del progetto qui descritto, ed AS2 S.r.l. l'azienda ospitante del relativo tirocinio, questo equivale, dal punto di vista del committente, allo sviluppo *in-house* di una soluzione software.

2.2 - Pubblicazione e consultazione degli atti amministrativi prima di MyPortal 2.5

Prima di MyPortal 2.5 l'ente Comune ci Rovigo aderì ad un progetto simile, anch'esso nato in seno alla Regione Veneto nel 2004. Tale progetto, conosciuto ora come MyPortal 1.0, prevedeva l'adozione di un portale multiente con licenze proprietarie il cui CMS era Oracle Portal. Ogni ente aderente dovette però effettuare un'installazione indipendente del portale sui propri sistemi. Questo comportava grandi costi diretti per gli enti coinvolti (pagamento delle licenze annuali ad Oracle, amministrazione del CMS e personalizzazione della parte dinamica del portale) e rendeva di fatto inutile la multiutenza del portale.

Nel caso specifico del Comune di Rovigo inoltre, le delibere di giunta, le delibere di consiglio e le determinazioni erano consultabili tramite una pagina web, raggiungibile da tale portale via link URL, ma che non si basava sul CMS del portale stesso. Si trattava infatti di una pagina web dinamica sviluppata in-house tramite linguaggio ColdFusion, che raccoglieva i dati di interesse direttamente dal DB di E-Praxi. Il server ColdFusion dell'ente è infatti interfacciato con il DB di E-Praxi e vi può formulare query sulla base delle richieste generate dalla pagina web invocante, dando in risposta una pagina web contenente i risultati della query.

Riassumendo:

1 - Il browser web di una generica postazione client effettua, attraverso Internet, la richiesta di una pagina .cfm (ColdFusion) al web server dell'ente. Si tratta di una "semplice" richiesta Http attraverso URL web;

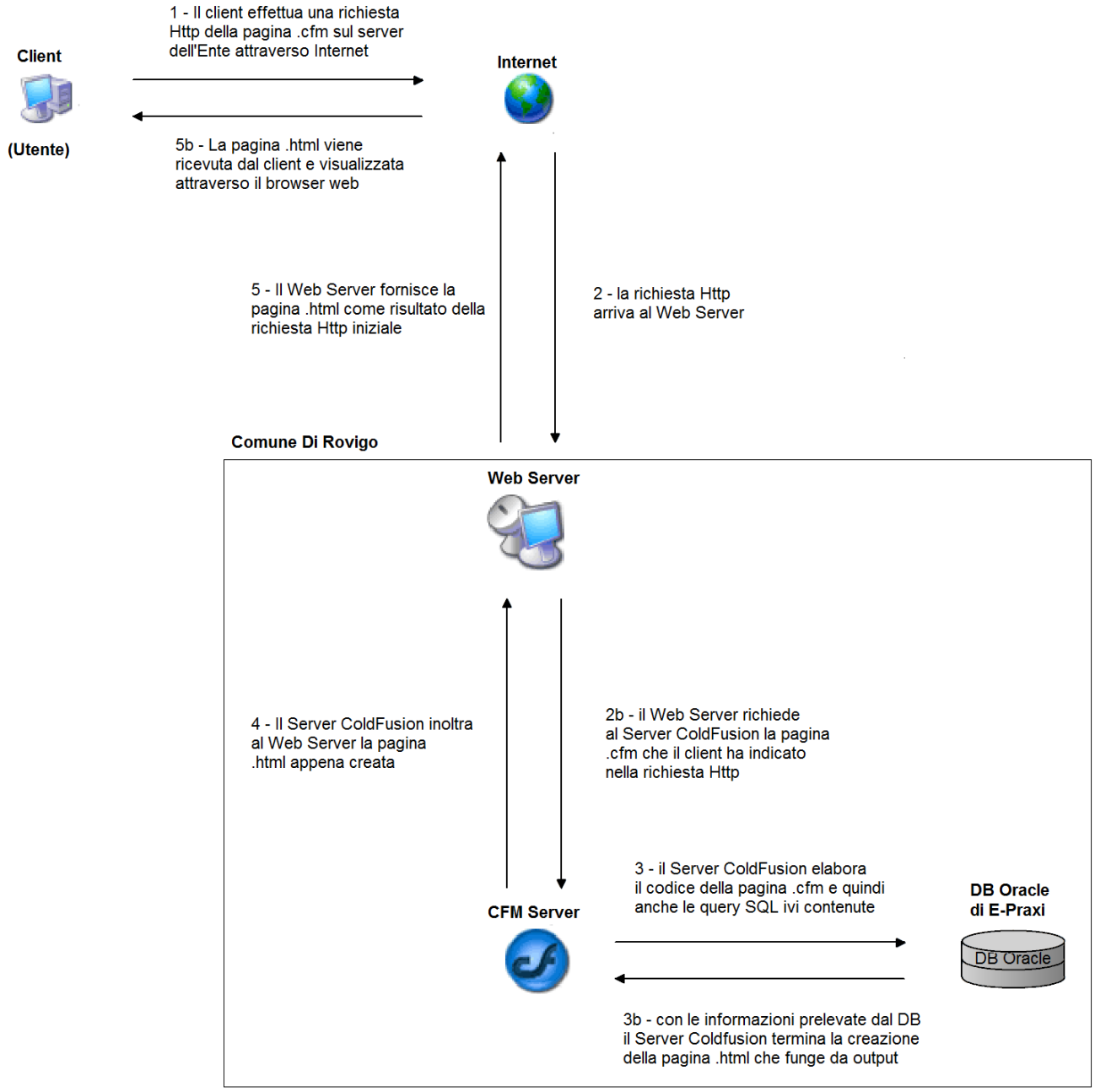
2 - Il web server dell'ente riceve la richiesta della pagina .cfm e inoltra la richiesta al server ColdFusion;

3 - Il server ColdFusion elabora la richiesta, eseguendo il codice che è contenuto nella pagina .cfm (che non è in nessun modo visibile dalla rete Internet). Nell'elaborare il codice incontra una o più query SQL che andrà ad eseguire sul DB di E-Praxi attraverso opportuni driver che sono stati opportunamente preinstallati. Una volta raccolte le informazioni dal DB, le inserisce in una pagina web .html creata dinamicamente (secondo le restanti istruzioni di codice ColdFusion);

4 - Il server ColdFusion inoltra la pagina web dinamica (creata al punto precedente) al web server;

5 - Il web server restituisce al browser del client invocante la pagina web dinamica creata dal server ColdFusion. Si noti che anche se nella barra di indirizzo del browser del client risulta l'indirizzo di una pagina .cfm, quella visualizzata è solo la controparte .html di tale pagina, creata in base al codice ColdFusion contenuto nella pagina .cfm che il client invece non potrà mai nemmeno visualizzare.

A seguire uno schema che aiuta la comprensione dei passaggi appena descritti:



Schema 1: la consultazione degli atti amministrativi del Comune di Rovigo prima di MyPortal 2.5

3 – MyPortal e il Progetto SIRV-Interop

Lo sviluppo di MyPortal 2.5 è stato ideato al fine di rendere il progetto SIRV-Interop una realtà. La realizzazione di tutti paradigmi di interoperabilità previsti dal progetto è un processo elaborato, costoso e non sempre in effetti realizzabile in tutti i suoi aspetti. Tuttavia, il progetto MyPortal, al Luglio 2011, già incorpora in sé quelle che sono le funzioni e i concetti basilari necessari alla costruzione del circuito di interoperabilità, soprattutto per quanto riguarda il paradigma delle Porte di Dominio. Nel capitolo presente verrà presentata una panoramica su quelli che sono i principali scopi dell'adesione ai progetti MyPortal e SIRV-Interop da parte delle Pubbliche Amministrazioni della Regione Veneto.

3.1 – Il progetto SIRV-Interop

Il progetto SIRV-Interop prende avvio in risposta al primo bando nazionale sull'E-Government del Ministero per l'Innovazione e le Tecnologie, inserendosi nel panorama tracciato dalla Legge Regionale n. 54/88 prima e dal Piano di Sviluppo Informatico e Telematico della Regione e dal Piano di Sviluppo della Società Veneta dell'Informazione poi (Legge Regionale 11/2001).
Con SIRV-Interop l'amministrazione regionale ha voluto, nell'ambito delle attività di informatizzazione degli enti locali, svolgere un ruolo di:

- guida e coordinamento a vantaggio di processi di innovazione tecnologica, organizzativa e metodologica della Pubblica Amministrazione locale;
- supporto e sostegno con azioni concrete all'ammodernamento delle Pubbliche Amministrazioni locali.

L'amministrazione regionale, nell'ambito del progetto SIRV-Interop, è andata a creare una piattaforma in grado di connettere tra loro i diversi domini degli enti locali, in modo sicuro e controllato, inserendosi come soggetto promotore e mediatore non intrusivo del sistema di cooperazione ed interoperabilità verso il territorio (ogni ente ha mantenuto infatti la propria autonomia a livello di sistemi informativi, in particolare di applicativi utilizzati).
Nello specifico, Regione Veneto è andata a definire le basi, infrastrutturali e procedurali, necessarie per attivare una comunicazione diretta tra sistemi. Tutto ciò con l'obiettivo di favorire l'integrazione e la cooperazione tra enti, così come la standardizzazione delle modalità di comunicazione e scambio dati e la semplificazione dei procedimenti coinvolti, con impatti in termini di efficienza interna e dell'intero sistema degli enti regionali, nonché di minimizzare il peso degli investimenti da parte degli stessi. Con la piattaforma SIRV-Interop la Regione Veneto si è fatta carico di soddisfare i requisiti di standardizzazione emergenti a livello nazionale ed internazionale, sia in input che in output.

SIRV-Interop e autenticazione

Un sistema di autenticazione deve garantire l'accesso a determinati servizi/informazioni esclusivamente a persone e/o domini riconosciuti e certificati. In un sistema circoscritto (LAN, WAN) quale la rete aziendale di una organizzazione l'accesso ai servizi avviene, di solito, identificando e registrando gli utenti ad uno a uno. Tale soluzione evidenzia tutti i suoi limiti in un sistema aperto in cui ogni organizzazione deve poter scambiare informazioni con altre. Il numero di utenti aumenta esponenzialmente con l'aumentare delle organizzazioni che aderiscono al circuito e diventa ingestibile l'identificazione degli accessi adottata per la modalità intranet.

Se da un lato è necessario garantire la sicurezza, nel contempo è indispensabile rendere leggera la gestione degli accessi. A tale requisito il progetto SIRV-Interop ha risposto adottando una metodologia detta di "asserzione di ruolo" che consente di indicare, per ogni utente sia esso una persona o un computer, le mansioni svolte e la carica ricoperta nell'organizzazione. L'asserzione di ruolo accompagnerà la persona in tutte le fasi di accesso ai servizi sia interni che di altre organizzazioni. L'organizzazione che riceve la richiesta di servizio è garantita dell'autenticità del richiedente, e del ruolo da questi posseduto, dall'organizzazione a cui quel richiedente appartiene e della cui identità quest'ultima garantisce. Spetterà poi all'organizzazione erogatrice verificare se il ruolo posseduto dal richiedente (non l'identità) soddisfa le proprie politiche di sicurezza e quindi fornire o meno il servizio. La gestione e la verifica dell'identità rimane quindi in carico dell'organizzazione a cui appartiene l'utente richiedente il servizio.

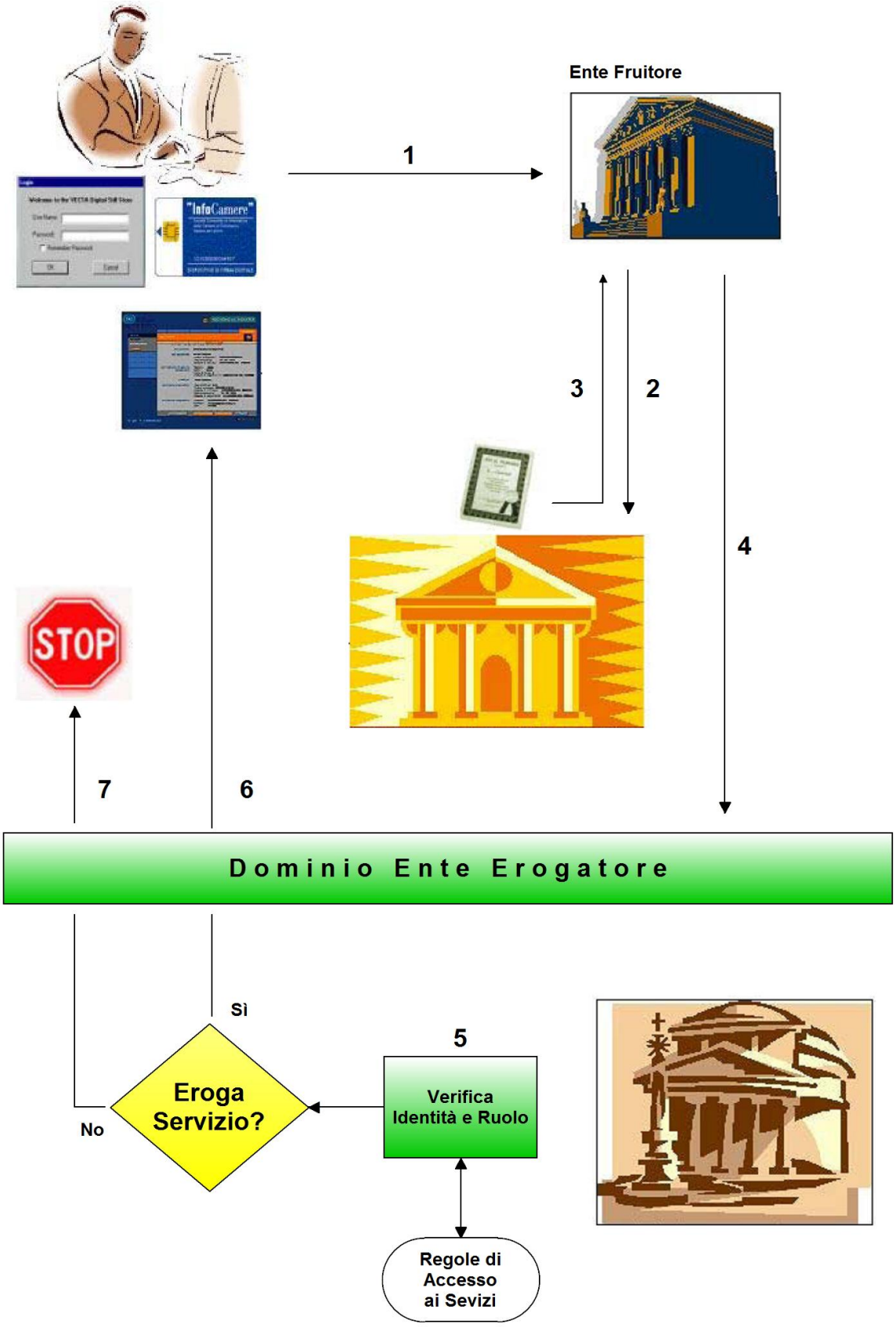
Riassumendo l'accesso alle informazioni può avvenire mediante due modalità:

- autorizzazione di ruolo
- autorizzazione individuale

Autorizzazione di ruolo

Come già detto, l'asserzione di ruolo permette che soggetti/organizzazioni che non si conoscono, ma che riconoscono una autorità di autenticazione/asserzione comune, di consentire l'accesso sicuro ai propri servizi, il tutto realizzato all'interno di un circuito di fiducia certificato dall'organo di autenticazione/asserzione sopraccitato.

A seguire uno schema che può aiutare la comprensione del concetto di Autorizzazione di Ruolo:



Schema 2: Autorizzazione di Ruolo

Descrizioni degli step dello schema 5:

1 - Il Sig. Mario Rossi appartiene ad una organizzazione (ente fruitore) presso la quale sono conservati i suoi estremi di identità. Per le sue attività istituzionali il sig. Rossi deve poter consultare informazioni presenti presso altre organizzazioni. A tale scopo, mediante una smart-card (autenticazione forte) o la coppia username e password (autenticazione debole), si attesta al proprio dominio. L'ente fruitore provvede ad autenticare il sig. Rossi in base al proprio sistema di identificazione e quindi genera un'asserzione di identità.

2 - Sulla base dell'asserzione di identità l'ente fruitore si fa rilasciare l'asserzione di ruolo da un'Autorità preposta. Tale organismo è riconosciuto da entrambi gli enti (fruitore ed erogatore).
Per i ruoli interni questa Autorità coincide con l'organizzazione stessa.

3 - L'Autorità restituisce l'asserzione di ruolo.

4 - L'ente fruitore allega alla richiesta anche l'asserzione di ruolo appena ricevuta, cioè attesta che si tratta di una persona identificata e appartenente alla propria organizzazione e che ricopre un determinato ruolo (ad esempio: è un ufficiale d'anagrafe, un medico responsabile dei referti medici, è un agente di polizia con compiti d'indagine, è un magistrato ecc.)

5 - La richiesta così confezionata giunge all'ente erogatore il quale è garantito sull'identità della persona dall'ente fruitore tramite dall'asserzione di ruolo contenuta nel certificato rilasciato dall'Autorità di autenticazione e asserzione. Verifica il ruolo confrontandolo con le regole di accesso ai servizi.

6 - Se il ruolo ricoperto soddisfa le regole di accesso, il richiedete otterrà l'erogazione del servizio.

7 - Se il ruolo ricoperto non soddisfa le regole di accesso, il richiedente otterrà una risposta di non accessibilità al servizio.

Autorizzazione individuale

L'operatore della sicurezza di un'organizzazione erogatrice, oltre a configurare il diritti d'accesso per i servizi può inserire anche autorizzazioni individuali. L'autorizzazione individuale è più forte dell'autorizzazione di ruolo.

E' infatti possibile inserire autorizzazioni individuali, o *ad personam*. Tali autorizzazioni sono specifiche di un servizio, il che comporta l'immissione, per uno stesso utente, di tante autorizzazioni quanti sono i servizi sui quali lo si vuole abilitare. E' evidente che in questa ipotesi la gestione per l'organizzazione è di gran lunga più onerosa (inserimento, revoca ecc.).

Organizzazioni abilitate

La Regione Veneto, quale supervisore dell'infrastruttura, ha il compito di riconoscere e abilitare le organizzazioni che fanno parte del circuito di interoperabilità. Ciò avviene mediante il rilascio da parte di Regione Veneto di una asserzione firmata digitalmente che, una volta resa disponibile nell'indice regionale, potrà essere utilizzata dall'organizzazione per validare la propria Porta di Dominio e le informazioni che verranno attraverso di questa erogate.

Il sistema di autenticazione prevede che la gestione del riconoscimento del personale assegnato ad una organizzazione sia a carico di quest'ultima. Le modalità di autenticazione offerte dalla piattaforma SIRV-Interop prevede una serie di adempimenti formali a cui adempiere in fase di adesione al circuito di interoperabilità, lasciando poi libertà di scelta nella gestione dei propri utenti.

3.2 – Il progetto MyPortal

Al fine di favorire un importante processo di riforma della Pubblica Amministrazione che prevede l'introduzione di nuove tecnologie e l'ammodernamento dei sistemi informativi, Regione Veneto ha promosso la redazione del progetto denominato "MyPortal". Tale progetto ebbe inizio sul territorio del Comune di Feltre, per essere poi esteso al territorio della provincia di Belluno ed infine a quello dell'intera Regione.
Nello specifico, il progetto si prefigge di incentivare la relazione tra Pubblica Amministrazione Locale ed utente finale, mettendo a disposizione, sia dei cittadini sia delle imprese, uno strumento di accesso ai servizi erogati dalla Pubblica Amministrazione che possa essere facilmente personalizzabile.

I tempi del progetto MyPortal

Il progetto MyPortal ha avuto formalmente inizio l'1 Gennaio 2003. Nel Luglio 2003 è iniziata la fase di sperimentazione della prima versione di MyPortal che ha coinvolto un gruppo di utenti della Comunità Feltrina. Dopo l'adesione di varie Comunità Montane della Provincia di Belluno, cominciarono ad aderire anche molti Comuni di tutta la Regione, soprattutto nelle province di Belluno, Verona e Rovigo. Da allora ebbero luogo varie "revisioni" del progetto MyPortal, che vennero denominate con un codice simile a quelli che vengono comunemente usati per i rilasci di applicativi software. Ad oggi (Luglio 2011) il Comune di Rovigo vuole aderire a ciò che viene denominato "MyPortal 2.5".

I servizi di MyPortal:

Il progetto si basa sostanzialmente sulla costruzione di un portale generale in cui le amministrazioni locali metteranno a disposizione informazioni di dominio pubblico.
Sul medesimo portale verrà data ad ogni cittadino l'opportunità di costruirsi un proprio portale personale attraverso il quale accedere, via rete Internet, alle informazioni di proprio interesse, sia ricorrenti che estemporanee. Il servizio andrebbe a beneficio soprattutto di soggetti distanti dalla normale sede di residenza, ad esempio per esigenze di lavoro o personali. Il portale dovrà essere inoltre costruito con modalità tali da soddisfare le esigenze anche delle fasce più deboli della comunità, quali persone anziane o portatori di handicap, oltre che della piccola e media impresa.

MyPortal oltre a fornire informazioni sull'attività degli enti e la consultazione di varie informazioni di pubblico dominio, vuole fornire una serie di servizi online, quali:

- **Autocertificazioni**
  Il servizio delle autocertificazioni è stato il primo ad essere implementato fin dalla prima versione di MyPortal;
- **Erogazione del servizio ICI**
  Attraverso lo sportello telematico ICI l'utente, precedentemente registratosi, può interrogare la posizione personale, valutare le aliquote, confrontare i dati dichiarati con i dati catastali a disposizione del comune, riscontrare incoerenze e segnalarle agli uffici di competenza;

- **Concessione edilizie**
  Previa identificazione, il portale deve consentire al cittadino/professionista che ha iniziato questo tipo di procedimento amministrativo, la consultazione dello stato delle pratiche da lui avviate;
- **Personalizzazione**
  Il portale deve consente al cittadino, precedentemente registratosi, la possibilità di personalizzare una parte della home-page creando un "portale personale" di accesso ai servizi di tutta la Pubblica Amministrazione . La home-page sarà divisa in più parti tematiche e una zona sarà destinata alla personalizzazione che consentirà di inserire o togliere servizi.

Altri servizi previsti a corredo di quelli principali sopra elencati sono:

- **Motore di ricerca**
  Il progetto prevede la realizzazione di un motore di ricerca che consenta di reperire velocemente e facilmente tutti i documenti pubblicati nel portale nelle diverse sezioni (gare, appalti, delibere, servizi, avvisi, ecc.);
- **Multicanalità**
  Per costruire un rapporto adeguato tra ente e cittadino, sarebbe preferibile utilizzare il canale più adatto, che dovrebbe essere scelto dal cittadino stesso. Per questo motivo tra i diversi servizi che possono essere attivati sul portale e che danno informazioni al cittadino, dovrebbe essere possibile attivare una comunicazione multicanale ente-cittadino (SMS, fax, e-mail, voce, WAP). I servizi coinvolti riguardano per esempio cinema, scadenze, mostre, spettacoli e avvisi comunali;
- **Autenticazione e single-sign-on**
  Il portale deve progettato e realizzato seguendo tutti gli standard di sicurezza per garantire l'integrità dei dati sensibili. Per consentire al singolo cittadino di accedere a servizi specifici o personali, l'utente deve essere soggetto ad una fase di autenticazione. Questa fase può essere di due tipi, in base al possesso di una carta di identità elettronica (CIE) oppure di una carta nazionale dei servizi (CNS). Inoltre sarà implementata la funzionalità di single-sign-on che consentirà all'utente di non dover reinserire login e password ogni volta voglia accedere ad un'area protetta, in quanto sarà il sistema che una volta riconosciuto l'utente registrato "inserirà per lui" le password richieste dai vari servizi. Nel caso in cui un cittadino non possegga CIE o CNS, la procedura chiederà la registrazione dell'utente tramite un modulo interattivo dove verranno richiesti dati anagrafici sufficienti all'identificazione. Successivamente verranno comunicati all'utente, tramite un sistema sicuro (posta certificata o posta tradizionale) i dati relativi al proprio nome utente e password che identificheranno in modo univoco l'utente all'interno del portale.

Durante la fase di sperimentazione si prevede la realizzazione di un call-center in grado di fornire supporto per differenti tipologie di quesiti inerenti il portale ed il corretto utilizzo degli strumenti offerti dal sistema. Tale servizio si rivolgerà tanto agli enti locali quanto ai fruitori finali del servizio (cittadini ed imprese).

4 – Tecnologie Utilizzate

4.1 – Web Service ed XML

La W3C definisce un Web Service come “un sistema software disegnato per supportare un’interazione orientata all’interoperabilità tra due elaboratori, attraverso una rete informatica”.
Un WS ha un’interfaccia descritta in un formato processabile da un elaboratore (Web Service Description Language, WSDL). Tramite essa si può interagire con il Web Service nel modo descritto dall’interfaccia stessa, attraverso lo scambio di *messaggi* inclusi in una *busta* (solitamente SOAP) che vengono tipicamente trasportati tramite protocollo Http e formattati tramite lo standard Xml.
Grazie all'utilizzo di standard basati su Xml e tramite un'architettura basata sui WS (detta Service Oriented Architecture - SOA), applicazioni software scritte in diversi linguaggi di programmazione e implementate su diverse piattaforme hardware possono quindi interagire. Questo grazie alle interfacce che esse "espongono" e mediante l'utilizzo delle funzioni che sono in grado di effettuare (i "servizi" che mettono a disposizione). Tali interazioni servono sia per lo scambio di informazioni che per l'effettuazione di operazioni complesse (quali, ad esempio, la realizzazione di processi di business che coinvolgono più aree di una medesima azienda), sia su reti aziendali come anche su Internet: la possibilità dell'interoperabilità fra diversi linguaggi di programmazione (ad esempio, tra Java e Python) e diversi sistemi operativi (come Windows e Linux) è resa possibile dall'uso di standard "aperti" come l’Xml.
La ragione principale per la creazione e l'utilizzo di Web Service è il "disaccoppiamento" che l'interfaccia standard esposta dal Web Service rende possibile fra il sistema utente ed il Web Service stesso: modifiche ad una o all'altra delle applicazioni possono essere attuate in maniera "trasparente" all'interfaccia tra i due sistemi; tale flessibilità consente la creazione di sistemi software complessi costituiti da componenti svincolati l'uno dall'altro e consente una forte riusabilità di codice ed applicazioni già sviluppate.

Pila protocollare dei Web Service

L’insieme dei protocolli di rete utilizzati per definire, localizzare, realizzare e far interagire tra di loro i Web Service è principalmente organizzato in quatto livelli:

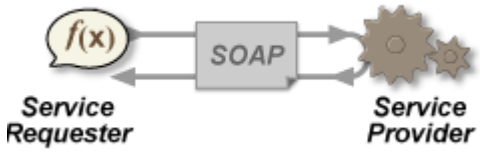
- **Trasporto del servizio**: responsabile per il trasporto dei messaggi tra le applicazioni in rete, include protocolli quali Http, Smtp, Ftp, Xmpp ed il recente Blocks Extensible Exchange Protocol (Beep);
- **Xml Messaging**: tutti i dati scambiati sono formattati mediante tag Xml in modo che gli stessi possano essere utilizzati ad entrambi i capi delle connessioni; il messaggio può essere codificato conformemente allo standard SOAP, come anche utilizzare JAX-RPC, XML-RPC o REST.
- **Descrizione del servizio**: l'interfaccia pubblica di un Web Service viene descritta tramite WSDL (Web Service Description Language) un linguaggio basato su XML usato per la creazione di "documenti" descrittivi delle modalità di interfacciamento ed utilizzo del Web Service.

- **Elencazione dei servizi**: la centralizzazione della descrizione e della localizzazione dei Web Service in un "registro" comune permette la ricerca ed il reperimento in maniera veloce dei Web Service disponibili in rete; a tale scopo viene attualmente utilizzato il protocollo UDDI.

Modalità di utilizzo dei Web Service

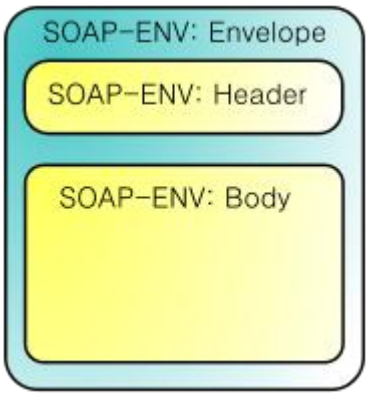
I Web Service sono strumenti che possono essere utilizzati in diversi modi. I più diffusi sono RPC, SOA e REST:

- **RPC** è il modello seguito dai primi strumenti basati su Web Service, ed è perciò un modello largamente installato e diffuso. Tuttavia è stato a volte criticato per non essere un sistema che consenta vero disaccoppiamento. Infatti tale modello era spesso implementato agganciando i servizi a specifiche funzioni o chiamate a metodi di un certo linguaggio di programmazione. In molti avvertirono che questo potesse essere un vicolo cieco per i Web Service e spinsero perché tale modello fosse escluso dal WS-I Basic Profile (specifica erogata dal consorzio Web Service Interoperability, o WS-I);



- **SOA** è un insieme di concetti (o per meglio dire, un *paradigma*) che porta i WS ad essere organizzati secondo una certa architettura, dove l'unità centrale diventa un *messaggio*, piuttosto che un'operazione. Solitamente questo è detto un servizio "orientato ai messaggi".
  I WS con modello SOA sono supportati dai maggiori sviluppatori software e analisti aziendali. Diversamente dalla modalità RPC, vi è un maggiore disaccoppiamento, per via dell'attenzione focalizzata sul "contratto" che il WSDL fornisce, più che su i dettagli di implementazione.
- **REST** descrive un'architettura che usa Http o protocolli simili restringendo l'interfaccia ad un set di operazioni standard "ben note" (come Get, Post, Put e Delete nel caso di Http).
  Qui l'attenzione è concentrata su delle "risorse rappresentabili", più che su messaggio o sulle operazioni. Un'architettura basata su REST (o "RESTful") può usare WSDL per descrivere lo scambio di messaggi SOAP tramite Http, può essere implementata come un'astrazione puramente al di sopra di SOAP (ad esempio WS-Transfer), o può essere usata esclusivamente senza SOAP.

SOAP ed Xml



SOAP (acronimo di Simple Object Access Protocol) è un protocollo leggero per lo scambio di messaggi tra componenti software. La parola "object" manifesta che l'uso del protocollo dovrebbe effettuarsi secondo il paradigma della programmazione orientata agli oggetti.
SOAP è una struttura operativa (framework) estensibile e decentralizzata che può operare sopra varie pile protocollari per reti di computer. Le invocazioni di procedure remote e lo scambio dati nelle architetture SOA possono essere modellizzati come scambio di messaggi SOAP. SOAP dunque è uno dei protocolli che realizzano i Web Service.
SOAP può operare su differenti protocolli di rete, ma Http è il più comunemente utilizzato e l'unico ad essere stato standardizzato dal W3C. SOAP si basa sul metalinguaggio Xml e la sua struttura segue la configurazione Head-Body, analogamente ad HTML.

SOAP può essere utilizzato in due modi:
1) Richiesta via SOAP di parametri: Il client controlla nel Service Registry l'oggetto d'interesse e sviluppa il messaggio secondo i parametri contenuti nel Service Registry.
2) General Purpose Messaging: un programmatore può sviluppare un suo protocollo privato, il client conosce a priori i parametri e non necessita di consultare il Service Registry. All'interno del body del messaggio inserisco i dati scritti nel formato concordato con lo sviluppatore.

Xml (sigla di eXtensible Markup Language) è un metalinguaggio di markup, ovvero un linguaggio marcatore che definisce un meccanismo sintattico che consente creare tag personalizzati. Rispetto ad Html, l'Xml ha uno scopo ben diverso: mentre il primo definisce una grammatica per la descrizione e la formattazione di pagine web (o più in generale, di ipertesti) il secondo è un metalinguaggio utilizzato per creare nuovi linguaggi, atti a descrivere documenti strutturati. Mentre l'Html ha un insieme predefinito di tag, con l'Xml è invece possibile definirne di propri a seconda delle esigenze. L'Xml è oggi molto utilizzato anche come mezzo per la migrazione dati tra diversi DBMS.

Per poter essere correttamente interpretato da un browser, un documento Xml deve essere ben formato, deve cioè possedere le seguenti caratteristiche:

- un prologo, che è la prima istruzione che appare scritta nel documento. Ad esempio: <?xml version="1.0" encoding="ISO-8859-1"?>;
- un unico elemento radice (ovvero il nodo principale, chiamato *root element*) che contiene tutti gli altri nodi del documento;
- All'interno del documento tutti i tag devono essere bilanciati.

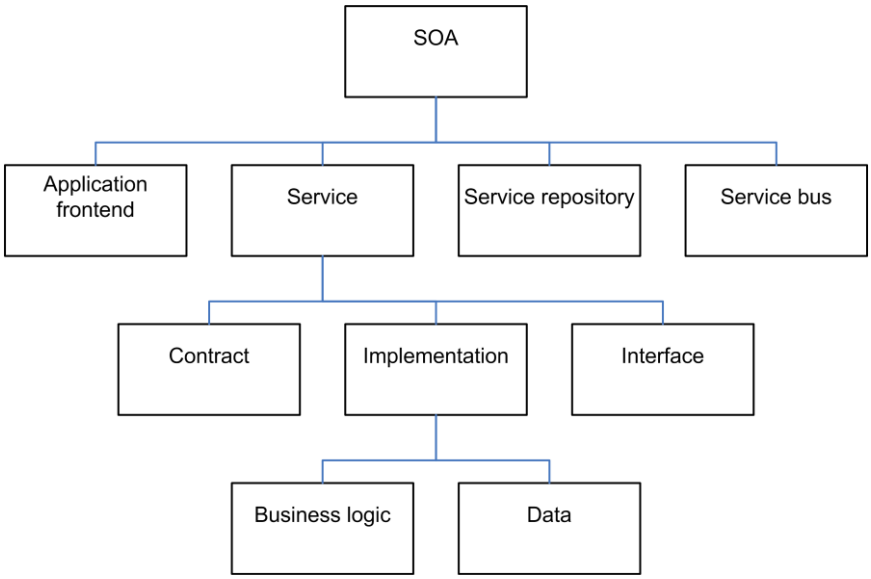
Se il documento Xml non contiene errori si dice *well-formed*. Se il documento è well-formed e in più rispetta i requisiti strutturali definiti nel file DTD o XSD associato, diventa *valid*.

L'Xml Schema o Schema Xml è l'unico linguaggio di descrizione del contenuto di un file Xml che abbia fino ad ora raggiunto la validazione (la 1.1) ufficiale del W3C.
Come tutti i linguaggi di descrizione del contenuto Xml, il suo scopo è delineare quali elementi sono permessi, quali tipi di dati sono ad essi associati e quale relazione gerarchica hanno fra loro gli elementi contenuti in un file Xml.
Gli Xml Schema Document hanno usualmente estensione .xsd. Non è stato ancora registrato uno specifico MIME-type per i documenti Xsd, ergo i tipi MIME solitamente utilizzati per questi documenti sono "application/xml" o "text/xml".
Tecnicamente, uno schema Xml è una collezione di metadati. A differenza delle DTD, gli XML Schema permettono la validazione del contenuto di un elemento o di un attributo rispetto a un determinato tipo di dato. Per esempio, un attributo potrebbe essere vincolato alla memorizzazione esclusiva di una data valida, o di un numero decimale. Xsd fornisce quindi un insieme di 19 tipi di dati primitivi (boolean, string, decimal, double, float, anyURI, QName, hexBinary, base64Binary, duration, date, time, dateTime, gYear, gYearMonth, gMonth, gMonthDay, gDay, e NOTATION). Xsd consente inoltre la costruzione di nuovi tipi di dato partendo da questi tipi primitivi attraverso tre possibili meccanismi: restrizione (riduzione dell'insieme dei valori permessi), lista (estensione ad una sequenza di valori) ed unione (possibilità di scelta di un valore da differenti tipi). Le specifiche stesse del linguaggio indicano venticinque tipi derivati, ed ulteriori derivazioni di tipi possono essere definite dagli utenti all'interno dei propri schemi.

4.2 – Service Oriented Architecture (SOA)

Una SOA è progettata per il collegamento a richiesta di risorse computazionali (principalmente applicazioni e dati), per ottenere un dato risultato per gli utenti, che possono essere utenti finali o altri servizi. L'OASIS (organizzazione per lo sviluppo di standard sull'informazione strutturata) definisce la SOA così:

"Un paradigma per l'organizzazione e l'utilizzazione delle risorse distribuite che possono essere sotto il controllo di domini di proprietà differenti. Fornisce un mezzo uniforme per offrire, scoprire, interagire ed usare le capacità di produrre gli effetti voluti consistentemente, con presupposti e aspettative misurabili."



Schema 3: componenti SOA, secondo Dirk Krafzig, Karl Banke, e Dirk Slama

Nell'ambito dell'informatica, con la locuzione inglese di Service-Oriented Architecture (SOA) si indica generalmente una "*architettura software adatta a supportare l'uso di servizi Web per garantire l'interoperabilità tra diversi sistemi così da consentire l'utilizzo delle singole applicazioni come componenti del processo di business e soddisfare le richieste degli utenti in modo integrato e trasparente*".

Nell'ambito di un'architettura SOA è possibile modificare, in maniera relativamente semplice, le modalità di interazione tra i servizi, oppure la combinazione nella quale i servizi vengono utilizzati nel processo, così come risulta più agevole aggiungere nuovi servizi e modificare i processi per rispondere alle specifiche esigenze di business: il processo di business non è più vincolato da una specifica piattaforma o da un'applicazione, ma può essere considerato come un componente di un processo più ampio e quindi riutilizzato o modificato.
L'architettura orientata ai servizi è particolarmente adatta per le aziende che presentano una discreta complessità di processi e applicazioni, dal momento che agevola l'interazione tra le diverse realtà

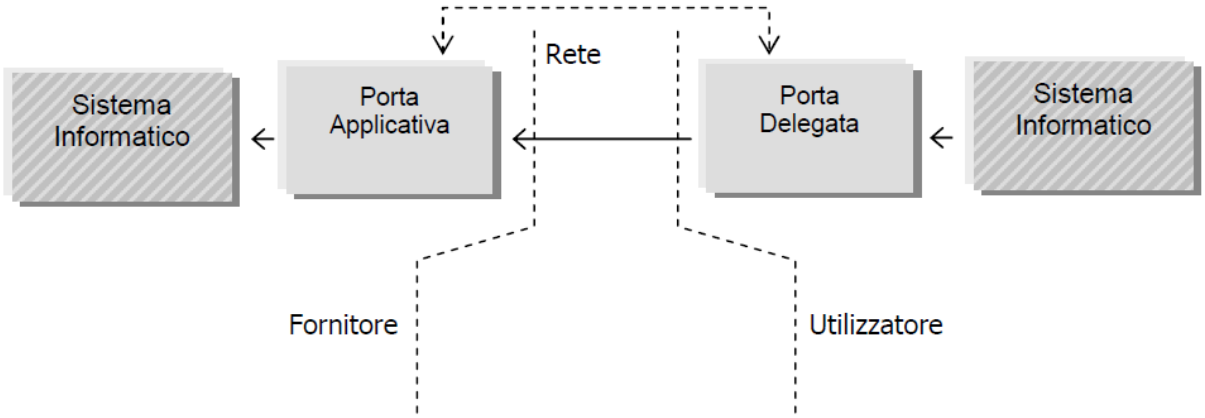
aziendali; permette quindi alle attività di business di sviluppare processi efficienti sia internamente che esternamente, e parallelamente ne aumenta la flessibilità e l'adattabilità.

Benché molte aziende offrano prodotti che possono formare la base di una SOA va sottolineato che la SOA non è un prodotto. Non è affatto vero che un'architettura orientata ai servizi non sia legata ad una specifica tecnologia. Chi dice che può essere realizzata usando una vasta gamma di tecnologie, comprese REST, RPC, DCOM, CORBA, MOM, DDS non ha chiaro il vero valore e l'intrinseco significato delle SOA: nessuna di queste tecnologie è infatti capace di implementare entità che descrivano se stesse come è possibile fare usando i Web Service mediante il linguaggio di definizione dei servizi WSDL. Infatti applicazioni di test che a run-time capiscano la semantica di un servizio e lo invochino senza "conoscere" nulla del servizio stesso "a priori", non esistono per nessuna delle tecnologie citate. La chiave risiede nella totale assenza di business-logic sul client SOA, il quale è totalmente agnostico rispetto alla piattaforma di implementazione, riguardo ai protocolli, al binding, al tipo di dati, alle policy con cui il servizio produrrà l'informazione richiesta.
I servizi che vengono resi disponibili nell'ambito di una SOA infatti sono basati su una definizione formale (o *contract*, ad esempio WSDL) che è indipendente dalla piattaforma sottostante e dal linguaggio di programmazione usato. La definizione dell'interfaccia quindi nasconde l'implementazione dello specifico linguaggio con cui il servizio viene realizzato.

I sistemi basati su SOA possono quindi funzionare in maniera indipendente dalle tecnologie usate per lo sviluppo e dalle piattaforme scelte (come Java, .NET, ecc.). Servizi scritti in C# che girano su piattaforne .NET, servizi scritti in Java che girano su piattaforma Java EE, per esempio, possono essere fruiti da una unica applicazione (o client). Applicazioni che girano su altre piattaforme possono inoltre fruire di tali servizi con il paradigma dei Web Service per facilitarne il riutilizzo. Piattaforme debitamente progettate possono inoltre gestire sistemi legacy, "incapsulandoli" all'interno di WS. Questo ha esteso la longevità di molti sistemi che restano tutt'ora largamente diffusi, indipendentemente dal linguaggio di programmazione che sta alla loro base (che a volte è anche davvero molto vecchio, come ad esempio COBOL).

4.3 – Porte Di Dominio e Web Service

Con il termine *servizio* si intende “una procedura/applicazione rivolta ad una determinata tipologia di utenza che ha l’obiettivo di soddisfare esigenze specifiche di carattere conoscitivo o procedurale”. Il modello delle *Porte di Dominio* si basa sul concetto di “servizio applicativo” erogato da un dominio servente e fruito da un dominio cliente. La richiesta di servizio consiste in un messaggio prodotto da una applicazione di un dominio cliente (ed inviato tramite una propria Porta Delegata) e diretto ad una applicazione di un dominio servente (che lo riceve attraverso una Porta Applicativa). Il messaggio determina l'esecuzione di una applicazione del dominio servente che, in base alle informazioni contenute nel messaggio stesso, esegue una procedura ed invia una risposta destinata all'applicazione cliente.



Schema 4: Porte di Dominio (schema generico)

Per realizzare una relazione di servizio tra domini, è necessario definire un esplicito accordo sulle modalità di erogazione e fruizione del servizio stesso: l'Accordo di Servizio (AS). Questo definisce, tra le altre cose:

- **Identificativo** univoco del **Servizio**;
- **Struttura** del **Servizio**: tipologia, profili di collaborazione utilizzabili;
- **Identificativi** per i possibili **Allegati** di Richiesta/Risposta;
- **Struttura**, in termini di DTD e/o schema, per le **Richieste/Risposte** del servizio, tramite il formalismo XML, anche definendo le eventuali **Eccezioni Applicative**.

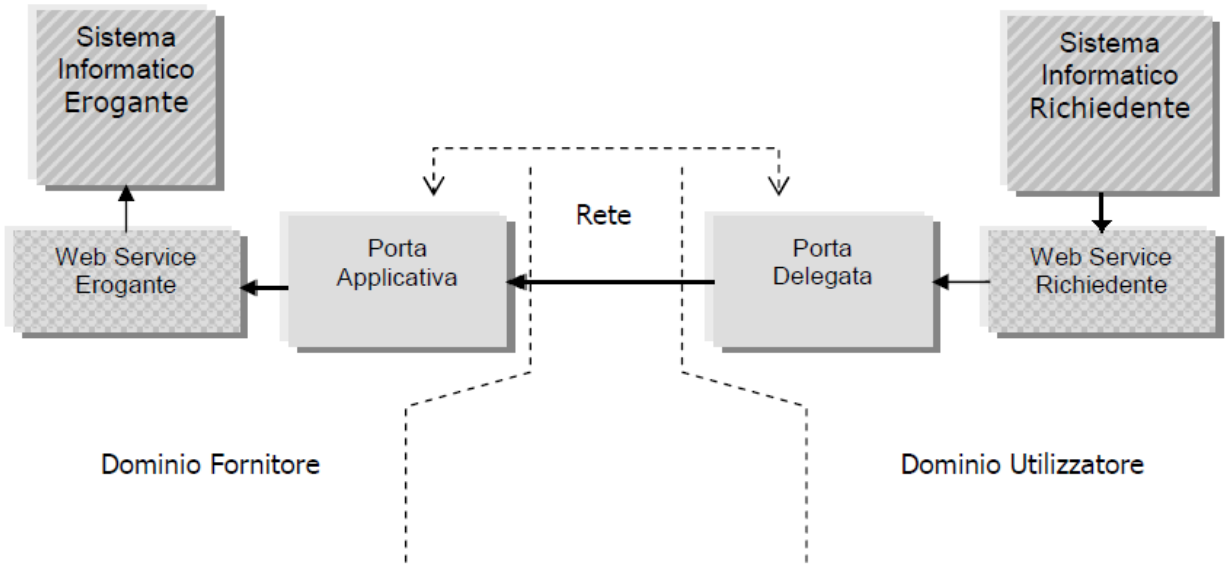
Da un punto di vista tecnico, è possibile interfacciarsi alle Porte di Dominio in diverse modalità:

- con un **Wrapping Diretto**: in tal caso si devono creare delle classi Java che utilizzano direttamente gli oggetti messi a disposizione dalle API delle porte stesse, garantendo il totale controllo sulla preparazione del messaggio telematico;
- con un **Wrapping Remoto**: da utilizzare quando i servizi applicativi siano stati realizzati tramite Web-Service.

Il Wrapping Diretto consiste fondamentalmente nella realizzazione di classi Java che *estendano* quelle classi comuni su cui si basa un circuito di interoperabilità. Ad esempio, nel caso del circuito SIRV-Interop le due classi fondamentali, comuni a tutti gli sviluppatori che dovevano realizzare i servizi applicativi dei vari enti sono PortaApplicativaAsincrona e MessaggioTelematico, contenute nei package interoperabilita.jar e sir.jar distribuiti dalla Regione Veneto.
Il Wrapping Remoto è invece basato su Web Service e si utilizza ogni qualvolta non sia possibile, o non conveniente, interfacciare in modo diretto le Porte di Dominio.

Wrapping Remoto o WS-Wrapping delle Porte Di Dominio

Nel caso si opti di interfacciare due Porte di Dominio con il metodo indiretto, lo schema precedente viene così modificato:



Schema 5: Porte di Dominio (WS-Wrapping)

Nel caso che il fornitore del servizio abbia sviluppato un WS e lo voglia mettere a disposizione di chi ne vorrà fare richiesta, dovrà essere configurata una Porta Applicativa per tale Dominio Fornitore. Relativamente alla Porta Applicativa esisterà un elenco di tutti i WS disponibili, memorizzato in un opportuno file di configurazione. In questo file, per ogni Amministrazione e per ogni servizio, verrà configurato il WS corrispondente.
Anche nel lato Dominio Utilizzatore, al fine di poter rendere il più disaccoppiato possibile l'uso della Porta Delegata, è possibile utilizzare ancora una volta i Web Service. In questo modo sarà il WS ad interfacciarsi con la Porta Delegata, lasciando all'utilizzatore la libertà di realizzare la chiamata a tale WS nel modo che meglio crede, ad esempio in Java, C++, Visual Basic, .NET, o qualsivoglia altro linguaggio.

4.4 – ColdFusion

ColdFusion è una tecnologia server creata da Allaire, ora distribuita da Adobe, che elabora pagine con l'estensione .cfm e .cfml. Si serve del linguaggio di programmazione CFML (ColdFusion Markup Language) , supportato anche da molti altri Java EE application server.
Come tutti i linguaggi di scripting server-side, ad esempio PHP, ASP e Perl, le pagine non necessitano di compilazione, e quindi possono essere modificate con un normale editor di testo, come blocco note di Windows, ma il codice sorgente è comunque non reperibile tramite Http.
Venne rilasciato per la prima volta nel 1995 ed ha ora raggiunto la versione 9 (Ottobre 2009), in cui vengono migliorate la possibilità di inserire componenti AJAX e l'integrazione con le piattaforme Adobe Flash, Adobe Flex e Adobe AIR, grazie all'inclusione dello strumento Flash Builder 4 all'interno di ColdFusion Builder.

4.5 – Eclipse

L’ambiente di sviluppo utilizzato per il modulo Java realizzato nell’ambito del presente progetto è Eclipse. Eclipse può essere utilizzato per la produzione di software di vario genere, si passa infatti da un completo IDE per il linguaggio Java (JDT, "Java Development Tools") a un ambiente di sviluppo per il linguaggio C++ (CDT, "C/C++ Development Tools") e a plug-in che permettono di gestire XML, Javascript, PHP e persino di progettare graficamente una GUI per un'applicazione JAVA (Eclipse VE, "Visual Editor"), rendendo di fatto Eclipse un ambiente RAD.
Il programma è scritto in linguaggio Java, ma anziché basare la sua GUI su Swing, il toolkit grafico di Sun Microsystems, si appoggia a SWT, librerie di nuova concezione che conferiscono ad Eclipse un'elevata reattività. La piattaforma di sviluppo è incentrata sull'uso di plug-in, delle componenti software ideate per uno specifico scopo, per esempio la generazione di diagrammi UML, ed in effetti tutta la piattaforma è un insieme di plug-in, versione base compresa, e chiunque può sviluppare e modificare tali plug-in. Nella versione base è possibile programmare in Java, usufruendo di comode funzioni di aiuto quali: completamento automatico (*code completion*), suggerimento dei tipi di parametri dei metodi e riscrittura automatica del codice (*refactoring*) in caso di cambiamenti nelle classi. Essendo scritto in Java, Eclipse è disponibile per le piattaforme Linux, HP-UX, AIX, Mac OS X e Windows.

4.6 – Apache Tomcat

Apache Tomcat (o semplicemente Tomcat) è un contenitore servlet open source sviluppato dalla Apache Software Foundation. Implementa le specifiche Java Server Pages (JSP) e Servlet di Sun Microsystems, fornendo quindi una piattaforma per l'esecuzione di applicazioni web sviluppate nel linguaggio Java. La sua distribuzione standard include anche le funzionalità di web server tradizionale, che corrispondono al prodotto Apache.
In passato, Tomcat era gestito nel contesto del Jakarta Project, ed era pertanto identificato con il nome di Jakarta Tomcat; attualmente è oggetto di un progetto indipendente.
Tomcat è rilasciato sotto la licenza Apache, ed è scritto interamente in Java; può quindi essere eseguito su qualsiasi architettura su cui sia installata una JVM.

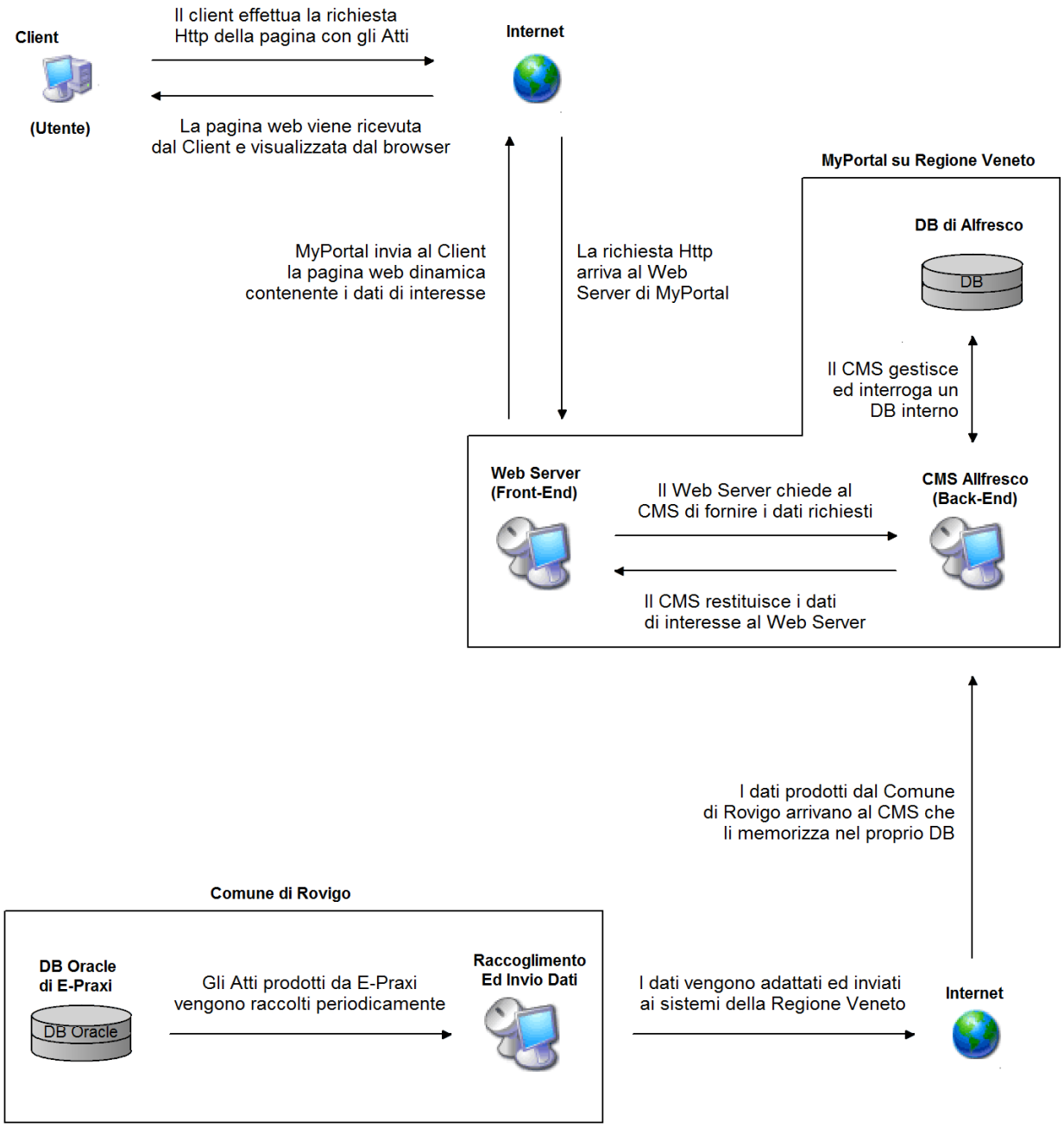
5 – Progettazione e Sviluppo dei Moduli Applicativi

L'adozione di MyPortal 2.5 da parte del Comune di Rovigo comporterà dei notevoli cambiamenti tecnologici e procedurali riguardo la pubblicazione in formato digitale degli atti amministrativi oggetto del presente progetto. Rispetto al sistema illustrato nel paragrafo 2.2 vi sono infatti sostanziali diversità (hardware e software) e nel modo in cui viene realizzato il punto di contatto tra il DB contenente le informazioni di interesse e l'utente finale che desidera consultarle.

Come già detto, MyPortal 2.5 è una piattaforma multiente per la realizzazione di portali web, basata su CMS. L'intero hardware necessario al funzionamento di MyPortal 2.5 verrà ospitato in una server farm della Regione Veneto e non più dal Comune di Rovigo stesso. L'utente finale navigherà le pagine web del portale del Comune di Rovigo connettendosi ai server della Regione Veneto fino a raggiungere le pagine contenenti gli atti amministrativi che desidera consultare. Il sistema quindi genererà una pagina web dinamica, in base alle informazioni che l'utente desidera avere, raccogliendo i dati da un DB interno al sistema stesso e gestito dal CMS. Risulta quindi chiaro che gli atti amministrativi dovranno in qualche modo arrivare dai sistemi del Comune di Rovigo (dove vengono originati ed archiviati) al CMS di MyPortal 2.5 sui server della Regione Veneto. In tale situazione si avrà quindi una copia dei dati presso i sistemi del Comune di Rovigo per l'archiviazione, e una copia dei dati presso i sistemi della Regione Veneto per la consultazione. Tali dati dovranno essere naturalmente sempre allineati tra i due DB ed aggiornati con tempestività sufficiente a non creare disagio per l'utente finale.

E' chiaro quindi che la costruzione del canale dati necessario a tale scopo si inquadra in un contesto di *interoperabilità informatica* tra Pubbliche Amministrazioni. Anche questo aspetto è stato fin da subito esaminato e definito nell'ambito del progetto MyPortal. Più specificamente, si è sempre ipotizzato che sarebbe stato contestualmente realizzato un sistema di *web service* atto a soddisfare l'interoperabilità tra Regione Veneto e gli enti aderenti al progetto MyPortal, cercando di conseguire gli obiettivi del progetto SIRV-Interop di Regione Veneto.

A seguire uno schema concettuale che aiuta la comprensione del risultato desiderato:



Schema 6: la consultazione degli Atti Amministrativi del Comune Di Rovigo con MyPortal 2.5 (concept iniziale)

La realizzazione di un sistema secondo il modello appena descritto è, in definitiva, l'obiettivo finale del progetto descritto nella presente relazione.

5.1 – Analisi delle strutture dati

L'analisi delle strutture dati è stato fin da subito l'ovvio punto di partenza per lo sviluppo effettivo del progetto. In particolare si sono analizzate le strutture dati e la periodicità con cui gli atti amministrativi vengono memorizzati sul DB di E-Praxi, e le specifiche sui dati richieste dal sistema MyPortal (fornite da Engineering). Trovare un modo di adattare i dati perché avvenisse un cambio di struttura senza perdita di informazione era il primo punto da risolvere.

Per quanto Engineering fosse, ai tempi in cui il presente progetto di interoperabilità fu avviato, ancora alle fasi concettuali di ideazione della parte riguardante i Web Service del nuovo MyPortal, fu chiaro fin da subito che lo standard adottato per lo scambio di dati sarebbe stato il formato Xml.

5.1.1 – Analisi delle strutture dati usate dal sistema legacy del Comune di Rovigo

Le delibere di giunta, le delibere di consiglio e le determinazioni del Comune di Rovigo sono registrate nel sistema informativo di AS2 S.r.l. a DB (Oracle) secondo i tracciati in figura 1, 2 e 3:

DELIBERE_GIUNTA_INTRA: Created: 10/02/2009 8.47.44 Last DDL: 17/02/2009 8.52.36

Columns | Indexes | Constraints | Triggers | Data | Scripts | Grants | Synonyms | Partitions | Subpartitions | Stats/Size | Referer

Column Name	I..	Pk	Null?	Data Type	Default	Histogram
LIVELLO2	1		Y	VARCHAR2 (4000 Byte)		Yes
LIVELLO3	2		Y	VARCHAR2 (4000 Byte)		Yes
DES_OGG	3		Y	VARCHAR2 (1000 Byte)		Yes
SIGLA	4		Y	VARCHAR2 (30 Byte)		Yes
ANNO	5		Y	NUMBER (4)		Yes
NUMERO	6		Y	INTEGER		Yes
ID_ATTO	7		Y	INTEGER		Yes
TS	8		Y	DATE		Yes
RIF_FIS_IN_REPOSITORY	9		Y	VARCHAR2 (1000 Byte)		Yes
COD_NATURA_CONTABILE	10		Y	VARCHAR2 (10 Byte)		Yes
FASE_CORRENTE	11		Y	VARCHAR2 (4000 Byte)		Yes
INIZ_PUB	12		Y	VARCHAR2 (10 Byte)		Yes
FINE_PUB	13		Y	VARCHAR2 (10 Byte)		Yes
ID_ATTO2	14		N	INTEGER		Yes
RIF_FIS_IN_REPOSITORY_ALLEGATO	15		Y	VARCHAR2 (1000 Byte)		Yes
ID_TP_DOC_ALL	16		Y	INTEGER		Yes

Figura 1: tracciato DB Oracle delle delibere di giunta

DELIBERE_CONSIGLIO_INTRA: Created: 16/02/2009 13.58.28 Last DDL: 17/02/2009 8.47.07

Columns | Indexes | Constraints | Triggers | Data | Scripts | Grants | Synonyms | Partitions | Subpartitions | Stats/Size | Referen

Column Name	I..	Pk	Null?	Data Type	Default	Histogram
LIVELLO2	1		Y	VARCHAR2 (4000 Byte)		Yes
LIVELLO3	2		Y	VARCHAR2 (4000 Byte)		Yes
DES_OGG	3		Y	VARCHAR2 (1000 Byte)		Yes
SIGLA	4		Y	VARCHAR2 (30 Byte)		Yes
ANNO	5		Y	NUMBER (4)		Yes
NUMERO	6		Y	INTEGER		Yes
ID_ATTO	7		Y	INTEGER		Yes
TS	8		Y	DATE		Yes
RIF_FIS_IN_REPOSITORY	9		Y	VARCHAR2 (1000 Byte)		Yes
COD_NATURA_CONTABILE	10		Y	VARCHAR2 (10 Byte)		Yes
FASE_CORRENTE	11		Y	VARCHAR2 (4000 Byte)		Yes
INIZ_PUB	12		Y	VARCHAR2 (10 Byte)		Yes
FINE_PUB	13		Y	VARCHAR2 (10 Byte)		Yes
ID_ATTO2	14		N	INTEGER		Yes
RIF_FIS_IN_REPOSITORY_ALLEGATO	15		Y	VARCHAR2 (1000 Byte)		Yes
ID_TP_DOC_ALL	16		Y	INTEGER		Yes

Figura 2: tracciato DB Oracle delle delibere di consiglio

DTD_INTRANET_DETERMINE: Created: 25/11/2009 12.59.17 Last DDL: 30/11/2009 11.41.37

Columns | Indexes | Constraints | Triggers | Data | Scripts | Grants | Synonyms | Partitions | Subpartitions | Stats/Siz

Column Name	I..	Pk	Null?	Data Type	Default	Histogram
ID_ATTO	1		N	INTEGER		Yes
NUMERO	2		Y	INTEGER		Yes
TS	3		Y	DATE		Yes
RIF_FIS_IN_REPOSITORY	4		Y	VARCHAR2 (1000 Byte)		Yes
ALLEGATI	5		Y	VARCHAR2 (1000 Byte)		Yes
ID_TP_DOC	6		Y	INTEGER		Yes
LIVELLO2	7		Y	VARCHAR2 (4000 Byte)		Yes
LIVELLO3	8		Y	VARCHAR2 (4000 Byte)		Yes
DES_OGG	9		Y	VARCHAR2 (1000 Byte)		Yes
FASE_CORRENTE	10		Y	VARCHAR2 (4000 Byte)		Yes
ANNO	11		Y	VARCHAR2 (10 Byte)		Yes
SIGLA	12		Y	VARCHAR2 (10 Byte)		Yes

Figura 3: tracciato DB Oracle delle determinazioni

I tracciati qui illustrati sono una rappresentazione dello schema logico (relazionale) del DB da cui bisognerà prelevare i dati di interesse legati agli atti amministrativi che dovranno essere pubblicati online.

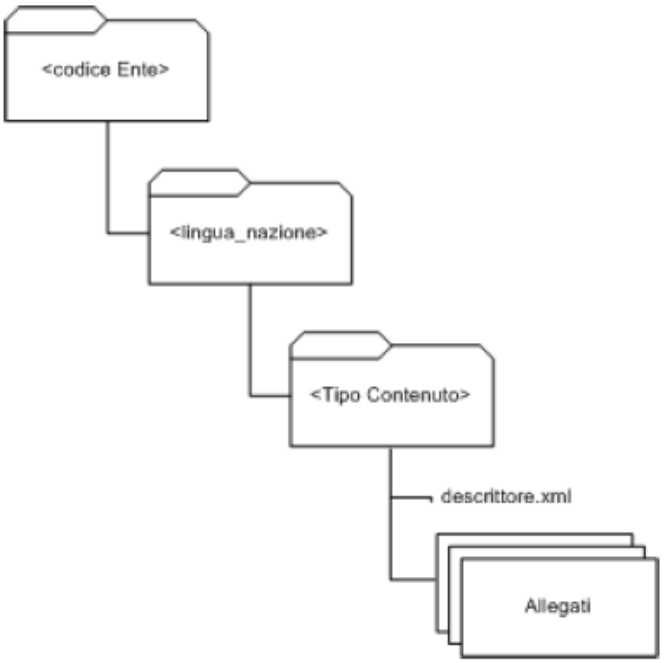
Non è necessaria in realtà nessuna analisi approfondita né dello schema logico, né dello schema concettuale, né della eventuale documentazione associata al DB. Tutto quello che è necessario fare è una semplice *mappatura* dei campi per creare una corrispondenza univoca tra i record del DB e i record che verranno preparati nei file .xml. L'unico aspetto delicato in questo procedimento è il *tipo di dato* con cui ciascun campo viene memorizzato a DB. Nella conversione verso il tipo di dato che sarà invece richiesto dalle specifiche di validazione dei file .xml che si andranno a creare, saranno necessari eventuali *cast* che dovranno essere fatti correttamente e senza perdita di informazione. Infatti, come anche riportato nel capitolo "Tecnologie Utilizzate", se in un file Xml compaiono dei dati in formato non valido (rispetto a quanto descritto nell'apposito Xsd), esso non può essere validato e verranno di conseguenza scartati anche tutti gli altri dati presenti nello stesso file.

5.1.2 – Analisi delle strutture dati usate dai web service di MyPortal 2.5

MyPortal 2.5 è stato adottato anche e soprattutto per il suo orientamento all'interoperabilità, che si manifesta soprattutto nell'utilizzo di Web Service per lo scambio dati con altri sistemi informatici. Il linguaggio che sta alla base dei WS è il linguaggio Xml (vedi capitolo "Tecnologie Utilizzate" per approfondimenti su WS e linguaggio Xml).

Prima che il portale possa essere alimentato dai periodici flussi dati che viaggiano attraverso la rete con i nuovi atti prodotti da ciascun ente, va fatto un primo popolamento massivo del DB di Alfresco, sia a scopo di collaudo e sia perché è chiaramente irragionevole inviare tramite WS tutti gli atti prodotti dal Comune di Rovigo nel periodo precedente all'apertura del nuovo portale. A tale scopo Engineering realizzò da subito un importer "manuale" o per meglio dire, *offline*, per il caricamento massivo dei dati di ciascun ente. Tale importer usa comunque gli standard Xml, quindi una prima grossa parte del progetto si è incentrata sulla costruzione di un modulo applicativo in grado di fare un "dump" dell'intero DB contenente gli atti amministrativi ed adattare il tutto in un formato Xml valido ad essere caricato su Alfresco tramite l'importer offline. Fu chiaro fin dall'inizio che tale lavoro era comunque un buon punto di partenza anche per la parte più strettamente orientata ai WS: si sarebbe cercato, per quanto possibile, di "riciclare" il formato scelto per i file .xml usati in questa prima parte anche per l'utilizzo tramite WS.

Ora vediamo più in dettaglio il formato dei dati richiesto dall'importer offline di MyPortal 2.5.
I dati da caricare tramite importer offline in MyPortal devono essere contenuti in un archivio Zip, rinominato con estensione ".myp". Tale archivio deve essere così strutturato:



Schema 7: alberatura di un generico archivio "myp"

Il numero e la struttura delle cartelle può variare in relazione al tipo di contenuto.

Nello schema 3:
codice_ente: è il codice IPA dell'ente;
lingua_nazione: identifica la lingua nella quale sono scritti i contenuti (esempio: "it_IT");
tipo_contenuto: identifica il contenuto presente nell'archivio (esempio: "sindaco" oppure "giunta_comunale", oppure "delibere_giunta");
descrittore.xml: un file Xml che contiene i dati da importare. La struttura di questo file varia in base al tipo di contenuto;
allegati: file binari logicamente legati al contenuto (esempio: documenti allegati, immagini).

Tale importer è costituito da una classe Java che viene eseguita ad intervalli fissi ("schedulata"). La schedulazione viene affidata a Quartz, che viene utilizzato da Alfresco. La classe schedulata espone un metodo che verifica la presenza, all'interno di un insieme predefinito di cartelle del file system, di flussi dati provenienti dagli enti. Tali flussi si presentano a coppie, nelle quali il nome è fisso ma l'estensione varia: un file ha estensione .md5 e l'altro .myp. Il metodo può essere realizzato a partire dalle classi di Apache Commons IO.

L'importer offline di Alfresco quindi, per ogni file .myp:

- esegue una validazione del relativo checksum;
- sposta e scompatta tale archivio in un'apposita directory su file system;
- ne analizza l'alberatura delle sotto-directory e la trasforma in un file .xml che deve essere valido tramite l'appropriato file .xsd per verificare la correttezza dell'alberatura stessa;
- valida tramite un apposito .xsd i file descrittori (possono essere più di uno);
- valida gli allegati binari, ovvero controlla la corrispondenza biunivoca tra i file dichiarati negli Xml descrittori e i file binari presenti nell'alberatura delle directory;
- esegue il caricamento del contenuto su Alfresco.

A seguire viene proposto, a titolo di esempio, il contenuto del file "validation-base.xsd".
Tale file è incluso anche in tutti gli altri Xsd presenti nel progetto. Non saranno proposti i contenuti di altri file similari nella presente relazione, vista la natura puramente tecnica del loro contenuto.

validation-base.xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
  <!-- *** -->
  <!-- *** MyPortal 2.5 - Importer -->
  <!-- *** schema per la validazione che viene incluso in tutti gli altri xsd  -->
  <!-- *** -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="data">
    <xs:restriction base="xs:string">
      <xs:pattern value="(0[1-9]|[12][0-9]|3[01])/(0[1-9]|1[012])/(19|20)[0-9]{2} ([01][0-9]|2[0-3]):[012345][0-9]" />
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="email">
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}" />
    </xs:restriction>
```

```
  </xs:simpleType>
  <xs:simpleType name="html">
    <xs:restriction base="xs:string">
      <!--xs:pattern value="^&gt;([a-z]+)[^&lt;]*(?&gt;!/)#&lt;"/-->
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="empty">
    <xs:restriction base="xs:string">
      <xs:maxLength value="0" />
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="dataFacoltativa">
    <xs:union memberTypes="data empty" />
  </xs:simpleType>
  <xs:simpleType name="emailFacoltativa">
     <xs:union memberTypes="email empty" />
  </xs:simpleType>
  <xs:simpleType name="notEmptyString">
    <xs:restriction base="xs:string">
      <xs:pattern value="[\s\ta-zA-Z0-9()._\-',\*\\#òàùèié&amp;/=@\[\];.:\?\^%+]+" />
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="notEmptyName">
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z0-9()._\-]+" />
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="booleanString">
    <xs:restriction base="xs:string">
      <xs:pattern value="true|false" />
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="booleanFacoltativo">
    <xs:union memberTypes="booleanString empty" />
  </xs:simpleType>
  <xs:simpleType name="numerico">
    <xs:restriction base="xs:string">
      <xs:pattern value="[0-9]{1,}" />
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="numericoFacoltativo">
    <xs:union memberTypes="numerico empty" />
  </xs:simpleType>
</xs:schema>
```

Il primo approccio per la creazione del modulo di export dei dati del DB Oracle fu fatto utilizzando il linguaggio ColdFusion. Tale linguaggio era già stato utilizzato dai tecnici di AS2, tra cui lo stesso DBA, per la creazione di pagine web dinamiche in grado di formulare query quanto si voglia complesse sul DB. La famigliarità del reparto di sviluppo software di AS2 con tale linguaggio, e il fatto che ColdFusion fosse un linguaggio più duttile e semplice di Java sono state le motivazioni di una scelta che si rivelò poi in effetti vantaggiosa.

5.2 – Sviluppo del modulo di export dati per raffinazioni successive

Il codice ColdFusion può incapsulare query da scriversi nel formato SQL utilizzato dal DB stesso. E' lo stesso server CF ad effettuare, tramite opportuni driver, le interrogazioni a DB, trasformando il multi-set ottenuto come risultato in una variabile CF, ispezionabile tramite appositi paradigmi e metodi del linguaggio di programmazione stesso. Il paradigma dell'Ingegneria del Software scelto per la progettazione di tale modulo di export dati fu quello del *modello prototipale incrementale a raffinazioni successive*.
Furono infatti sviluppati vari prototipi, partendo da uno molto semplice, ma che già realizzava la funzione basilare (senza personalizzazioni né controlli adeguati), cioè quella di interrogare il DB estraendone soltanto le informazioni di interesse. Nei passaggi successivi si eseguivano alcune revisioni parziali del codice e se ne producevano molte nuove righe. Venivano aggiunte mano a mano metodi e procedure dai FP (Function Points) sempre più elevati, ed eseguiti ad ogni passaggio test di varia natura (test di completezza, test di correttezza, test di stress) per garantire che quanto realizzato al punto precedente fosse perfettamente riutilizzabile per i passaggi successivi. Ogni prototipo veniva inoltre proposto al DBA che lo validava da un punto di vista funzionale.

Prototipo 1

Il primo prototipo del modulo di export dati era una semplice pagina web dinamica che mostrava a video la query eseguita dal server CF e il risultato che essa generava. Il multi-set ottenuto veniva trattato come un multi-insieme di dati di tipo String e mostrato a video, senza nessun particolare adattamento o impaginazione. Questo primo modulo consentiva di verificare l'esattezza delle query formulate e la gestibilità (nonché correttezza) del loro output. Le query vennero limitate da un apposito filtro (ad esempio, cercando i soli atti amministrativi appartenenti ad un certo semestre) in modo da effettuare test che non risultassero troppo pesanti per il server CF. In questa fase non si riscontrarono problemi degni di nota ma solo normali operazioni di debug del codice.

Prototipo 2

Il secondo prototipo del modulo di export dati era ancora una volta una pagina web dinamica che mostrava a video la query eseguita dal server CF e il risultato che essa generava. Questa volta però l'output a video combinava l'output delle query formulate con il codice Xml. Grazie a caratteri di andata a capo e una adeguata indentatura era possibile avere riscontro visivo di come sarebbero stati formati i futuri file Xml contenenti i record prodotti dalla query. Da questo passo si cominciava a focalizzare l'attenzione su *come* scrivere nei file Xml i record ottenuti in modo che poi tali file potessero passare la fase di validazione e cominciarono ad essere necessari i primi *cast* di variabile. Il linguaggio CF a tale proposito mette a disposizione dei mezzi davvero potenti che permettono di far risparmiare moltissimo tempo allo sviluppatore. Anche in questa fase non si riscontrarono problemi degni di nota ma solo normali operazioni di debug del codice.

Prototipo 3

Il terzo prototipo era il primo a non mostrare output a video. Questa volta il codice Xml conteneva dei comandi "<cffile>" che permettono, tra le altre cose, di scrivere una stringa all'interno di un file di testo. Da qui in avanti diventa senz'altro più corretto di parlare di *esecuzione* della pagina Cfm anziché della sua *visualizzazione*. Da questo prototipo in poi si è scelto infatti di mostrare a video soltanto un messaggio di avvenuta terminazione di tutte le righe di codice CF e, all'occorrenza (per motivi di debug), le query incapsulate nel codice stesso. I file di testo con estensione .cfm venivano scritti all'interno di una data cartella a file system ed avevano semplici nomi incrementali. Per ispezione dei file Xml prodotti si effettuarono test di correttezza e completezza. Si iniziarono a testare inoltre vari tipi di filtri per limitare l'output della query, ad esempio filtri per numero di atto amministrativo, vari tipi di filtri basati sulla data degli atti stessi, o solo di quelli che contenessero un certo tipo di documento allegato. In questa fase si riscontrarono per la prima volta limitazioni che si rivelarono fastidiose per tutto il resto del progetto, ovvero quelle legate ai permessi di scrittura a file system. Il server CF viene riconosciuto infatti in modi diversi dai computer della LAN aziendale a seconda del sistema operativo. Talvolta la creazione di file su un percorso di rete era impossibile a meno di onerose configurazioni sulle regole di sicurezza da parte dei sistemisti. Fu scelto quindi di far scrivere i file di output tutti in una predefinita cartella del server stesso. Da qui in avanti fu necessaria particolare cautela perché il riempimento dello spazio su disco poteva compromettere le prestazioni di una macchina ritenuta *mission critical* per l'azienda.

Prototipo 4

Il prototipo seguente generava gli stessi file Xml del precedente e cercava inoltre di preparare un'alberatura di directory che rispecchiasse quella richiesta dalle specifiche fornite da Engineering. Contestualmente, si sono anche andati ad inserire nell'alberatura tutti gli allegati binari relativi a ciascun Xml. I path di ciascun allegato binario sono ricostruibili a partire da un particolare attributo della relazione interrogata. I percorsi sono memorizzati a DB secondo un path assoluto della macchina dove risiede il DB stesso, ma per sostituzione di una certa porzione di indirizzo è stato possibile trasformarlo in un indirizzo di rete raggiungibile dal server CF. Infine la copia effettiva del file si sarebbe voluta fare sempre tramite <cffile>, che grazie alla sintassi <cffile action = "copy"> può creare copie di file nelle unità disco e/o percorsi di rete locali. A questo punto ci si accorse però di essere finiti in un vicolo cieco, in quanto il server CF non aveva permessi nemmeno in lettura per i file contenenti gli allegati binari. Il problema fu aggirato con l'ideazione di un file batch (un normale .bat da lanciare in ambiente MS-DOS), creato dinamicamente dal modulo CF, contenente i comandi di copia. Dai test effettuati risultò infatti che tale file batch avviava un processo riconosciuto come lecito dal firewall aziendale, e che riusciva quindi ad ottenere i permessi in lettura dei file. L'inconveniente che si venne a creare fu che risultava quindi necessario concatenare le due esecuzioni, quella della pagina .cfm e quella del file .bat da essa costruito per ottenere il risultato completo.

A tal proposito si può notare la versatilità del linguaggio CF, con uno sguardo a queste righe di codice:

<cfset fileCorrente = replace(fileCorrente,
"C:\PercorsoAssolutoMacchinaDB","Z:\PercorsoDiscoReteRispettoServerCF", "all")>

dove fileCorrente è una variabile di tipo String che contiene il path del file recuperato dal DB. Tale riga di codice consente di sostituire ogni occorrenza di un certo pattern in una certa stringa con un altro pattern a piacimento in maniera molto semplice.
La vera potenza del comando è stata sfruttata in uno dei prototipi per il modulo basato su WS, quando fu necessario cambiare tutti i caratteri di *slash* in caratteri di *backslash* per migliaia di percorsi di rete.
Un altro nuovo, fondamentale comando che si è iniziato ad usare in questa fase è <cfdirectory>, comando che consente la creazione e l'eliminazione di directory su file system.

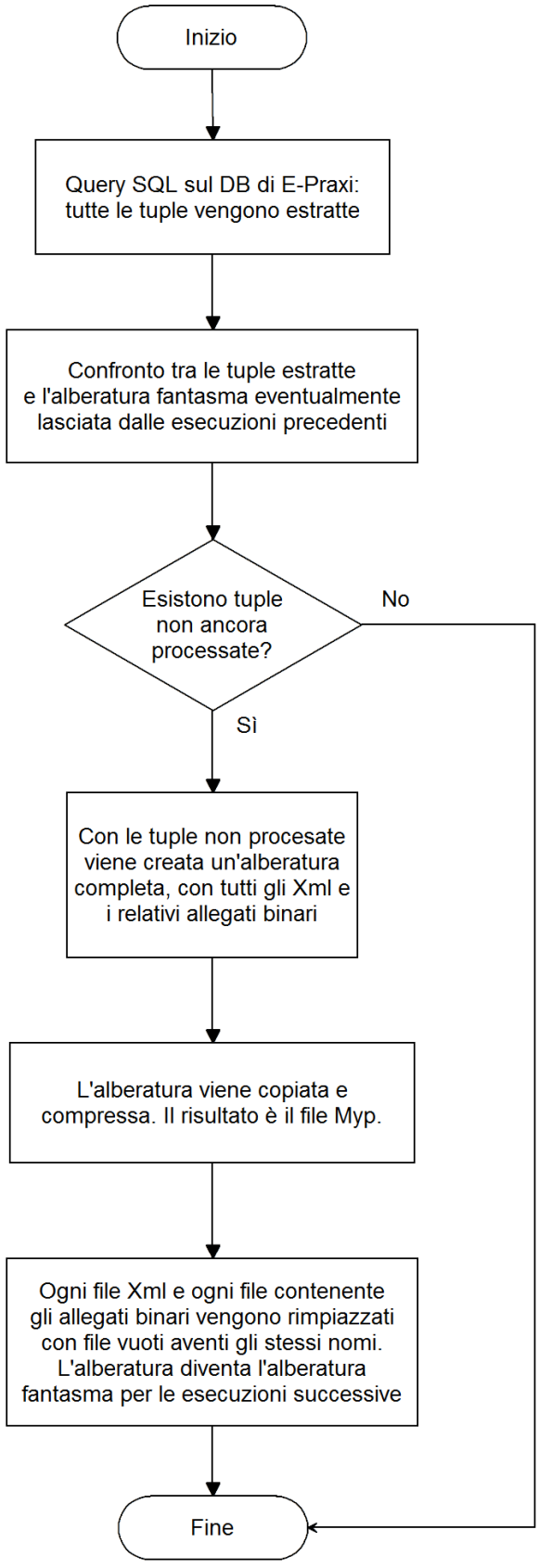
Prototipo 5

Il prototipo finale del modulo di export dati generava tutta l'alberatura di directory completa di file Xml e allegati binari come da specifiche fornite da Engineering. In coda al file batch di copia degli allegati erano stati inoltre aggiunti i comandi che:

- comprimono il risultato in un file .zip
- rinominano tale file in .myp
- eseguono la pulitura dei file temporanei

Altro problema da risolvere era rendere il tutto incrementale rispetto ai nuovi atti, prodotti in un momento successivo all'ultima esecuzione del modulo. Si voleva cioè, che ogni qualvolta si sarebbero registrati a DB nuovi atti, l'esecuzione schedulata del modulo non processasse gli atti che erano già stati elaborati in esecuzioni precedenti. Questo comportamento risulterà assolutamente necessario quando il modulo di export dati verrà eseguito nella fase di esercizio del programma e si dovranno effettuare, giornalmente, invii tramite WS dei soli nuovi atti prodotti nell'ultima giornata.
Per motivi di sicurezza, si è preferito che non venisse effettuata nessuna operazione di inserimento o modifica sui record del DB di E-Praxi, quindi fu scartata da subito la prima idea, che consisteva nell'aggiungere un campo di *flag* ai record del DB o di creare una tabella di appoggio per distinguere gli atti già processati da quelli non ancora processati.
La soluzione adottata fu invece quella di eseguire una sorta di "pulizia parziale" dei file temporanei in modo che ne rimanesse almeno una traccia per capire quali operazioni fossero già state eseguite in esecuzioni precedenti. Più precisamente, la pulizia dei file temporanei venne modificata in un'operazione di "svuotamento" dei file. I file non venivano più cancellati, ma ne rimanevano delle copie vuote, nell'alberatura di directory destinata ad essere compressa e diventare un file .myp. Il risultato di questo procedimento è un'alberatura completa di tutte le directory, i file Xml (vuoti) e gli allegati binari (vuoti), detta *alberatura fantasma*, su cui il modulo può effettuare controlli, capendo quali atti siano stati processati in esecuzioni precedenti e quindi non necessitino di essere esportati inutilmente una seconda volta.

A seguire uno schema che aiuta la comprensione dei passaggi eseguiti:



Schema 8: funzionamento del modulo di export dati (versione definitiva)

Si noti che tale approccio consente sia di rendere il processo incrementale rispetto i nuovi atti, ma anche di riprocessare eventuali atti per cui insorge la necessità di ripetere l'operazione di export. Per fare ciò è sufficiente eliminare la directory corrispondente agli atti su cui deve essere ripetuta l'operazione di export. Il modulo infatti ricontrolla ogni volta tutte le tuple del DB e va a produrre l'output (il file .myp) sempre e soltanto dove non trova corrispondenze con l'alberatura fantasma lasciate dalle eventuali esecuzioni precedenti. Semplici test empirici hanno dimostrato che tale controllo massivo non inficia le prestazioni e non risulta eccessivamente pesante per il computer che ospiterà l'ambiente di run-time.

A tale prototipo fu inoltre aggiunta una sorta di interfaccia, ovvero una pagina (sempre .cfm) interattiva in cui attraverso form e menu a tendina si potessero impostare varie scelte, come i filtri sulle date, filtri sul numero di tuple da estrarre, filtri sul numero di atto, e clausole del tipo "crea un'alberatura che contenga tutti gli allegati", oppure "crea un'alberatura che contenga l'allegato Dispositivo Atto", e da cui si potevano eventualmente forzare le operazioni di pulizia per motivi di debug.

Infine, si lavorò sulla *portabilità* del modulo. Tutti i path e le variabili comuni vennero raccolti nel file Application.cfm. Tale file è una pagina CF che viene eseguita prima (e senza soluzione di continuità) dell'esecuzione di qualsiasi altra pagina contenuta nella stessa directory. Essa può quindi contenere tutte le variabili comuni a un certo numero di pagine CF, e viene normalmente utilizzata anche per creare un *header* comune a tutte le pagine. Con la modifica di una sola variabile contenente il path assoluto della cartella del modulo di export è possibile spostare il programma su qualsiasi altra macchina, anche eventualmente montante sistema operativo diverso, purché abbia installato un server CF. Sulla base di tale variabile, i restanti path necessari al programma vengono infatti creati dinamicamente durante il run-time.

Quando il modulo fu perfezionato rimase solo la gestione della segnalazione di eventuali errori di run-time. A questo proposito fu aggiunta una parte di codice che generasse dei file di log in cui venivano registrate sia le azioni andate a buon fine sia quelle che avevano generato errori. Gli eventuali errori CF (che non dovrebbero più verificarsi, superata la fase di debug) furono gestiti in un'unica soluzione monolitica, ancora una volta grazie ai potenti mezzi messi a disposizione dal linguaggio stesso. ColdFusion mette infatti a disposizione uno strumento con il quale si può dire al server come comportarsi in caso di qualsiasi errore di run-time. La scelta fu quella di nascondere gli errori rispetto allo standard output per motivi di sicurezza, ovvero per evitare che righe di codice potessero essere eventualmente viste da un attaccante che volesse appositamente provocare un errore, e di inviare al DBA un report completo via e-mail (ad esempio dati sul browser e indirizzo IP della macchina invocante, tipo di errore, data, ora, ecc.) . Il perfezionamento della sicurezza delle pagine .cfm prodotte fu ultimato con l'inserimento nel codice di ulteriori controlli, come controlli sui tipi di dato che le pagine si aspettano in input e controlli anti SQL-injection.

5.3 – Sviluppo del modulo di invio atti tramite WS

Lo sviluppo del modulo di invio degli atti amministrativi tramite WS verso i server di MyPortal della Regione Veneto doveva avere il suo naturale punto di partenza nell'ampliamento delle funzionalità del modulo di export dati appena ultimato, o perlomeno nell'interfacciamento dell'output di quest'ultimo con un nuovo modulo che avesse il solo compito di interagire con le Porte di Dominio. La soluzione finale fu determinata vagliando soluzioni diverse, tramite anche vari incontri con i tecnici di Engineering, la cui collaborazione diventò una costante da questo punto del progetto in poi.

Step 1 – Riunione di addestramento su WS e PDD

Come punto di partenza si doveva capire quale tipo di implementazione del paradigma dei WS fosse stata scelta da parte di Engineering per MyPortal 2.5. A tal fine fu programmata una seduta formativa con Engineering ed il gruppo CReSCI (Centro Regionale Servizi di Cooperazione ed Interoperabilità), presso il complesso Vega di Regione Veneto, a cui partecipò il sottoscritto tirocinante accompagnato dal DBA di AS2. La riunione si è rivelata fondamentale per ottenere le conoscenze di base e i know-how di base per iniziare lo sviluppo di un modulo Java che potesse interagire con i WS e su come installare e configurare una piattaforma di interoperabilità di collaudo che già implementasse i concetti delle Porte di Dominio.

Dalla seduta di formazione si è evinto che:

- il **canale** di interoperabilità si sarebbe **basato su PDD**;
- lo scambio dati sarebbe avvenuto **tramite WS** comunicanti tramite tali PDD;
- i WS sarebbero stati **interfacciati alle PDD tramite Wrapping Remoto** (WS-Wrapping);
- i WS sarebbero stati del **tipo SOA**;
- lo **sviluppo del modulo invio dati** sarebbe stato **a carico dell'ente** ma comunque **su una base di partenza offerta da Engineering** da cui poi si sarebbe effettuata personalizzazione e ampliamento del codice al fine di adeguarli alle specifiche esigenze dell'ente stesso.

Contestualmente vennero forniti una copia della piattaforma Apache Tomcat (prevista nel pacchetto di interoperabilità di SIRV-Interop), adeguatamente personalizzata per poter diventare PDD per il Comune di Rovigo, e un semplice modulo Java in grado di inviare a WS una busta di tipo SOAP.
La seduta di formazione quindi continuò con un esempio pratico di come configurare semplici WS di collaudo interfacciati alla piattaforma Tomcat. Si configurarono dapprima i WS basilari per test di echo (funzionanti sia tra due enti distinti sia per un test di echo locale) e poi due semplici WS che potessero essere un buon punto di partenza per la creazione del canale dati.
Il primo dei due fu il WS di tipo RPC denominato *WebServicePortaDelegataXMLBeans*. Si tratta di un WS che si interfaccia ad una Porta Delegata tramite WS-Wrapping. Tale WS mette fondamentalmente a disposizione un metodo (entry-point) detto *chiamaPortaDelegata* tramite cui è possibile creare ed inviare una busta SOAP con anche eventuali allegati binari. Questo è un WS che sarà assolutamente fondamentale all'intero circuito di interoperabilità perché realizza di fatto quelle che saranno le Porte Delegate dei vari enti aderenti.

Il secondo WS mostrato fu un semplice WS da interfacciare con la Porta Applicativa che accettasse in ingresso una generica busta SOAP e provvedesse a salvarla su file system locale, con entry-point detto *memorizzaAllegati*.
Infine si analizzò il codice di un semplice modulo Java che prepara una Busta SOAP e la inoltra al WS di Porta Delegata. In tale busta si indicava sia come *mittente* che come *destinatario* il Comune di Rovigo stesso e *memorizzaAllegati* come WS remoto richiesto; il risultato dell'operazione è che la Porta Delegata del Comune di Rovigo invocava la Porta Applicativa del Comune di Rovigo (di fatto, un *loopback*) a cui passava una busta SOAP. Tale busta veniva quindi ricevuta dal WS *memorizzaAllegati* in ascolto sulla Porta Applicativa e memorizzata a file system dal WS stesso.

A quel punto il lavoro sarebbe proseguito presso la sede di AS2 con assistenza remota tramite comunicazioni e-mail e telefoniche con tecnici di Engineering.

Step 2 – Installazione della PDD di collaudo per il Comune di Rovigo

Con la collaborazione dei sistemisti di AS2 venne predisposto un web server che fungesse da PDD di collaudo per il Comune di Rovigo. Si trattava di una macchina Linux su cui venne copiata ed avviata la copia di Apache Tomcat che era stata fornita durante la seduta formativa. Tale piattaforma espone un web server che si mette in ascolto sulla porta TCP 8880, che deve essere resa disponibile per essere raggiunta da Internet.
Tramite interfaccia web grafica, oppure tramite l'editing manuale di appositi file .xml è possibile configurare, tra le altre cose, i servizi di Porta Applicativa che si intende mettere a disposizione delle altre organizzazioni e una lista di Servizi Accessibili che le altre organizzazioni hanno già reso disponibili per la fruizione.
Il progetto SIRV-Interop prevede che vi sia una unica lista di Servizi Accessibili, online e consultabile da tutte le organizzazioni aderenti al circuito di interoperabilità. Su tale lista ogni organizzazione deve quindi indicare i servizi esposti, i relativi URL ed entry-point, e una descrizione del servizio tramite WSDL. Allo stato attuale (Giugno 2011) la creazione di tale indice comune non è ancora nemmeno incominciata e si configura invece per ogni PDD di ogni ente un indice locale.
Quando la Porta Delegata quindi deve invocare un WS remoto consulta il proprio indice locale, da cui desume URL ed entry-point da utilizzare e inizializza di conseguenza la comunicazione con la Porta Applicativa remota relativa a al WS stesso.
Il primo servizio ad essere stato configurato per il Comune di Rovigo è stato il servizio di echo, sia per il lato Porta Applicativa che per il lato Porta Delegata.

Lato Porta Applicativa

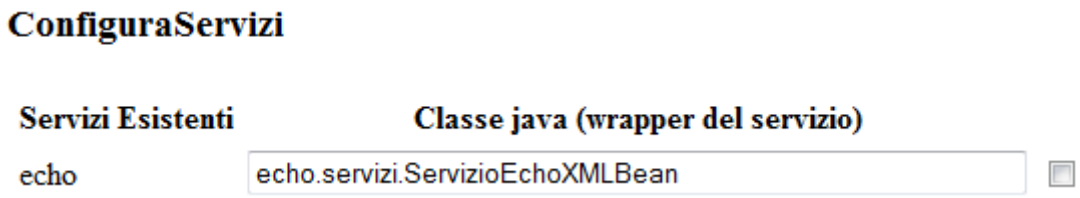


Figura 4: configurazione WS di echo (lato Porta Applicativa)

Lato Porta Delegata

Amministrazione	Tipo	Indirizzo telematico	Del?
C_H620	SPC	http://localhost:8880/portadidominio/servlet/spasaaj	
Servizi Implementati	**Tipo**	**Profilo Collaborazione**	
echo	SPC	EGOV_IT_ServizioSincrono	
MYPORTAL	SPC	http://pdd-interop.collaudo.regione.veneto.it/portadidominio/serv	
Servizi Implementati	**Tipo**	**Profilo Collaborazione**	
echo	SPC	EGOV_IT_ServizioSincrono	

Figura 5: configurazione WS di echo (lato Porta Delegata)
Si noti che C_H620 è il codice IPA del Comune Di Rovigo stesso

Il servizio di echo così configurato ha permesso di effettuare dei test in entrambe le direzioni (PD Comune di Rovigo verso PA di MyPortal e viceversa) e anche del tipo loopback. Per invocare tali WS di prova venne usata un'interfaccia grafica già compresa nella piattaforma Apache fornita:

Figura 6: interfaccia grafica per il collaudo delle PDD
compresa nella piattaforma Apache distribuita da CReSCI

Il WS di echo spedisce alla PA remota una busta SOAP e termina con un OK solo se entro un dato timeout riceve indietro dalla stessa PA la stessa busta SOAP inizialmente inviata. In caso contrario termina con un KO (ad esempio se scade il timeout), oppure con eventuali eccezioni sollevate dalla PA remota (ad esempio, se il servizio echo non è ancora stato attivato e/o correttamente configurato ed implementato).
I test fornirono da subito risultati soddisfacenti e si iniziò la configurazione dei servizi per l'invio degli atti amministrativi e lo sviluppo del relativo modulo applicativo invocante.

Step 3 – Sviluppo e collaudo del modulo applicativo invocante un WS di test

Il terzo step è stato il più lungo e laborioso. E' stato infatti quello in cui si è iniziata effettivamente la stesura del codice Java di un modulo client in grado di preparare buste SOAP adatte ad un WS che riceva e memorizzi gli atti amministrativi. Arrivare a tale risultato comportava però molte configurazioni di sistema, stesura e debug di codice sia Java che ColdFusion (per correzioni ad-hoc del modulo di export al fine di interfacciarlo il meglio possibile con quello di invio), e l'acquisizione nei vari passaggi di molti know-how precedentemente non in possesso di AS2. Per iniziare a comprendere la logica di programmazione che si sarebbe dovuta utilizzare si iniziò personalizzando del codice precedentemente fornito da Engineering, in modo da creare un mini-modulo che inviasse una busta SOAP ad un WS lato Porta Applicativa detto *memorizzaAllegati*. Tale WS fu configurato sul lato PA della PDD del Comune di Rovigo, in modo da effettuare dei test di tipo loopback tra la PD e la PA dell'Ente stesso. Il WS resta in ascolto ed attende delle buste SOAP e, se correttamente formate, ne memorizza gli allegati su file system.

Il mini-modulo Java fornito da Engineering serviva soltanto alla creazione di una generica busta SOAP a partire da una variabile di tipo String che fungeva da unico allegato.
Il punto saliente di tale classe Java era il metodo che portava la seguente firma:

chiamaPortaDelegata(String url, String metodo, String intestazione, String allegato)

Si noti che questo è un metodo Java, e non è il WS *chiamaPortaDelegata* precedentemente citato che porta lo stesso identico nome. I valori che questo metodo attende sono:

- **url:** l'URL del WS di Porta Delegata. Nel nostro caso "http://127.0.0.1:8880/portadidominio/services/WebServicePortaDelegataXMLBeans" 127.0.0.1 se si lavora sullo stesso server di PD, altrimenti l'indirizzo assoluto di rete LAN del server di PD;
- **metodo:** entry-point del WS di Porta Delegata. Nel nostro caso "chiamaPortaDelegata". (N.B.: anche in questo caso non bisogna confondersi col metodo Java omonimo);
- **intestazione:** una stringa in formato Xml valido che diventi l'intestazione della busta SOAP. Un esempio di intestazione valida usata nei test è:
  "<Intestazione>
  <Mittente tipo="SPC">C_H620</Mittente>
  <Destinatario tipo="SPC">C_H620</Destinatario>
  <OraRegistrazione>2011:05:02_10:45</OraRegistrazione>
  <Servizio tipo="SPC">memorizzaAllegati</Servizio>
  <Azione>Richiesta</Azione>
  <ProfiloCollaborazione tipo="SPC">EGOV_IT_ServizioSincrono</ProfiloCollaborazione>
  <AllegatoIN>RICHIESTA</AllegatoIN>
  <AllegatoOUT>RISPOSTA</AllegatoOUT>
  </Intestazione>"
  (Dove C_H620 è il Codice Ente del Comune Di Rovigo stesso)

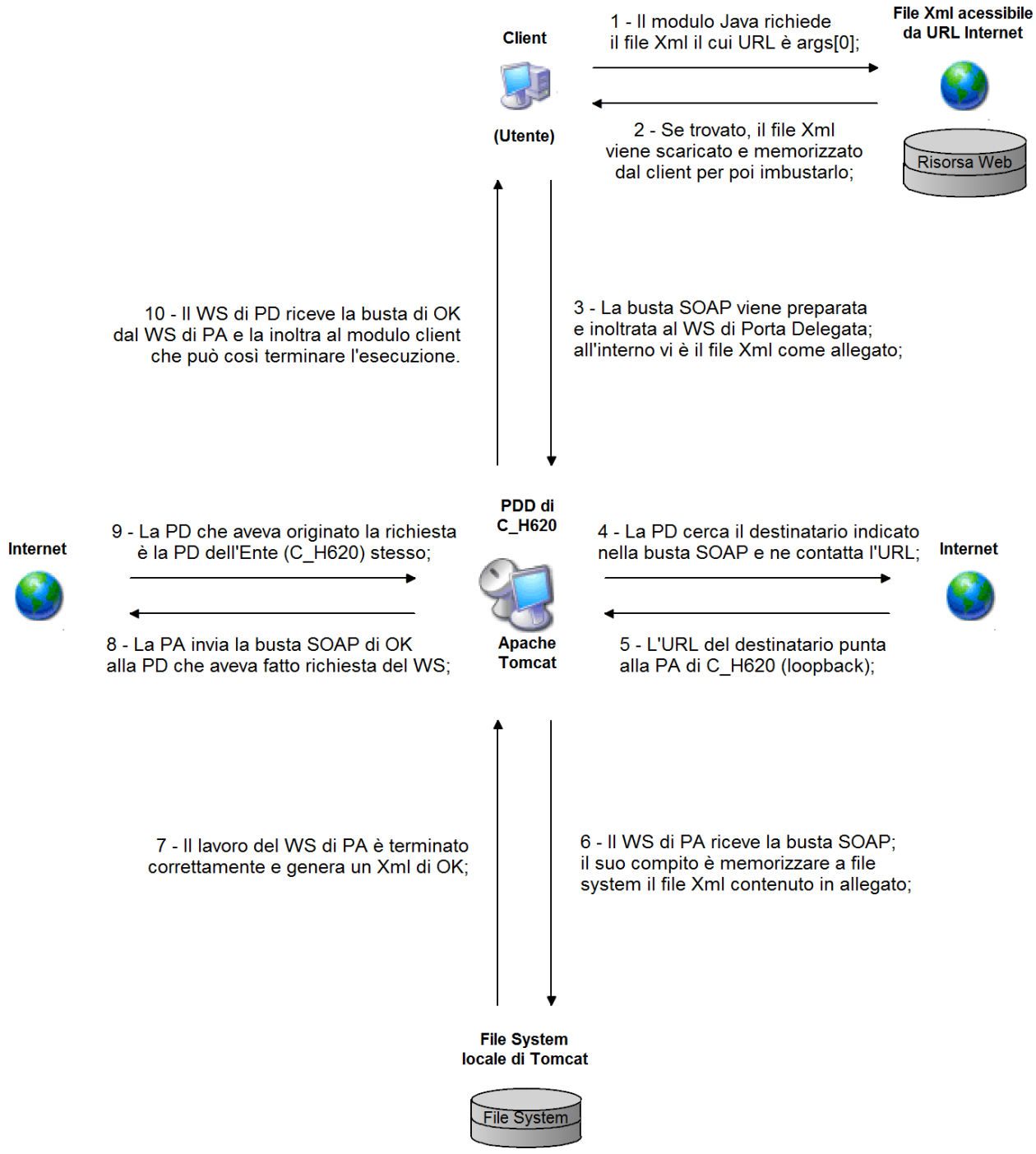
- **allegato:** una stringa generica. Può essere ad esempio un anche il contenuto di un file Xml, ma i caratteri in essa contenuti non hanno alcuna rilevanza. Tale stringa verrà semplicemente memorizzata a file system in un file di testo dal WS *memorizzaAllegati*.

Quindi si personalizzò il codice Java in modo che la classe potesse accettare come parametro di input l'URL di un generico file Xml. Fu quindi adattato il metodo *main* affinché attendesse su *args*[0] un URL web. Da tale URL, tramite la classe *BufferedReader*, venivano lette le varie righe di testo che componevano il file Xml remoto, per poi essere concatenate e memorizzate nella variabile *allegato* di tipo String. L'invocazione del metodo *chiamaPortaDelegata*, avviava la creazione della busta SOAP da passare al WS di Porta Delegata. La creazione di tali buste avvenivano invocando altri metodi di classi provenienti da pacchetti come *org.apache.commons*, *org.apache.axis* e *javax.xml.rpc*.
A quel punto il lavoro del modulo Java è sostanzialmente terminato. Esso resta solo in attesa di una risposta da parte della Porta Delegata del tipo "OK", "KO" oppure "ECCEZIONE". La PD invece, una volta ricevuta la richiesta di WS da parte del modulo Java, consulta il proprio indice locale e cerca l'ente <Destinatario>. La ricerca di C_H620 dà come risultato la Porta Applicativa del Comune di Rovigo (di fatto, un loopback) e chiede all'URL indicato nell'indice la fruizione del WS indicato nel tag <Servizio> (ovvero *memorizzaAllegati*). Tale servizio richiede un allegato di tipo IN e fornisce un allegato di tipo OUT (un file Xml che può indicare una risposta del tipo "OK", "KO" oppure "ECCEZIONE"). La PA invocata concede l'utilizzo del WS, in quanto l'ente del campo <Mittente> risulta attendibile secondo una lista stilata in precedenza. Il WS "*fa quello che deve fare*"*, ovvero memorizza gli su file system la stringa che riceve come allegato IN e notifica alla PA il completamento dell'operazione senza errori. La PA a questo punto prepara il già citato allegato OUT con l'Xml che sostanzialmente coincide con una risposta del tipo "OK", e lo invia alla PD che aveva avviato la comunicazione sul canale dati. La PD infine inoltra la risposta al modulo Java che scrive sulla standard input l'esito finale della comunicazione.

*l'espressione "*fa quello che deve fare*", per quanto inelegante, è decisamente espressiva riguardo quella che è la vera natura dei WS. I WS sono infatti strumenti che permettono di disaccoppiare talmente bene gli elaboratori e addirittura i vari moduli all'interno di uno stesso elaboratore, che una volta fornito ad un WS il corretto input, è davvero inutile sapere "che cosa" stia davvero "facendo" il modulo che realizza di fatto il WS e ne origina l'output. L'importante, soprattutto quando i WS sono utilizzati in una SOA, i messaggi scambiati tra di essi siano "ben formati".

Si noti, che con questo tipo di architettura (del tipo SOA), l'elaboratore invocante il WS di Porta Delegata può risiedere in un qualunque punto della LAN di AS2, la risorsa puntata dall'URL è un qualsiasi file Xml memorizzato su un qualsiasi calcolatore connesso ad Internet, e la natura del canale dati tra PD e PA è del tutto ininfluente dal punto di vista del modulo Java. Il disaccoppiamento risulta quindi davvero notevole.

A seguire uno schema riassuntivo che aiuta la comprensione dei vari passaggi:



Schema 9: collaudo di WS e PDD

Step 4 – Sviluppo e collaudo del modulo applicativo invocante i WS di MyPortal

Il quarto step fu relativamente semplice, soprattutto in virtù dell'esperienza e dei risultati ottenuti nello step precedente.

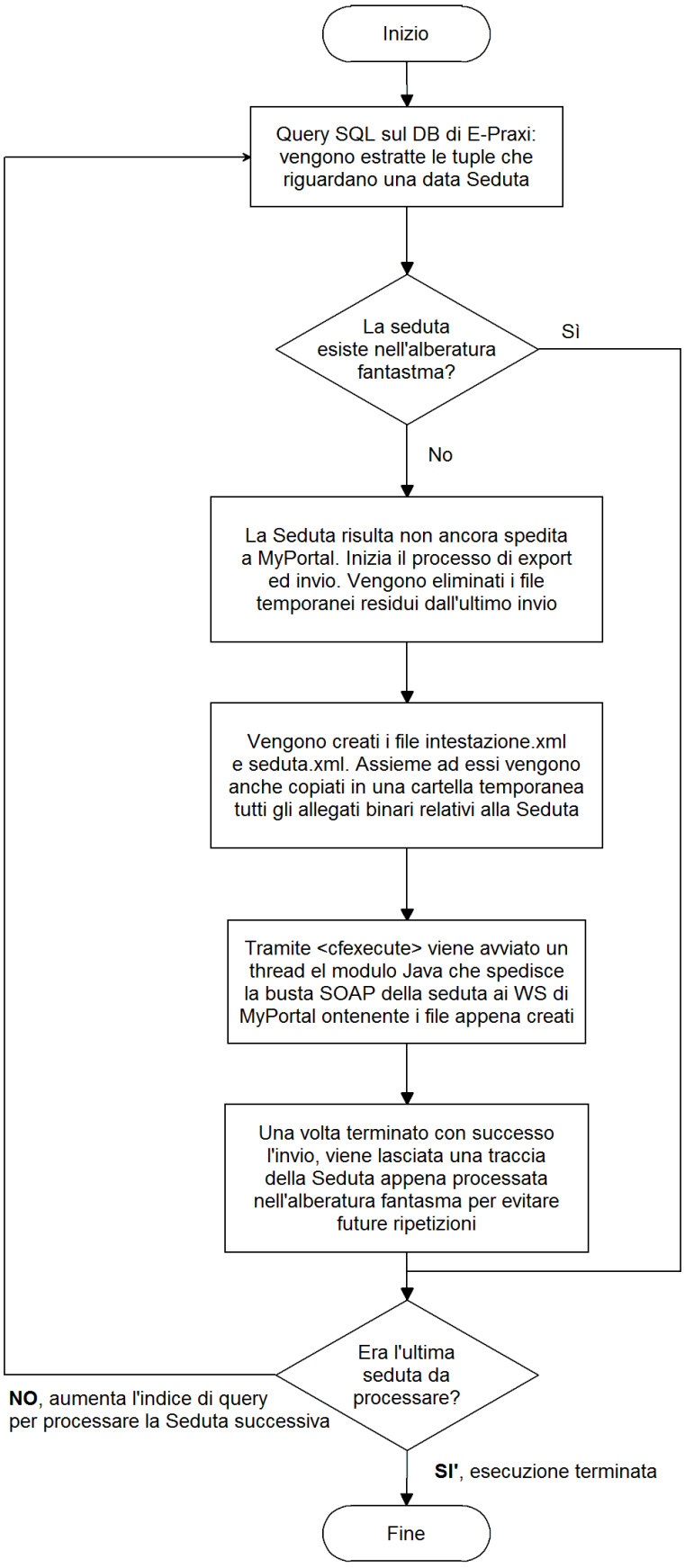
Dapprima di configurarono i tre WS messi a disposizione da MyPortal che sono di interesse al presente progetto sulla piattaforma Tomcat. Sul menù Lista Servizi Accessibili, secondo specifiche fornite da Engineering, si aggiunsero i servizi *InvioDelibereGiunta*, *InvioDelibereConsiglio*, e *InvioDetermine*.

N.B.: Per semplicità, essendo i tre casi distinti ma davvero troppo simili per discuterne senza risultare ridondanti, da ora in poi ci si riferirà soltanto al caso dell'invio di **delibere di giunta**.

Le specifiche sui dati fornite da Engineering che descrivevano il formato dei file Xml che Alfresco richiede come input dei WS con cui è interfacciato differivano veramente di poco rispetto a quelle relative all'importer offline. Si notano alcune semplificazioni, ad esempio riguardanti i livelli di nidificazione delle sotto-directory, e l'aggiunta di alcuni campi, come ad esempio quelli riguardanti la pubblicazione degli atti nell'Albo Pretorio online.

Al fine di avere un modulo ColdFusion in grado di generare file Xml corretti si iniziò a riadattare il codice precedentemente scritto per il modulo di export. Il codice venne in realtà riscritto ed ampliato in modo di poter soddisfare anche nuove esigenze legate all'invio online degli atti estratti. Ad esempio, era necessario che le operazioni di estrazione ed invio fossero atomiche per ogni singola *seduta* (le sedute di giunta corrispondenti alle varie delibere). Precedentemente invece, ogni tupla estratta generava una parte dell'output che andava poi a comporre il risultato finale, che poteva anche essere imperfetto o incompleto. Riguardo i WS furono invece aggiunti controlli più restrittivi che impediscono il caricamento di una delibera se essa non viene caricata con la seduta per intero in un singolo invio, comprensiva di tutti ogni ODG (ordine del giorno) e di tutti i file binari in allegato, oppure se uno dei suoi campi è malformato o esprime un valore non valido.

Il risultato fu un modulo ColdFusion di export dati che funziona secondo il seguente schema:



Schema 10: funzionamento modulo ColdFusion di invio a WS

Ora bisogna spiegare meglio cosa succede quando il codice ColdFusion incontra il comando <cfexecute> ed avvia il modulo Java. Il comando <cfexecute> permette ad un server ColdFusion che sia stato installato in ambiente Windows, di lanciare un comando MS-DOS arbitrario con la possibilità di specificare anche argomenti a piacimento. Nel nostro caso vogliamo lanciare un comando che faccia partire il modulo Java di invio a WS della Seduta corrente, ovvero un comando del tipo:

C:\>java –jar PDDAttiClient.jar [argomenti]

Dove [argomenti] saranno campi testuali separati da un carattere vuoto che andranno a comporre il vettore args[] del metodo main della classe *PDDAttiClient*. Tale classe è stata sviluppata sulla falsariga di quella sviluppata per i test dello step precedente. La macchina ospitante il server ColdFusion però non aveva installato nessun ambiente JRE (il run-time di Java) e non era nemmeno desiderabile installarne uno, al fine di salvare tutta la RAM possibile di una macchina assolutamente cruciale per i servizi rivolti al pubblico. Si scelse quindi di avvalersi di un JRE che non richiede installazione, detto jPortable. Esso è un JRE completo, che non necessita di installazione e che, una volta copiato a file system, permette di avviare il tradizionale "java.exe".
Ora passiamo ad analizzare il funzionamento della versione finale del modulo Java. Tale modulo consta di una singola classe Java il cui metodo main attende dei parametri attraverso il vettore args[]. Il comando corretto da lanciare è il formato secondo la seguente sintassi:

C:\>java –jar PDDAttiClient.jar UrlWsPortaDelegata Metodo intestazione.xml seduta.xml allegato1 allegato2 ... allegatoN

Dove:

- **UrlWsPortaDelegata:** l'URL del WS di Porta Delegata. Nel nostro caso "http://127.0.0.1:8880/portadidominio/services/WebServicePortaDelegataXMLBeans" 127.0.0.1 se si lavora sullo stesso server di PD, altrimenti l'indirizzo assoluto di rete LAN del server di PD;
- **Metodo:** entry-point del WS di Porta Delegata. Nel nostro caso "chiamaPortaDelegata". (N.B.: non si tratta del metodo Java omonimo);
- **intestazione.xml:** percorso assoluto di un file in formato Xml valido memorizzato a file system. Esso è generato dinamicamente dal modulo di export dati ColdFusion e diventerà l'intestazione della busta SOAP che viene processata dal WS di Porta Delegata;
- **seduta.xml:** percorso assoluto di un file in formato Xml valido memorizzato a file system. Esso sarà il primo allegato della busta SOAP, e verrà processato dal WS di import dati di MyPortal. E' il descrittore di una generica seduta relativa ad una delibera di giunta, e viene generato dal modulo ColdFusion di export dati;
- **allegato1 allegato2 ... allegatoN:** URL web, separati da spazio, di allegati binari relativi alla Seduta che si sta processando.

E' compito del modulo ColdFusion creare dinamicamente il comando corretto, formato secondo la sintassi appena descritta, e lanciarlo tramite <cfexecute>.

Ora vediamo un esempio di entrambi i file Xml che i WS coinvolti nella comunicazione attendono come input:

intestazione.xml:

```
<Intestazione>
        <Mittente tipo="SPC">C_H620</Mittente>
        <Destinatario tipo="SPC">MYPORTAL</Destinatario>
        <OraRegistrazione>2011:06:14_11:58</OraRegistrazione>
        <Servizio tipo="SPC">InvioDelibereGiunta</Servizio>
        <Azione>Richiesta</Azione>
        <ProfiloCollaborazione tipo="SPC">EGOV_IT_ServizioSincrono</ProfiloCollaborazione>
        <Allegati>
                <Allegato tipo="IN" mimeType="txt/xml">seduta.xml</Allegato>
                <Allegato tipo="IN" mimeType="application/octet-stream">
                        20110223123512884121223112 7_3.DOC</Allegato>
                <Allegato tipo="IN" mimeType="application/octet-stream">
                        20110223123132650203423012_3.DOC</Allegato>
                <Allegato tipo="IN" mimeType="application/octet-stream">
                        20110217165534005144791315 6_1.DOCX</Allegato>
                <Allegato tipo="IN" mimeType="application/octet-stream">
                        20110223122410728320809411_3.DOC</Allegato>
                <Allegato tipo="IN" mimeType="application/octet-stream">
                        20110223122606572547868405_4.DOC</Allegato>
                <Allegato tipo="OUT" mimeType="txt/xml">RISPOSTA</Allegato>
        </Allegati>
</Intestazione>
```

seduta.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<seduta>
   <name>Seduta_2011-02-22</name>
   <title/>
   <description/>
   <author/>
   <edit-inline/>
   <sede>Giunta</sede>
   <data-ora>22/02/2011 16:00</data-ora>
   <data-pubblicazione/>
   <data-da/>
   <data-a/>
   <ordiniDelGiorno>
      <odg>
         <name>14064_Delibera_n_19</name>
         <title/>
         <description/>
         <author/>
         <edit-inline/>
         <numero>14064</numero>
```

```
    <argomento><![CDATA[Conferma tariffe della tassa occupazione spazi ed aree pubbliche e
            dell'imposta di pubblicità/diritti sulle pubbliche affissioni per l'anno 2011]]></argomento>
    <numero-delibera>19</numero-delibera>
    <documenti-allegati>
    <allegato>20110223123512884121223 1127_3.DOC</allegato>
    </documenti-allegati>
    <data-pubblicazione/>
    <data-da/>
    <data-a/>
</odg>
<odg>
    <name>14038_Delibera_n_18</name>
    <title/>
    <description/>
    <author/>
    <edit-inline/>
    <numero>14038</numero>
    <argomento><![CDATA[Donazione quadro di Carlo Campi al Comune di Rovigo]]></argomento>
    <numero-delibera>18</numero-delibera>
    <documenti-allegati>
    <allegato>20110223123132650203423012_3.DOC</allegato>
    <allegato>20110217165534005144791315 6_1.DOCX</allegato>
    </documenti-allegati>
    <data-pubblicazione/>
    <data-da/>
    <data-a/>
</odg>
<odg>
    <name>13959_Delibera_n_17</name>
    <title/>
    <description/>
    <author/>
    <edit-inline/>
    <numero>13959</numero>
    <argomento><![CDATA[Concessione in comodato all' Associazione "Pro Loco Città di Rovigo
            Centro" locali di proprietà comunale presso il Castello di Rovigo .]]></argomento>
    <numero-delibera>17</numero-delibera>
    <documenti-allegati>
    <allegato>20110223122410728320809411_3.DOC</allegato>
    </documenti-allegati>
    <data-pubblicazione/>
    <data-da/>
    <data-a/>
</odg>
<odg>
    <name>13997_Delibera_n_16</name>
    <title/>
    <description/>
    <author/>
    <edit-inline/>
    <numero>13997</numero>
```

```
        <argomento><![CDATA[Verifica regolare tenuta schedario elettorale
                informatizzato]]></argomento>
        <numero-delibera>16</numero-delibera>
        <documenti-allegati>
        <allegato>20110223122606572547868405_4.DOC</allegato>
        </documenti-allegati>
        <data-pubblicazione/>
        <data-da/>
        <data-a/>
      </odg>
   </ordiniDelGiorno>
   <pubblicaInAlbo>true</pubblicaInAlbo>
   <pubblicaInAlboInBozza>false</pubblicaInAlboInBozza>
   <dataInizioPubblicazione>25/02/2011</dataInizioPubblicazione>
   <durataPubblicazione>14</durataPubblicazione>
</seduta>
```

Il punto chiave della classe Java sviluppata in quest'ultimo step funziona in un modo decisamente diverso rispetto a quella precedente, ma solo da un punto di vista tecnico. Da un punto di vista concettuale invece è decisamente simile. Per spiegare meglio cosa questo voglia dire, analizziamo il metodo *chiamaPortaDelegata*, che è stato parzialmente riscritto e che ora porta la seguente firma:

chiamaPortaDelegata(String[] args)

Ancora una volta, si noti che questo è un metodo Java, e non è il WS *chiamaPortaDelegata* omonimo. Si noti inoltre che il vettore *args* del presente metodo non è il vettore *args* del metodo *main*. I valori attesi dal presente metodo sono:

- **args[0]:** l'URL del WS di Porta Delegata. Nel nostro caso "http://127.0.0.1:8880/portadidominio/services/WebServicePortaDelegataXMLBeans" 127.0.0.1 se si lavora sullo stesso server di PD, altrimenti l'indirizzo assoluto di rete LAN del server di PD;
- **args[1]:** entry-point del WS di Porta Delegata. Nel nostro caso "chiamaPortaDelegata". (N.B.: anche in questo caso non bisogna confondersi col metodo Java omonimo);
- **args[2]:** una stringa che rappresenti il percorso assoluto a file system di un file in formato Xml valido che diventi l'intestazione della busta SOAP (è il file intestazione.xml);
- **args[3]:** una stringa che rappresenti il percorso assoluto a file system di un file in formato Xml valido che diventi il primo allegato della busta SOAP (è il file seduta.xml);
- **args[4], args[5], … , args[n]:** URL web di uno o più file binari da aggiungere come allegati alla busta SOAP. Sono i file binari relativi ad una data Seduta che devono essere pubblicati su MyPortal. Il modo Java si occupa di scaricarli e crearne una copia temporanea a file system locale per poterli imbustare.

Il metodo quindi si occupa di creare effettuare la chiamata a WS della Porta Delegata. Si veda il seguente codice:

```
Call call = (Call) service.createCall();
call.setSOAPActionURI(args[1]);
call.setOperationName(args[1]);
call.setTargetEndpointAddress(args[0]);
```

con cui viene creata la busta SOAP da inviare. Gli allegati della busta sono invece contenuti nel vettore:

```
Object[] input = new Object[args.length - 2];
```

la cui lunghezza è pari al numero di allegati della busta SOAP. La chiamata a WS vera e propria viene effettuata attraverso la sintassi:

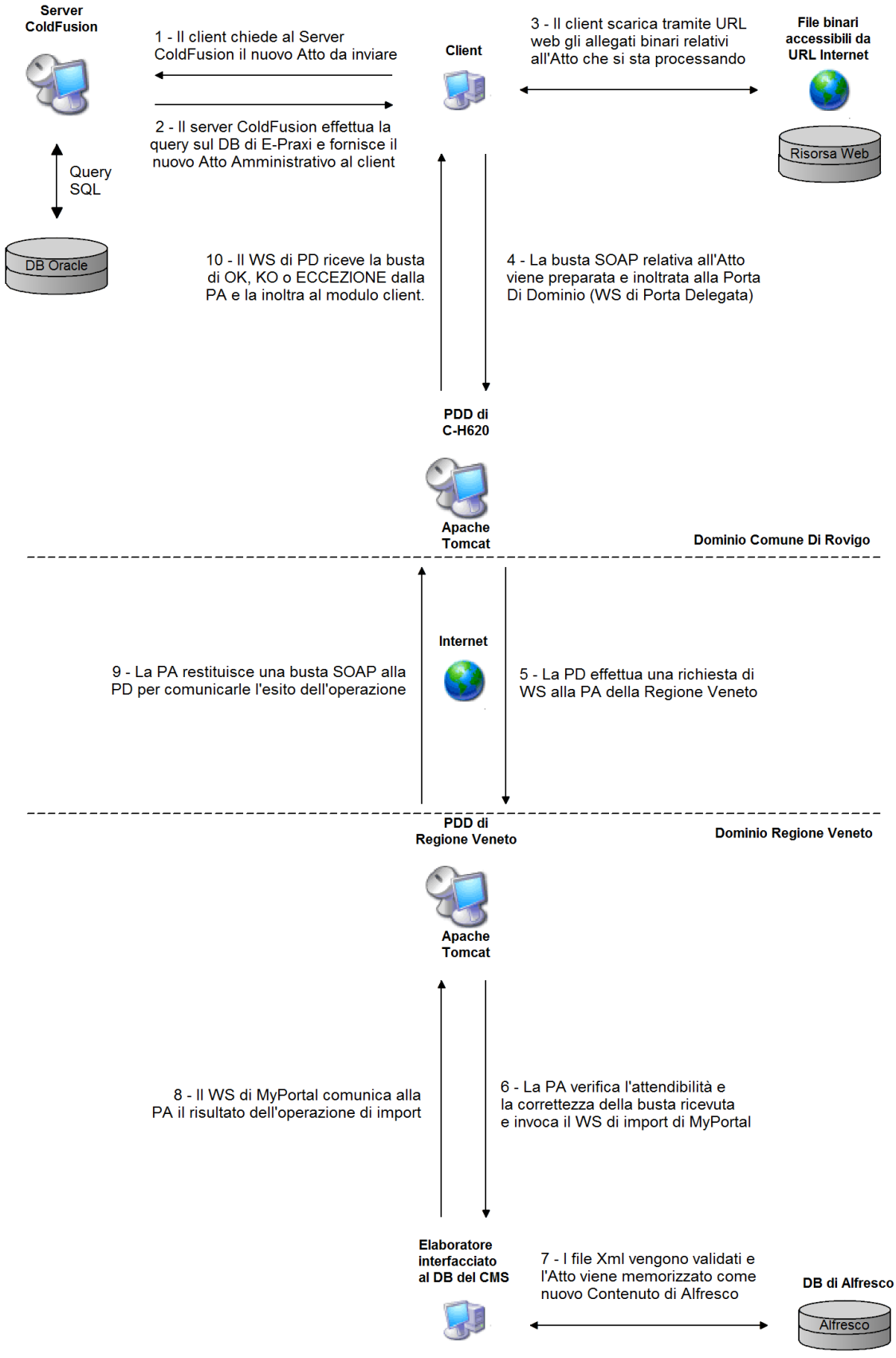
```
Object[] result = (Object[]) call.invoke(new Object[] {input});
```

che restituisce come risultato un vettore di Object rappresentante la busta SOAP di risposta. Essa sarà una busta composta della sola intestazione, che altro non è che un file Xml descrittivo per uno dei tre eventi OK, KO o ECCEZIONE. La risposta viene estratta e processata nel seguente modo:

```
byte[] intestazione = (byte[]) result[0];
System.out.println("Intestazione Ricevuta :\n" + new String(intestazione));
```

Il modulo ColdFusion, in accoppiata con il modulo Java appena descritto, compongono quindi il lato *client*, o per meglio dire, lato *richiedente* di un'architettura di tipo SOA. L'unico limite architetturale è legato alle regole aziendali di AS2. L'elaboratore da cui parte la richiesta di servizio deve cioè essere un elaboratore della rete aziendale (LAN) di AS2, ma potrebbe in realtà essere un qualsiasi dispositivo connesso ad Internet e che sappia formulare normali richieste Http. Tale macchina altro non fa che effettuare una richiesta di una pagina web .cfm ad un dato web server (il web server di ColdFusion) che effettua una richiesta Http di Web Service al web server di Porta Delegata (il web server della piattaforma Tomcat). La Porta di Dominio dell'ente (in funzione di PD) contatta la PDD di Regione Veneto (che quindi funge da Porta Applicativa) e inoltra la richiesta di servizio originata dal modulo Java. Se viene accettata la richiesta di servizio, il WS viene "erogato". Si noti che il termine "erogato" è leggermente ambiguo nel nostro caso, in quanto avviene l'erogazione di un servizio che memorizza un input fornito dalla Porta Delegata. L'unica cosa davvero "erogata" è la conferma di avvenuta memorizzazione. La conferma viene ricevuta dalla PD ed inoltrata al modulo Java. Il cerchio si chiude con la visualizzazione a monitor di avvenuto trasferimento o meno degli atti che si è tentato di inviare. Si noti che il vero e proprio canale dati tra i due anti altro non è che Internet, attraverso cui vengono scambiate delle buste SOAP. Tali buste giocano il ruolo fondamentale nella richiesta/erogazione dei Web Service coinvolti, sia nel canale dati tra di due enti, sia nelle comunicazioni intradominio tra gli elaboratori.

A seguire uno schema che riassume ed aiuta a comprendere meglio i passaggi sopra descritti:



Schema 11: invio atti amministrativi tramite WS (topografia)

6 – Risultati

Ricordiamo che il risultato voluto per cui si è avviato il presente progetto è la pubblicazione online degli atti amministrativi del Comune di Rovigo sul portale di nuova generazione MyPortal 2.5, installato su sistemi della Regione Veneto.
La visualizzazione di tali atti nell'apposita pagina web del portale è quindi lo scopo e anche la misura della riuscita del presente progetto. L'unica altra richiesta vincolante da parte del committente è che tale pubblicazione funzionasse poi in maniera incrementale e periodica, con periodicità di un invio al giorno di tutti gli atti non ancora pubblicati. Il committente non ha effettuato altri tipi di richieste e non ha imposto altro tipo di vincoli. Naturalmente buon senso vuole che si dovesse sviluppare un applicativo che avesse prestazioni e occupazione di risorse ragionevoli.

Modulo di export dati

Il modulo di export dati, il cui sviluppo è descritto nel paragrafo 4.2, ha richiesto circa due mesi di lavoro (5 ore al giorno per 5 giorni la settimana) dall'acquisizione delle competenze necessarie fino ad arrivare alla validazione dell'output generato. Dapprima si sono acquisite le conoscenze di base sul linguaggio ColdFusion e sulla formulazione di query SQL verso DB Oracle. Successivamente si è proceduti allo sviluppo, ovvero alla stesura del codice, e quindi ai test. Il modulo di export fa operazioni *una tantum*, e si può parlare di fase di *funzionamento* del ciclo di vita di questo applicativo solo nella sua versione riadattata ai Web Service. La prima versione, tutt'ora rilasciata e in uso al Comune di Rovigo, servirà per il primo popolamento della versione definitiva di MyPortal 2.5 e per eventuali funzioni di export future per cui i WS risultino inadeguati (si pensi ad esempio ad un atto che ha per allegato un file binario di dimensioni eccezionalmente grandi). Come già detto, il modulo è quasi interamente scritto in ColdFusion ed è dotato di una semplice ma efficace interfaccia grafica (pagina web interattiva). Vi sono sottomoduli comuni a tutto l'applicativo e sottomoduli specifici per ogni tipo di atto che si deve processare. In tutto sono state scritte poco più di 1.300 righe di codice ColdFusion e circa 110 righe di codice batch del sottomodulo di supporto (che si occupa della copia degli allegati e della compressione in formato Myp). L'estrazione massiva di tutti gli atti di un certo tipo (ad esempio, l'estrazione dal DB di tutte le determinazioni) comportava un'occupazione di risorse del server ColdFusion accettabile. Anche il tempo necessario risultava accettabile, essendo di sull'ordine di grandezza del centinaio di minuti nel caso di estrazione massiva (si ricordi che l'export massivo di tutto il DB avviene davvero raramente e solo in casi eccezionali) mentre sull'ordine di grandezza di qualche decina di secondi per gruppi di 5 atti amministrativi. L'unico effetto collaterale deriva dalla copia tramite rete Ethernet dei file allegati di ciascun atto, che determinava un'occupazione di banda della rete aziendale di AS2 rilevante, ma comunque sopportabile. La schedulazione in orari notturni del modulo può aggirare anche questo inconveniente.

Engineering dichiarò valido l'output prodotto e confermò che i contenuti potevano essere caricati correttamente ad Alfresco verso la fine di Gennaio 2011. In attesa di progressi riguardo il versante Web Service, a partire da Febbraio 2011 si avviarono test per verificare che il modulo di export potesse lavorare effettivamente in maniera incrementale, per quando sarebbe stato usato nel processo giornaliero di invio nuovi atti. I risultati furono davvero soddisfacenti, e si constatò che

effettivamente la schedulazione quotidiana del modulo di export generava un nuovo file .myp solo in quei giorni in cui erano stati prodotti nuovi atti. Gli atti già processati non venivano esportati inutilmente una seconda volta e a file system restava anche un log dettagliato sulle operazioni che venivano eseguite ad ogni funzionamento. L'unico inconveniente si registrò per un particolare atto che presentava un'incoerenza tra record a DB e corrispondente entità reale: uno dei record su cui era memorizzato l'atto conteneva un percorso di rete relativo ad un allegato che era errato. Quando il modulo cercava quindi di copiare il file non ci riusciva e il file .myp generato era incompleto, ovvero era privo di quel file. Durante una verifica si volle confrontare il numero cumulativo di tutti gli allegati presenti in tutti i file .myp generati e il numero di allegati che erano riferiti da record del DB Oracle. Questi ultimi risultavano maggiori di una unità e si cominciò quindi a cercare di capire cosa fosse andato storto. Fortunatamente ci si accorse subito dai log registrati dall'applicativo che uno dei comandi di copia degli allegati non era andato a buon fine. La discrepanza era dovuta banalmente all'errata memorizzazione a DB dell'estensione del file allegato: a DB era stato registrato come file .doc quando era in realtà un file .docx.

Modulo di invio a Web Service

Il modulo di invio a Web Service è una rivisitazione ed estensione del modulo precedente, più un modulo Java (e non più batch) di appoggio. Lo sviluppo del modulo Java fu leggermente impegnativo per la natura stessa del codice Java (molto meno flessibile di ColdFusion) e per l'acquisizione di competenze riguardo i WS, che rappresentano un campo decisamente più vasto rispetto al semplice formulazione di query SQL. Tuttavia, le righe di codice Java scritte sono molte meno rispetto al modulo ColdFusion, ovvero poco più di 400. La differenza sostanziale in questa seconda parte del lavoro è stata invece il passaggio da un lavoro puramente individuale (programmatore a tempo pieno) ad uno più orientato al lavoro di gruppo. La stesura del codice Java, la configurazione della piattaforma Tomcat, e l'acquisizione di competenze riguardo i paradigmi dei WS sono state effettuate tramite riunioni, incontri individuali e numerose telefonate con tecnici di Engineering e del gruppo CReSCI. Il lavoro di squadra è stato fondamentale nelle parti di debug e assolutamente necessario riguardo alcuni *troubleshooting* relativi alle Porte di Dominio. Purtroppo a tutto ciò si sono aggiunti continui e gravi ritardi nell'avanzamento del progetto SIRV-Interop e MyPortal 2.5, e la realizzazione del canale dati è finita per durare alcuni mesi. I lavori partirono nel Marzo 2011 e sono a tutt'ora (Luglio 2011) in fase di completamento. Il progetto si può comunque considerare concluso (essendo mancanti solo alcune rifiniture all'interfaccia e al sistema di logging) soprattutto in virtù dei risultati positivi degli ultimi desti di invio.

La versione definitiva del modulo di invio (o meglio, di export più invio) degli atti amministrativi è stata comunque rilasciata ed è tutt'ora in uso dal Comune di Rovigo. Bisogna tuttavia ancora parlare di fase di *collaudo* e non di *funzionamento* del ciclo di vita dell'applicativo. La maggior parte dei test hanno dato esito positivo, ovvero gli atti amministrativi che si sceglie di processare compaiono effettivamente sulla pagina apposita di MyPortal (lato front-end) e dall'interfaccia di Alfresco (lato back-end) si può avere ulteriore riscontro del contenuti caricati.

Il test di invio delle Delibere di Consiglio diede in un primo momento un risultato non desiderato ma che si può comunque considerare positivo: Engineering non aveva eseguito correttamente il WS-Wrapping del servizio *InvioDelibereConsiglio* e il modulo Java registrava a log l'Xml di eccezione generica con tanto di descrizione puntuale dell'errore.

Il modulo Java terminava infatti l'esecuzione con la seguente dicitura a standard output:

(*ClassNotFoundExceptionPAAImporterDelibereConsiglioWrapper*)

Si considera comunque positivo il risultato del test in quanto tutti moduli applicativi coinvolti hanno fatto il loro compito e laddove si è verificato un errore esso è stato correttamente segnalato e gestito. Un altro tipo di problema, molto più grave, è sorto invece per via dell'elevato livello di disaccoppiamento introdotto dalle architetture SOA. La discussione di questo inconveniente è rimandata al capitolo "Conclusioni", in quanto evidenzia come il disaccoppiamento sia un risultato sperabile in un circuito di interoperabilità, ma introduca anche degli effetti collaterali notevoli.

7 – Conclusioni

Il presente progetto documenta la progettazione e lo sviluppo *in-house* presso un committente reale (Comune di Rovigo) di una soluzione per l' interoperabilità tra Pubbliche Amministrazioni. Il progetto si può considerare concluso e i relativi obiettivi conseguiti. Il committente infatti si avvale tutt'ora dell'applicativo contestualmente sviluppato per gli scopi che si era prefisso.

Come già detto, lo scopo generico del presente progetto era la costruzione di un canale dati in grado di far arrivare alcune categorie di atti amministrativi prodotti giornalmente dal Comune di Rovigo (memorizzati su DB Oracle) al CMS del portale multiente MyPortal 2.5, messo a disposizione da Regione Veneto. L'unico punto fisso stabilito fin dall'inizio è che per la realizzazione del canale dati si sarebbe adoperata un'architettura SOA. Questo obiettivo generale ha generato tre sotto-obiettivi:

- progettazione di un modulo di export dati da DB Oracle degli atti verso il formato Xml;
- l'adozione da parte del Comune di Rovigo di sistemi che realizzassero i paradigmi delle Porte di Dominio per la creazione di una SOA (nell'ambito del progetto SIRV-Interop) tra i sistemi del Comune di Rovigo e di Regione Veneto;
- l'interfacciamento del modulo applicativo con il canale dati e la schedulazione giornaliera, parziale e incrementale dell'invio quotidiano dei nuovi atti prodotti.

Tutti e tre i sotto-obiettivi sopraelencati sono stati considerati pienamente conseguiti dal committente stesso. Il sottoscritto tirocinante ha intrapreso in prima persona la progettazione e la realizzazione del modulo applicativo e del suo interfacciamento con le Porte di Dominio, mentre ha avuto il ruolo di persona di riferimento nell'interazione con i tecnici di Engineering per l'installazione e la configurazione del canale dati. La collaborazione con il personale di AS2, Engineering e del gruppo CReSCI è stata comunque sempre fondamentale.
Il sistema di interoperabilità risultante dal lavoro svolto e qui documentato funziona secondo le aspettative ma evidenzia anche dei limiti in parte perfettamente previsti e in parte da considerarsi come aspetti delicati su cui prestare la dovuta attenzione. Non essendosi trattato soltanto di un lavoro di pura programmazione, ma un vero e proprio ruolo di addetto all'interoperabilità tra PA è stato possibile accumulare una discreta esperienza riguardo le strategie adottate, da cui si possono ricavare riflessioni su pregi e limiti delle architetture di tipo SOA. A seguire una panoramica su punti di forza e debolezze del risultato finale visto nel suo complesso.

7.1 – Obiettivi conseguiti e punti a favore delle SOA

Funzionamento giornaliero, parziale e incrementale del modulo applicativo

Il modulo applicativo di export dati sviluppato funziona esattamente come sperato. Esso infatti esporta verso il formato Xml tutti gli atti amministrativi di interesse che ogni giorno vengono registrati a DB del Comune di Rovigo, senza ripetizioni e senza errori, a meno di fatti del tutto eccezionali come l'incoerenza di cui si è discusso nel capitolo "Risultati".

Invio di dati da un sistema legacy verso Web Service

L'invio degli Atti Amministrativi tramite Web Service funziona come sperato ed avviene in tempi e con occupazione di risorse ragionevole. Si noti che è stato specificato "ragionevole" e non "ottimale", poiché quello delle prestazioni è uno degli aspetti delicati che verranno discussi nel prossimo sottoparagrafo. Ciò che si vuole evidenziare adesso è invece il punto di forza dei Web Service applicati in questo ambito. Grazie ad essi è infatti molto semplice poter mantenere inalterato il funzionamento dei sistemi *legacy* di un ente, interfacciandoli con elaboratori, canali dati, e piattaforme hardware/software di nuova generazione, con sforzi contenuti. L'adozione di una soluzione ad-hoc in questo ambito sarebbe stata molto più costosa (si pensi ad esempio a quanto semplifichi la vita ai programmatori l'adozione do uno standard come Xml) e il cambiamento di alcuni sistemi *legacy* risulta non solo sconveniente ma spesso impraticabile.

Interoperabilità, autenticazione e scambio informazioni tra enti

Similarmente ai sopracitati benefici di "generalità" dei Web Service legati allo sviluppo di moduli applicativi, vi sono benefici analoghi anche in campo di interoperabilità in senso lato. I WS infatti riescono a far fronte a gravi problemi di scalabilità non solo informatica ma anche logistica e procedurale. Per spiegare meglio cosa si intende dicendo questo, verrà riportato un esempio. Si pensi di voler mettere a disposizione la consultazione delle informazioni registrate in un proprio DB per la consultazione da parte degli addetti di un ente partner. La soluzione ad-hoc di tale progetto di interoperabilità comporterebbe lo sviluppo di un applicativo basato sullo specifico DB e quindi dal sorgente molto poco riutilizzabile, se non per nulla. Servirebbe poi di effettuare una installazione di tale applicativo presso ogni ufficio da cui gli impiegati addetti dell'ente partner vorrebbero effettuare tali consultazioni. Si dovrebbe, per motivi procedurali, effettuare una registrazione individuale di ogni singolo addetto in modo che possa connettersi al DB con proprie credenziali e poter dimostrare di avere diritto alla consultazione. Tali credenziali poi dovrebbero anche essere correttamente aggiunte al DBMS invocato durante la consultazione. Infine, essendo una soluzione ad-hoc, si introdurrebbero probabilmente nuove esigenze legate alle policy di sicurezza delle reti aziendali di entrambi gli enti (potrebbe essere necessario aggiungere molte voci per l'apertura di porte Internet sui router/firewall aziendali, configurare indirizzi IP *trusted* addizionali e così via).
L'adozione dei WS invece decentra e semplifica tutti questi problemi appena elencati. L'utilizzo dello standard "intermedio" Xml facilita la progettazione di applicativi che in genere risultano sia più portabili che meno impegnativi da sviluppare. I paradigmi delle Porte di Dominio invece semplificano infinitamente le procedure di autenticazione rendendo così i canali di interoperabilità scalabili quasi all'infinito (si veda il capitolo 3 per approfondimenti su quest'ultimo aspetto). In una vera SOA infatti, si potrebbero sviluppare degli applicativi, per usare un termine leggermente improprio, *general purpose*, che agiscono ad "altissimo livello", essendo il WSDL e lo standard Xml basati su scambio di informazioni di natura molto più flessibile di quelle che vengono scambiate tra i moduli applicativi sottostanti.

7.2 – Aspetti delicati e sconvenienza delle SOA

SOA e sicurezza

Una delle critiche rivolte alle SOA più diffuse è sicuramente quella riguardante la sicurezza. Nell'ambito di una SOA infatti, l'utilizzo dei Web Service finisce per "dirottare" vari canali di scambio dati tra organizzazioni diverse tutti attraverso comuni connessioni Http. In pratica avviene quindi quello che in termini informatici viene chiamato *tunneling* delle comunicazioni attraverso la porta TCP 80 (o una porta designata equivalente). Di fatto tale tipo di connessioni vengono ammesse per default su ogni sorta di router/firewall aziendale, al fine di non rendere troppo complicata la normale navigazione Internet tramite browser da parte degli addetti. Solitamente questo genere di connessioni viene considerato poco pericoloso in quanto non dovrebbero contenere comunicazioni interattive di tipo critico (quali accessi a DB dedicati), e un'euristica avanzata per controllare la natura di ogni singola connessione sarebbe davvero troppo onerosa da realizzare, in grandi realtà con decine di addetti continuamente connessi. Se è vero che ormai moltissimi servizi realizzano *tunneling* sulla porta TCP 80 proprio per aggirare firewall aziendali (si pensi ai moderni protocolli di Desktop Remoto, per fare un esempio lampante di un utilizzo critico delle connessioni Http) e pur vero che questo potrebbe, col tempo, portare ad una interazione incontrollata tra sistemi di enti diversi, con scarse possibilità di monitorare le connessioni a fini di sicurezza. La genericità del protocollo Http apre quindi la porta ad eventuali malintenzionati, che invece dovrebbero faticare di più nel caso di connessioni dedicate, con protocolli segreti, e un sistema di identificazione ad-hoc degli elaboratori coinvolti e degli utenti.

L'utilizzo dei Web Service, dal punto di vista della sicurezza del canale dati, è consigliabile soprattutto per gli ambiti meno critici. D'altra parte, è piuttosto chiaro che le grandi esigenze di scalabilità, in cui i WS sono la soluzione ideale, spesso sono legate ad erogazione di dati poco sensibili, essendo fruibili da un vasto bacino di utenza.

SOA ed overhead

E' intuitivo pensare che il grande disaccoppiamento informatico, tecnologico e protocollare introduca un certo overhead, ovviamente indesiderato. Prove empiriche fatte nell'ambito del presente progetto lo dimostrano. Una discussione teorica è possibile ma probabilmente meno interessante, in questo caso. Avendo infatti un esempio pratico si preferisce fare riferimento a cosa, in concreto, si è verificato sul campo durante i test effettuati contestualmente al presente progetto.

Il primo overhead introdotto nei vari passaggi necessari per far pervenire un'informazione, quale un dato registrato a DB Oracle, fino ad essere memorizzato in un oggetto di un CMS su un sistema remoto tramite WS è ovviamente la conversione di tale dato nel formato Xml. L'operazione richiede, nel nostro caso, l'invocazione di un server ColdFusion, che esegue del codice ed effettua query verso il DB Oracle, e sulla base di un severo schema di validazione crea il file Xml. Nonostante questa operazione sia quasi trascurabile in termini prestazionali, è soltanto il primo piccolo passo per far arrivare l'informazione di partenza alla destinazione finale. Nel caso di una soluzione ad-hoc, forse sarebbe stato lo stesso elaboratore con installato il server ColdFusion a poter stabilire una connessione TCP diretta verso l'elaboratore remoto, su cui memorizzare direttamente il file Xml a file system.

Il secondo overhead è decisamente più delicato, soprattutto nello specifico caso di questo progetto. Le classi Java di interoperabilità ideate per l'invocazione di WS erano state infatti concepite per l'invio di soli documenti Xml. In una architettura di tipo SOA, nonostante la sua natura generica, ciò che normalmente si presuppone di fare è lo scambio di informazioni, o di oggetti che abbiano comunque natura descrittiva e non contenutistica (si pensi ad esempio anche alle architetture REST). Uno dei passaggi necessari alla pubblicazione degli atti amministrativi su pagina web invece è la contestuale pubblicazione di alcuni file binari a loro allegati. Un file binario è un'entità talmente generica da non poter essere contenuta in un modello descrittivo per essere trasmessa verso un sistema remoto. In poche parole, il file deve essere trasmesso, a meno di una codifica, bit per bit da un elaboratore ad un altro. Nello specifico, le classi Java usate nel presente progetto prendono ciascun allegato binario e ne effettuano una vera e propria codifica verso il formato testuale ASCII per poter essere inseriti come allegati di una busta SOAP. Prima di tutto, si è notato che la codifica ASCII di un file binario occupa maggior memoria di almeno il 20%. Secondo aspetto, e di certo più delicato, è l'inefficienza di tali classi Java, se utilizzate in questo modo, durante il run-time. L'array di Object che viene integralmente passato come parametro per il metodo che effettua la chiamata a Web Service deve contenere tutti gli allegati di un certo atto amministrativo. Per atti amministrativi con una totalità di allegati che pesi da circa 40 MegaByte in su si sono iniziati a verificare infatti errori di Java Heap Space insufficiente. Ergo la quantità di memoria fisica dedicata alla JVM risultava insufficiente, ma si tenga conto che tale valore di default corrisponde a 128 MegaByte. Questo significa che si era introdotto un overhead del 300% sull'occupazione di memoria fisica. Il problema fu aggirato allocando alla JVM fino ad un limite di 512 MegaByte.

Il terzo punto critico riguardo l'overhead è sicuramente l'effettiva chiamata a Web Service di Porta Delegata sulla piattaforma Tomcat. I primi test di invio con allegati di dimensioni trascurabili andarono subito a buon fine. Un primo, piccolo aumento delle dimensioni degli allegati portò invece a prestazioni sempre più scadenti, fino ad arrivare all'interruzione del web server per allegati che si aggiravano tra i 10 e i 20 MegaByte. Per capire quale fosse il problema, bastò collegarsi all'interfaccia web di Tomcat e restare in ascolto sul monitor delle risorse. Durante una singola invocazione del WS di Porta Delegata, lo spazio riservato al run-time locale di Java andava via via riempiendosi. Il problema era che il picco di occupazione massima corrispondeva ad una quantità di memoria pari a circa 10 volte la somma delle dimensioni dei singoli allegati. Lo spazio di memoria inizialmente dedicato al run-time di Java di Tomcat era di 64 MegaByte, che venivano esauriti già da un atto amministrativo con un totale di allegati per 7 MegaByte. Il problema fu risolto anche in questo caso allocando 512 MB di RAM alla JVM locale. Talvolta tale quantità continuava a risultare non ottimale, ma sufficiente a fare in modo che tutti i moduli continuassero a funzionare con operazioni di swap più piccole e gestibili.

Infine si tenga conto dell'overhead in senso lato che viene introdotto dall'utilizzo dei WS per lo scambio di dati. Il caso specifico di questo progetto ha evidenziato solo uno dei due grandi vantaggi delle SOA, ovvero quello della riusabilità di sistemi *legacy*. L'altro vantaggio, ossia quello legato alla scalabilità di un circuito di interoperabilità invece è diventato quasi un ostacolo. L'invio di atti amministrativi verso un CMS remoto è infatti un'operazione fortemente strutturata. E' del tutto ragionevole che essa avvenga ripetitivamente, sempre da parte dello stesso elaboratore, e con procedure e tipo di dati scambiati inalterati nel tempo. Nemmeno l'autenticazione tramite

asserzione di ruolo (vedi paragrafo 3.1) ha il minimo senso in questo caso. Detto questo, è chiaro che l'utilizzo di un canale dati dedicato, tra due sistemi, uno originante i dati ed uno in ascolto per la ricezione, con un protocollo che permettesse meno trasformazioni dei dati possibile, avrebbe avuto prestazioni decisamente migliori. Ciononostante, va pur sempre considerato che il grado di disaccoppiamento tecnologico di quanto prodotto nel presente progetto ha originato applicativi estremamente portabili e una procedura di invio atti di utilizzo semplicissimo.

SOA e disaccoppiamento

Un altro problema sorto durante la fase di test aiuta ad evidenziare il terzo ed ultimo aspetto critico delle architetture SOA, ovvero l'alto livello di disaccoppiamento. Se il disaccoppiamento è infatti una peculiarità che rende le SOA uniche nel loro genere, può diventare davvero un ostacolo durante alcuni tipi di *troubleshooting*. Un esempio di quanto accaduto aiuterà a capire meglio.

Si può fare un'analogia tra SOA e la rete Internet. Entrambe infatti sono nate per permettere lo scambio dati e l'interoperabilità tra sistemi hardware e software eterogenei. In entrambi i casi i sistemi *legacy* di una data organizzazione vengono in qualche modo "incapsulati" in un'architettura e protocolli di scambio dati di livello più alto. Sia SOA che Internet infine, sono basati sullo scambio di messaggi e non sono vincolate al tipo di dato effettivamente scambiato, o alle procedure che lo generano, offrendo un servizio di trasporto dati di tipo *best effort*. Questa ultima considerazione è forse di carattere non accademico e formale, ma risulta appropriata per descrivere un fenomeno che può verificarsi in fase di debug nei collaudi dei Web Service.

Durante uno dei test accadde che l'invio di una delibera di giunta non generava nessun errore (e anzi, il modulo Java terminava persino con un OK) ma la delibera non compariva su MyPortal, né lato front-end né lato back-end. Cosa era andato storto? La domanda aveva di sicuro una risposta ma essa non poteva essere immediata. L'alto livello di disaccoppiamento infatti, nel caso l'operazione desiderata non vada a buon fine, e non generi nemmeno un errore sufficientemente descrittivo, occulta quale che sia la radice del problema. Se il modulo Java di partenza è infatti convinto di aver creato correttamente la busta SOAP e di averla inoltrata al WS di Porta Delegata, ed esso termina ricevendo una "falsa" busta di OK (o anche se non terminasse per niente) è difficile perlustrare tutti gli step della consegna del messaggio, degli effetti sui sistemi remoti generati dalla consegna, e dalla successiva riconsegna del messaggio di risposta. Forse la busta SOAP si era perduta nel canale dati? Forse non veniva inoltrata all'importer di MyPortal dal WS di Porta Applicativa? Forse non era ben formata e l'importer si rifiutava di creare il contenuto su Alfresco? Forse dal punto di vista dei WS era tutto corretto ma su Alfresco si generava un errore legato a permessi di scrittura? Questa e molte altre ipotesi potevano essere formulate e, in assenza di un messaggio di errore descrittivo, erano tutte parimenti valide.

Si tentò quindi di procedere in maniera ingegneristica per la risoluzione del problema, analizzando tutte le fasi coinvolte dall'operazione nel suo complesso. Dapprima tecnici di Engineering vollero analizzare il codice Java che era stato scritto per vedere se generava un output corretto. Eliminati i dubbi sulla bontà del modulo Java, si preferì tentare subito di caricare manualmente la delibera desiderata su MyPortal con la procedura locale (che quindi escludeva l'uso di WS). La procedura manuale però diede esito positivo e i file descrittivi della delibera risultavano validi. Quindi si passò ad una procedura di "ascolto" dei log di sistema delle varie macchine coinvolte durante una vera prova di invio. Al telefono con tecnici di Engineering si effettuavano tentativi di invio della delibera

mentre loro restavano in ascolto sui log di sistema della macchina che effettua l'import dei file Xml. Tale elaboratore però non registrava nessuna ricezione di file Xml né nessun tentativo di connessione. Si ripeté quindi il test provando a monitorare i log di sistema della Porta Applicativa, per vedere se i file della delibera venissero bloccati in un punto intermedio, da qualche parte nella rete locale di Regione Veneto. Anche questo tentativo però diede lo stesso esito, ovvero nei log di sistema di Porta Applicativa non compariva nessun tentativo di connessione da parte dei sistemi del Comune di Rovigo. Per sicurezza quindi si ripeterono i test di echo delle PDD, che però andarono a buon fine. L'ultima parte che rimaneva da analizzare era quindi cosa succedesse nella piattaforma Tomcat del Comune di Rovigo e perché non tentasse nemmeno di effettuare l'invio verso PA della delibera. Dopo vari tentativi, ci si accorse che se i nomi file degli allegati binari di una certa delibera superavano i 32 caratteri complessivi (compresa quindi l'estensione) si verificava un bug sulla piattaforma Tomcat di PD, la quale interrompeva immediatamente ogni operazione e terminava con un OK. Il percorso a ritroso per l'individuazione di questo errore fu davvero oneroso anche se era topograficamente vicino, e fu chiaro fin da subito che per individuare da subito quale fosse nella comunicazione il nodo problematico sarebbe servita più fortuna che intuito. L'adozione di soluzioni ad-hoc sono tipicamente in grado di gestire meglio il tipo di errori che si possono generare, fornendo riscontri più dettagliati su cosa possa essere andato storto, mentre in un'architettura dove la chiave è il disaccoppiamento "ad ogni costo e il più possibile" è chiaro che ci sono più punti dove possono nascondersi errori che sono per forza di cose più imprevedibili. Nel caso specifico, il modulo Java "nulla sa" su cosa effettivamente faccia il WS di Porta Delegata, e proprio per questo motivo non può prevedere che dietro ad una sua risposta di OK (o dietro ad una sua non-risposta) possa essere stato generato un errore critico. Più impensabile ancora è che il modulo Java possa fare un'analisi dell'errore stesso.

Per questi motivi, è evidente che l'alto livello di disaccoppiamento è contemporaneamente sì uno dei maggior pregi, ma forse anche l'aspetto più delicato di una architettura SOA.

Infine, ecco una semplice considerazione riassuntiva di pregi e difetti di una SOA. Dall'esperienza del presente progetto risulta chiaro che il vero punto di forza delle SOA è la scalabilità. In secondo luogo viene il disaccoppiamento, da cui derivano però i primi problemi legati all'individuazione topografica di errori occulti. Scendendo infine nello specifico, lo scambio di dati tramite SOA risulta davvero inefficiente. L'adozione di una SOA è quindi probabilmente appropriata laddove le prestazioni non sono lo scopo principale, ed è sicuramente l'unica strategia consentita quando si vuole rendere adatti per nuovi scopi di business sistemi *legacy*, soprattutto in ambito di vasti circuiti di interoperabilità.