ALMA MATER STUDIORUM - UNIVERSITA' DI BOLOGNA
CAMPUS DI
CESENA

SCUOLA DI SCIENZE
CORSO DI LAUREA IN SCIENZE E TECNOLOGIE INFORMATICHE

Studio dell'approccio SOA realizzato mediante REST service su IBM WebSphere Commerce

Relazione finale in:
Programmazione

Relatore:

Prof. Antonella Carbonaro

Presentata da:

Matteo Dalle Fabbriche

Correlatori:

Dott. Federico Succi

Ing. Lorenzo Cavina

Sessione I

Anno Accademico 2013/2014

Prefazione

Il presente elaborato è stato svolto presso TECLA, azienda leader nella realizzazione di soluzioni e progetti integrati per il Digital Business e il Real-Time Web basati su tecnologia Internet, Intranet e Mobile, nonché Business Partner IBM di primo piano a livello nazionale per soluzioni ecommerce e collaboration.

L'elaborato oltre a tracciare un percorso di formazione e crescita personale e professionale, si inserisce pienamente nel contesto operativo di TECLA e si prefigge come obbiettivo quello di soddisfare alcune precise esigenze aziendali nate su progetti cliente reali.

In particolare la trattazione, che vuole essere un sunto del percorso fatto, verte sullo studio, l'analisi e la successiva personalizzazione di alcuni dei servizi REST presenti out of the box nella piattaforma IBM WebSphere Commerce.

L'elaborato è volutamente diviso in due parti strettamente correlate tra di loro, partendo da un contesto più generale fino ad arrivare ai dettagli tecnici più fini.

La prima parte, che come detto è più di contesto, è volta ad illustrare i principi e le filosofie dell'architettura SOA e la loro implementazione mediante web services, in particolare si tratterà di architetture RESTful, che per la loro semplicità stanno guadagnando consenso nel mondo enterprise.

La seconda parte è dedicata alla presentazione della piattaforma IBM WebSphere Commerce, ampiamente utilizzata all'interno dell'azienda TECLA, seguita da un'illustrazione delle modalità e del funzionamento dei servizi presenti sulla piattaforma, spiegando poi in che modo questa abbia adottato una soluzione architetturale SOA basata su web service RESTful.

Una volta esposte queste nozioni che facevano parte sia della preparazione dell'attività operativa finale, sia del percorso di crescita interna all'azienda, verrà analizzata la procedura volta a modificare il servizio REST presente nell'asset

standard di WebSphere Commerce.

Indice

Capitolo 1: Introduzione 8

1.1 Servizi nella realtà 9

1.2 Come IBM è presente nel panorama SOA 12

Capitolo 2: Web Service e Servizi REST 13

2.1 Introduzione 13

2.2 Cos'è un Web Service 13

2.2.1 Il servizio 14

2.2.2 Semantica 14

2.2.3 I messaggi 14

2.3 Interazioni client-server 15

2.4 Cosa sono i servizi REST 18

2.4.1 Funzionamento di base delle interazioni 19

2.4.2 Caratteristiche 19

2.4.3 Conseguenze nell'adozione di REST service 20

2.4.4 Come viene ovviato il problema della gestione dello stato 20

Capitolo 3: Introduzione a SOA 22

3.1 Introduzione 22

3.2 Cosa si intende per SOA 22

3.2.1 Definizioni basilari presenti in SOA-RM 23

3.2.2 Ecosistema SOA 24

3.2.3 Struttura sociale 25

3.2.4 Interazioni 26

3.3 Ruoli..... 27

3.4 Progettazione di SOA secondo IBM 27

Capitolo 4: WebSphere Commerce..... 32

4.1 Introduzione..... 32

4.2 Introduzione a WebSphere..... 32

4.2.1 Web application 32

4.2.2 Application Server e WebSphere Application Server 32

4.3 WebSphere Commerce..... 33

4.3.1 Funzionalità offerte 34

4.3.2 Supporto ai siti estesi (Extended Sites) 36

4.4 WebSphere Commerce e Tecla 37

Capitolo 5: WebSphere Commerce come piattaforma di servizi 38

5.1 Introduzione..... 38

5.2 Adozione dello stile architetturale SOA da parte di WebSphere Commerce ... 38

5.3 Facade..... 40

5.4 Client library..... 41

5.5 Service binding..... 42

5.6 WebSphere Commerce framework 43

5.6.1 Gestione della richiesta 43

5.6.2 Controller 43

5.6.3 Adapter framework 44

5.6.4 Gestione delle risposte 45

5.6.5 Persistence layer 45

5.7 Componenti utilizzati tra i layer di WebSphere Commerce 46

5.7.1 Enterprise JavaBean (EJB) 46

5.7.2 DataBean 47

5.7.3 Standard OAGIS e messaggi BOD 47

5.7.4 Service Data Object (SDO) 48

Capitolo 6: I servizi REST di WCS 50

6.1 Introduzione 50

6.2 Iter seguito da una richiesta REST 51

6.3 Front-end o presentation layer 53

6.3.1 WebSphere Commerce REST API 53

6.3.2 Connessioni criptate 54

6.3.3 Servizio di autenticazione 55

6.3.4 Caching 57

6.4 Business Logic Layer 59

6.4.1 Business Object Command Framework 61

6.5 Persistence layer 61

6.5.1 Business Object Mediator 62

6.5.2 I mediator 63

6.5.3 Data Service layer 65

6.5.4 Physical Data Container 66

6.6 WebSphere Commerce Search 66

6.6.1 Architettura del Search server 67

6.6.2 Apache Solr 71

Capitolo 7: Personalizzazione di un servizio esistente 73

7.1 Introduzione..... 73

7.2 Solr e il preprocessore 74

7.2.1 Le utility di Solr 77

7.3 Profili di ricerca 79

7.4 Mappatura 82

7.5 Estendere la configurazione del Business Object Mediator..... 83

7.6 Mappare l'URL..... 84

7.7 Verifica del risultato 85

7.8 Difficoltà riscontrate 86

Capitolo 8: Conclusioni..... 88

Capitolo 9: Bibliografia..... 90

Capitolo 1: Introduzione

Il Web nasce nel 1993 grazie a Tim Bernes Lee[1]; la sua idea iniziale era utilizzare una macchina che avrebbe preso il ruolo di web server per fornire accesso libero alla documentazione presente su di essa da parte di altre macchine indipendentemente dalla piattaforma hardware o software di queste, mediante l'uso di un programma chiamato browser.

I documenti allora disponibili erano insiemi di pagine collegate tra loro mediante parole chiave, definiti ipertesti.

Le pagine di ipertesto utilizzavano il formato HTML per essere rappresentate, esponendo cosi un contenuto statico, cioè l'unico modo disponibile per modificare il contenuto era modificare manualmente la pagina presente sul web server da parte di un'addetto.

Ben presto ci si rese conto dei limiti imposti da questa soluzione, cosi vennero introdotte lato server inizialmente le CGI, successivamente JSP, PHP, ASP, ecc.

Le CGI permettevano di invocare un'applicazione da server e trasmetterne il risultato come pagina HTML, implementando cosi il dinamismo nelle pagine.

Nel corso degli anni, grazie ai progressi tecnologici ed al crescente interesse verso l'argomento, internet e il web sono diventati sempre più invasivi, assumendo portata via via maggiore e assorbendo una grande quantità di ruoli in tutti gli scenari della nostra società.

Ad oggi un'application server è in grado di operare come host per le logiche applicative, gestendo l'output da inviare ai client e recuperando informazioni da basi dati; tuttavia l'evoluzione delle tecnologie adottate da questo componente gli ha permesso di diventare un fornitore e un consumatore di servizi, in grado quindi di

[1] *Tim Berners-Lee: Informatico inglese, inventore del web e direttore del W3C.*

sostenere interazioni non solo con esseri umani ma anche con altri calcolatori che espongono un'interfaccia apposita.
Gruppi di sistemi in grado di trattare servizi possono essere raggruppati in architetture modellate secondo scenari ben definiti per offrire a loro volta servizi di complessità crescente.
Queste tecnologie trovano largo impiego in campo aziendale dove le architetture utilizzate possono venir ricavate da una modellazione fatta grazie alle catene di produzione.
Un esempio potrebbe essere un'azienda che produce articoli e li vende on-line; quindi mette a disposizione un servizio per l'acquisto, questo fa uso dei servizi messi a disposizone dalle infrastrutture presenti in produzione per visualizzare la situazione inventariale degli articoli o trasmettere un'ordine avvenuto; l'infrastruttura di produzione usufruisce dei servizi messi a disposizione dai fornitori, ordinando parti qual'ora la situazione inventariale lo richiedesse.
Ad oggi numerose aziende adottano architetture che forniscono web service, permettendo ai servizi offerti di sfruttarsi a vicenda, e offrendo a nuove applicazioni dei mezzi per una facile interazione con questi.

1.1 Servizi nella realtà

Di seguito viene illustrato un elenco con una breve descrizione di come big dell'ambito IT abbiano implementato soluzioni volte a fornire servizi; indicando quindi casi d'uso tangibili delle tecnologie che si andranno ad illustrare.

Ebay[2]

Il celebre sito di aste on-line offre API per interfacciarsi con i servizi messi a disposizione.

Le API scambiano dati in formato XML, su cui passano le descrizioni delle aste richieste e le liste di quelle presenti, raggruppate per categoria o altri criteri.

Sono in oltre specificate le varie azioni che un client può eseguie mediante servizi, come gestione delle liste di oggetti, gestione di acquisti e vendte, ricerca di oggetti, oltre alla gestione di transizioni e checkout.

Facebook[3]

I servizi messi a disposizione offrono la possibilita di interagire col social network, permettendo inanzi tutto il login per accedere ai servizi, questi offrono la possibilità di consultare eventi, stati e foto di altre persone, si possono poi scambiare foto, stati propri ed altro.

Le API permettono anche di far interagire Facebook con altre applicazioni, offrendo l'integrazione di commenti e la funzione like ad esempio mediante altri siti o la condivisione di un'attività svolta su un'applicazione esterna al social sulla bacheca dell'utente.

Google

La stessa Google offre la possibilità di interrogare i propri servizi, mediante applicazioni scritte da terzi, offrendo eventualmente anche diverse API orientate al servizio ce si vuole usare.

Maps[4]

Un esempio può essere costituito da Google Maps API Services che offre una rapida consultazione al servizio mappe offerto da Big G[5] mediante richieste RESTfull.

---

[2] *Ebay API: http://developer.ebay.com/Devzone/guides/ebayfeatures/index.html*

[3] *Facebook Docs: https://developers.facebook.com/docs/*

[4] *Google Maps API Web Services: https://developers.google.com/maps/documentation/webservices/?hl=it*

Tra le operazioni permesse, oltre ovviamente alle informazioni di un punto richiesto mediante coordinate, figurano: le indicazioni per partire da un punto e arrivare ad un'altro, la possibilità di richiedere informazioni approposito delle altezze di una zona, Informazioni sugli orari nelle varie zone del pianeta, ecc.

Drive[6]

Il servizio di cloud storage offerto da Google, mette a disposizione dei servizi per la gestione dei file contenuti.
In oltre permette ad applicazioni esterne di poter interagire con esso; per esempio aprendo un file con un componente di terzi.

Amazon[7]

Il sito di e-commerce mette a disposizione i propri servizi sotto il nome di Amazon Web Service (AVS), questi non si limitano alla sola gestione di carrelli e transizioni commerciali, ma offrono anche servizi di calcolo cloud-based, storage, strumenti per l'analisi di volumi dati, ecc.

PayPal [8]

L'infrastruttura PayPal offre servizi per integrare i suoi sistemi di pagamento nei siti e nelle applicazioni che ne fanno richiesta, mediante l'uso di servizi REST o protocollo SOAP.

Gil esempi appena appena menzionati hanno lo scopo di chiarire i traguardi delle soluzioni studiate nell'elaborato, che trova un suo inizio nell'argomento dei web service, ovvero secondo il contesto qui sopra, l'architettura minima richiesta per poter fornire un servizio.

---

[5] *Big G: Soprannome acquisito da Google*

[6] *Google Drive SDK: https://developers.google.com/drive/web/*

[7] *Amazon Web Services: http://aws.amazon.com/*

[8] *PayPal Developer: https://developer.paypal.com/docs/classic/api/PayPalSOAPAPIArchitecture/*

1.2 Come IBM è presente nel panorama SOA

IBM impegna numerose risorse, umane, economiche, ecc; per offrire un insieme di servizi e soluzioni software, hardware volti a costruire e supportare un'architettura SOA.

Oltre all'offerta di prodotti per la realizzazione di architetture SOA, IBM ha aperto nel mondo 3 SOA Leadership Center, tra cui uno di questi a Roma.

I SOA Leadership Center sono strutture, a cui si può accedere fisicamente o via web, in cui è possibile rafforzare la propria formazione e trovare risorse utili in ambito SOA.

Capitolo 2: Web Service e Servizi REST

2.1 Introduzione

In questo capitolo, di carattere generale e introduttivo, vengono prese brevemente in esame le architetture web service e RESTfull, lo scopo è quindi illustrare come il funzionamento di un web application che segue questi modelli (nel caso dell'elaborato WebSphere Commerce) debba essere visto dall'esterno, ovvero come una sorta di blocco intercambiabile che sfrutta un'interfaccia ben definita per le interazioni col resto dell'ambiente.

Questo capitolo e il successivo costituiscono gli argomenti basilari su cui si sviluppa il percorso intrappreso in azienda e servono a rendere chiaro il contesto nel quale ci si muove all'interno di questo elaborato.

2.2 Cos'è un Web Service

Un'implementazione di web service può essere definita come un sistema software auto-contenuto e auto-descrittivo al servizio di un'applicazione che comunica sulla medesima rete grazie al protocollo HTTP.

I web service permettono di essere integrati da parte delle applicazioni in maniera semplice e rapida, fornendo mezzi standard per l'interoperabilità; le applicazioni possono andare da semplici richieste fino a complicate operazioni di business.

Di fatto un web service è una nozione astratta che deve essere implementata in un entità concreta come un agente, ovvero una parte software o hardware in grado di inviare e ricevere messaggi.

La Web Service Architetture [9](WSA) fornisce quindi modelli concettuali per definire i web service e le relazioni tra componenti, queste linee guida non specificano tuttavia come un web service debba essere implementato e non impongono

[9] *Web Service Architetture: http://www.w3.org/TR/ws-arch/*

restrizioni su come questi debbano interagire tra loro; vengono invece descritte le caratteristiche strettamente necessarie comuni a tutti i web service.

2.2.1 Il servizio

Un servizio è una risorsa che offre un set di funzionalità astratte mediante un'interfaccia che ne nasconde i dettagli implementativi garantendo indipendenza dalla sua implementazione, infatti se viene sviluppato un agente che mette a disposizione un web service scritto in un determinato linguaggio e in esecuzione su una determinata piattaforma, il servizio deve rimanere lo stesso in caso si decida di utilizzare un altro agente scritto in un altro linguaggio e in esecuzione su un'altra piattaforma.

2.2.2 Semantica

La semantica è il comportamento che ci si attende dal servizio, in particolare il modo di rispondere ai messaggi ricevuti, è un accordo tra richiedente e fornitore che non deve però necessariamente essere scritto o negoziato in maniera esplicita; mentre la descrizione esposta rappresenta un contratto che disciplina i meccanismi di interazione con un determinato servizio, la semantica rappresenta un contratto che disciplina il significato e lo scopo di tale interazione.

2.2.3 I messaggi

Un messaggio è l'unita base di dati inviata da una sorgente a una destinazione, viene rappresentata come una struttura dati definita nella descrizione del servizio.
Principalmente un messaggio si divide in due parti, la prima è la busta che consiste in un set non precisato di header, la seconda parte è il corpo del messaggio vero e proprio.
La busta incapsula il corpo del messaggio inserendo nelle intestazioni presenti anche le informazioni per il trasporto e l'instradamento del messaggio.
Il corpo invece contiene il messaggio vero e proprio o l'URI corrispondente.

Una variante che in questo elaborato merita di essere menzionata si verifica quando il messaggio è una richiesta HTTP, questo caso è molto usato nei servizi RESTles, qui l'header del messaggio coincide con l'header HTTP, mentre i parametri presenti nell'URL sono presi come contenuto del messaggio.

2.3 Interazioni client-server

Nelle interazioni si possono individuare due protagonisti principali: il provider che ospita l'agente su cui è implementato il web service e il client che desidera utilizzare il web service messo a disposizione dal provider; questi si scambiano messaggi secondo le meccaniche descritte nel Web Service Description (WSD) ovvero una specifica presente nell'interfaccia web del servizio, che lo descrive in un formato elaborabile da una macchina.

WSD definisce una sorta di accordo tra le due parti che governa le meccaniche di interazione del servizio come il formato dei messaggi, il tipo di dati da utilizzare, il protocollo di trasporto, le posizioni di rete su cui il servizio può essere invocato, ecc. Di seguito si presenta a grandi linee quello che può essere considerato un scenario tipico di interazione; ci sono tutta via alcuni casi in cui lo scenario può richiedere passi addizionali, la ridefinizione di questi, un'ordine di esecuzione diverso.

1. Il client ed il provider prendono conoscenza l'uno dell'altro, questo è necessario per poter avviare un'interazione in quanto è ovvio che chi vuole interagire con un'entità deve sapere che questa esiste.
   La condizione può verificarsi in due modi:
   Il caso più tipico è quello in cui l'agente richiedente avvia la comunicazione, in questo caso il client deve prendere coscienza del soggetto provider e quindi aver ottenuto in qualche modo l'indirizzo a cui inviare la richiesta, direttamente dall'entità provider o usando un servizio che trovi una descrizione del service utilizzabile (che conterrà un indirizzo da cui invocare il servizio) o mediante configurazione manuale.

Il secondo caso è assai meno comune, qui è l'agente provider ad iniziare lo scambio di messaggi ed è costui ad essere a conoscenza del soggetto richiedente dopo aver ottenuto in qualche modo l'indirizzo client.
Come il raggiungimento di questo step si verifichi dipende dall'applicazione ed è irrilevante nelle specifiche date dall'architettura.

2. Client e provider si devono ora accordare sulla descrizione del servizio e la semantica da utilizzare, trovando un'accordo che andrà a governare le interazioni tra richiedente e provider.
   Non significa che il client e provider debbano negoziare l'uno con l'altro, ma entrambe le parti dovranno avere la stessa comprensione della descrizione del servizio e della semantica; le modalità di accordo possono verificarsi in diversi modi tra cui vale la pena citare:
   - Il richiedente e il fornitore comunicano direttamente tra loro per concordare in maniera esplicita la descrizione del servizio e la semantica.
   - L'entità fornitore può pubblicare e offrire la descrizione del servizio e la semantica con un approccio take-it-or-leave-it, che il richiedente deve accettare senza modifiche come condizione di utilizzo (un esempio viene dato dal protocollo SOAP con la descrizione in formato WSDL).
   - La descrizione del servizio e la semantica (eccetto l'indirizzo di rete del servizio specifico) sono definiti come standard da un'organizzazione del settore e utilizzati da molteplici entità client e provider come standard. In questo caso il raggiungimento di un accordo è compiuto da entrambe le parti se conformi allo stesso standard; un esempio dell'approccio sono appunto i REST service.
   - La descrizione del servizio e la semantica (tranne l'indirizzo di rete del servizio) vengono definiti e pubblicati da parte del richiedente (anche se

sono scritte dal punto di vista dell'entità provider) e vengono offerte al provider con approccio take-it-or-leave-it.

Ciò può verificarsi ad esempio se una grande azienda richiede ai propri fornitori di servizi Web conformità ad una particolare descrizione di semantica.

In questo caso un accordo viene raggiunto dall'ente fornitore adottando la descrizione del servizio e semantica che l'entità richiedente ha pubblicato.

3. La descrizione del servizio e la semantica, in caso non siano già presenti, vengono dati in adozione all'agente client e all'agente provider.
   In altre parole le informazioni della descrizione devono essere passate o in input o direttamente implementate nel client e nel provider.
   Anche in questo step le specifiche dell'architettura non si curano dei mezzi utilizzati per soddisfare questa condizione, quindi un'agente potrebbe essere appositamente sviluppato per implementare una particolare descrizione del servizio e della semantica, oppure potrebbe essere sviluppato in modo più generale e la descrizione del servizio desiderato e la semantica potrebbero essere specificate dinamicamente in ingresso, oppure potrebbe essere creato per primo e la descrizione del servizio e / o semantica potrebbero essere generati o dedotti dal codice dell'agente.
   Indipendentemente dal metodo usato sia la semantica sia la descrizione del servizio devono essere noti tra gli agenti prima dell'inizio dell'interazione.

4. L'agente richiedente e il provider si scambiano messaggi per conto dei loro proprietari.

2.4 Cosa sono i servizi REST

Il world wide web opera come un sistema di informazioni interconnesse, imponendo severi vincoli come l'identificazione degli agenti all'interno del sistema o la chiamata delle risorse attraverso Uniform Resource Identifier (URI).

Gli agenti che interagiscono al suo interno rappresentano, descrivono, comunicano stati tramite rappresentazione di risorse in una varietà di formati ampiamente supportati (Es. XML,HTML,CSS,JPEG,PNG), mediante protocolli che usano URI per identificare direttamente o indirettamente agenti e risorse.

Uno stile architetturale utile per realizzare architetture Web Service aderendo ai vincoli e requisiti che le linee guida sanciscono, utilizzando le caratteristiche del web appena descritte è conosciuto come Representation State Transfert (REST).

L'argomento fu affrontato per la prima volta da Roy Fieding[10] nel 2000 all''interno della sua tesi di dottorato intitolata : "Architectural Styles and the Design of Network-based Software Architectures"[11] che trattava elementi base delle architetture software, tra cui approcci per l'utilizzo del web come piattaforma per l'elaborazione distribuita.

L'architettura REST tratta un insieme di principi architetturali volti alla creazione di un Web Service, ispirato al funzionamento del web, i cui agenti forniscono un'interfaccia e una semantica uniforme.

Lo sviluppo si concentra quindi sulla descrizione di risorse, elementi fondamentali di questa architettura, che consistono in un qualsiasi elemento identificato in maniera univoca per poter essere individuato e richiamato.

---

[10] *Roy Fielding: Informatico Statunitense, http://roy.gbiv.com/.*

[11] *http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.*

2.4.1 Funzionamento di base delle interazioni

L'interazione ha inizio quando il client effettua una chiamata ad un web service REST per richiedere una risorsa mediante l'URL, utilizzando l'URI come identificativo di risorsa su cui vuole lavorare e l'azione da eseguire scelta tra quelle che il protocollo HTTP mette a disposizione (GET,PUT,POST,DELETE); il server risponderà alla richiesta con un messaggio in un formato che lo standard non si preoccupa di definire.
L'URL tipo di una richiesta, Es."/webapp/risorsa/ID_settore", ha una parte fissa (webapp) e alcune parti variabili in base alla tipologia di risorsa (risorsa e ID_settore), questo permette al servizio di organizzare le risorse in categorie separate.

2.4.2 Caratteristiche

I servizi REST si basano per le interazioni sul modello client - server, ci deve essere quindi un server che mette a disposizione dei servizi fruibili tramite URL e un client che in caso di bisogno ne fa uso.
Le interazioni tra le entità sono stateless, ovvero una chiamata sarà indipendente dalla precedente e dalla successiva, questo perché il server non gestisce stati; da ciò conseguono benefici in quanto il provider non dovrà consumare risorse e capacità di calcolo per mantenerli; tuttavia il client può gestire stati a livello applicativo.
Appoggiandosi al protocollo HTTP i servizi REST beneficiano di caratteristiche come il caching delle risorse su client e su server, supporto ai server Proxy, supporto a connessioni criptate mediante HTTPS.
Le linee guida sanciscono che i REST service devono rimanere indipendenti dalla piattaforma che li invoca e da quella che li mette a disposizione.
Si possono identificare 2 gruppi principali di web services:

- REST-compliant Web services in cui lo scopo primario è manipolare le rappresentazioni XML delle risorse web, usando un insieme uniforme di operazioni senza stato.

- Web services arbitrari in cui il servizio è esposto come set di operazioni arbitrarie.

2.4.3 Conseguenze nell'adozione di REST service

REST prevede che la scalabilità del servizio e la crescita siano diretti risultati di pochi principi chiave di progettazione:

- Lo stato dell'applicazione e le funzionalità sono divisi in risorse web.
- Ogni risorsa è unica e indirizzabile invocabile mediante sintassi universale tramite URL.
- Tutte le risorse sono condivise come interfaccia uniforme per il trasferimento di stato tra client e risorse, questo consiste in un insieme vincolato di operazioni ben definite e un insieme vincolato di contenuti.

L'affermazione dei servizi RESTful nei confronti di altri standard quali SOAP si deve alla loro semplicità di utilizzo, in quanto i servizi REST si affidano ad HTTP per definire le operazioni disponibili sulle risorse, sgravando il client dall'adozione di metodi e interfacce dedicate all'accesso delle informazioni, contrariamente a protocolli come SOAP che si devono definire tutto ciò.

2.4.4 Come viene ovviato il problema della gestione dello stato

Come già citato la comunicazione intrattenuta da un servizio REST è stateless, ovvero il server non tiene traccia di relazioni tra richieste diverse, questa condizione come detto porta diversi vantaggi, ma anche svantaggi a cui si deve ovviare.
E' comunque compito del server prendersi carico dello stato delle risorse, ovvero l'insieme dei valori che questa assumerà in un dato momento e che il client mediante operazioni CRUD può modificare.
In oltre il server contribuisce allo stato dell'applicazione,che è frutto appunto delle interazioni tra client e server.

In fase di progettazione di un web server va tenuto conto quindi della gestione dello stato in termini di gestione delle risorse.
Es. per la gestione di un carrello su un sito e-commerce, può essere implementato l'oggetto carrello su server come risorsa REST, rendendolo di fatto sempre disponibile mediante URI .
Potrebbe essere vista come eccezione l'uso dell'autenticazione, dove una volta che le credenziali sono state verificate il server passa un token al client (segnalando cosi l'avvenuta autenticazione), che dovrà essere incluso in tutte le prossime richieste; di fatto quello che si fa è cambiare lo stato, nel caso le informazioni di autenticazione siano corrette; generalmente però questo viene delegato ad un componente esterno al servizio.

Capitolo 3: Introduzione a SOA

3.1 Introduzione

Dopo aver analizzato i web service, si illustra come questi siano idonei ad essere raggruppati in organizzazioni che riflettono caratteristiche di strutture sociali, in grado di interagire tra loro; inserendo cosi ciò che è stato visto in un panorama più ampio; l'obbiettivo di tali strutture è fornire servizi complessi.

3.2 Cosa si intende per SOA

Un insieme di Web Service può andare a formare un'architettura orientata ai servizi ovvero SOA (Service Oriented Architetture), questa è definita dal consorzio OASIS[12], che ne ha rilasciato un modello di riferimento, come "un paradigma per l'organizzazione e l'utilizzo di capacità distribuite, che possono essere sotto il controllo di diversi domini di proprietà".

I modelli di riferimento come SOA-RM[13] descrivono i concetti e le relazioni, mettendo in evidenza la distinzione degli elementi del dominio.

A differenza del modello, un'architettura di riferimento elabora ulteriormente ciò, mostrando un quadro più completo, come quello che viene coinvolto nella realizzazione e nelle entità.

Le architetture di riferimento possono essere elaborate a più livelli di dettaglio, ma si applicano ad una classe di soluzioni restando comunque indipendenti da qualsiasi soluzione specifica.

Un architettura è l'organizzazione fondamentale di un sistema, impiegata nei relativi componenti, nelle relazioni tra questi, nelle linee guida e nel design.

Le tecniche di progettazione di SOA offrono la realizzazione di un’architettura enterprise scalabile, che collega risorse on demand, basata su servizi (nello specifico

[12] *OASIS: Advancing Open Standards for the Information Society; consorzio no-profit con lo scopo di promuovere tra gli sviluppatori standard aperti per l'informazione globale.*

[13] *SOA-RM: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm.*

del caso in esame web service); queste tecniche devono in oltre essere in grado di affrontare l'identificazione, la specifica e la realizzazione dei servizi.
Le applicazioni realizzate con questo approccio devono poter essere utilizzate in una catena di forniture, ed essere esposte a terzi in modo che possano combinarle ed incapsularle in una nuova applicazione, prestandosi ad essere adottate da aziende che necessitano una discreta complessità in applicazioni e processi.

3.2.1 Definizioni basilari presenti in SOA-RM

Il modello SOA Reference Model definito dall'OASIS sancisce delle linee guida volte ad illustrare come i componenti dell'architettura debbano essere distinti e come dialoghino tra loro; all'interno del documento vengono spesso usate per definire dettagli dell'architettura le seguenti terminologie:
Modelli: Rappresentazioni astratte di alcuni aspetti di un sistema, tutti i modelli architetturali utilizzati in una determinata vista sono sviluppati utilizzando il punto di vista associato, un modello può partecipare a più viste, ma dovrà attenersi ad un solo punto di vista.
Stakeholder: Entità singola o gruppo che ha interessi o preoccupazioni a proposito di un sistema pur non presentando un bisogno o un requisito formale, ha comunque interesse per la "zona" su cui opera il sistema; all'interno di uno stakeholder gli attori possono avere esigenze molto diverse tra loro.
Viste: Rappresentano il sistema dal punto di vista di un insieme di relazioni correlate**.**
Punti di vista: Sono una convenzione per costruire o utilizzare una vista, si parte da un modello da cui si sviluppano le singole viste, stabilendo il pubblico e le finalità. Quindi la vista è la visone del sistema percepita da dei soggetti dallo stesso settore; il punto di vista definisce la prospettiva da cui è tratta la vista, la modalità e i vincoli che deve avere.

Al fine di mantenere coerenza tra le varie viste dell'architettura, ognuna fa riferimento ad un solo punto di vista.

Attori: Partecipano come parti interessate o delegate che agiscono per conto dei partecipanti (non hanno dovere di alcuna partecipazione nelle azioni che sono state incaricati di eseguire), sono impegnati in azioni che hanno un impatto sul mondo reale e il cui significato e l'intento sono determinati dalla semantica implicita o esplicita.

3.2.2 Ecosistema SOA

I servizi e le varie funzionalità possono essere distribuite tra domini diversi e di diversa proprietà che osservano politiche e condizioni d'uso diverse, questo è indicato come ecosistema SOA, ovvero un ambiente in cui le funzioni di business vengono implementate come servizi e dove dovranno produrre determinati effetti, volti a soddisfare le esigenze aziendali del mondo reale.

L'ecosistema SOA si interessa di tutti i soggetti che coinvolge e il ruolo che questi possono giocare al suo interno, come le esigenze degli stakeholder vengano soddisfatte, come vari attori si relazionino tra di loro attraverso le politiche e come comunicare e stabilire relazioni affidabili tra processi per arrivare al conseguimento di un risultato.

Qualsiasi sistema tecnologico che fornisce un servizio in questo ambiente è visto come un sistema basato su SOA, ovvero uno stakeholder IT progettato deliberatamente per essere in grado di funzionare all'interno di un ecosistema SOA.

Un sistema basato su SOA focalizza come gli attori interagiscano all'interno del sistema per fornire un risultato specifico riscontrabile nel mondo reale, i partecipanti che interagiscono con un sistema basato su SOA assumono il ruolo di attori, membri di una struttura sociale col ruolo di stakeholder.

I partecipanti sono in genere persone, in quanto una macchina solitamente non può prendere luogo ad una struttura sociale.

In questo contesto il servizio è il meccanismo che porta una capacità ad un sistema basato su SOA insieme alle esigenze delle parti interessate nel più ampio ecosistema.
L'ecosistema comprende parti interessate che sono partecipanti allo sviluppo, all'implementazione, alla gestione e all'utilizzo di un sistema e dei suoi servizi o possono non partecipare ad alcuna attività, ma sono comunque influenzate dal sistema.

3.2.3 Struttura sociale

Le azioni che vengono effettuate da persone sono eseguite all'interno di un contesto sociale, tale contesto è fornito da strutture esistenti nella società come comunità, associazioni, imprese, aziende, agenzie governative o intere nazioni; il contesto sancisce il ruolo giocato da ogni persona nella veste di attore in queste strutture.
Le strutture interagiscono tra loro in tutto il mondo, utilizzando trattati, contratti, regole di mercato, strette di mano, trattative e dove necessario ricorrono alla legislazione.
Esse interagiscono attraverso i loro confini di proprietà definiti o impliciti, perché c'è un vantaggio reciproco in questo, ovvero uno ha qualcosa che l'altro può fornire.
Il paradigma SOA è particolarmente attento alla progettazione, la configurazione e la gestione di tali sistemi attraverso i confini di proprietà, perché questo rispecchia le interazioni del mondo reale tra strutture discrete e i loro reali confini di proprietà.
La struttura sociale è quindi stabilita con obbiettivi impliciti o espliciti, spesso espressi in termini di azioni specifiche, necessarie per realizzare i corrispondenti obbiettivi, coinvolgendo un qualsiasi numero di soggetti e un gran numero di relazioni tra questi.
Qualsiasi persona può essere uno stakeholder in molteplici strutture sociali e una struttura sociale può essere un attore come parte di un'altra struttura sociale o in un'altra ancora completamente diversa;

come conseguenza di questa molteplicità di ruoli ci possono essere dei disaccordi, in particolar modo quando missioni e obbiettivi non sono allineati.
Una struttura sociale può assumere varie forme, l'impresa ad esempio è una struttura sociale con una distinta personalità giuridica, una comunità online , un mercato che presenta una struttura con acquirenti e venditori, le legislazioni di aree geo-politiche possono poi fornire un quadro in cui le strutture sociali operano.

3.2.4 Interazioni

Una struttura sociale può perseguire i suoi obbiettivi in due modi:

- Agendo da sola con propri mezzi.
- Interagendo con altre strutture, utilizzando le risorse messe a disposizione.

Molte interazioni avvengono all'interno di strutture sociali, ma queste possono anche attraversare i confini di proprietà per sfruttare i mezzi messi a disposizione da altre strutture, la modalità con cui questo accade è dettato dell'organizzazione della struttura; per esempio un'azienda può essere composta da aziende più piccole.
La validità delle interazioni tra le strutture sociali non è sempre chiara, spesso è determinata in ultima analisi dalla normativa vigente; per esempio se un cliente acquista un libro su internet, la validità della transizione sarà determinata dalla locazione fisica del venditore o del compratore o magari da entrambi.
I sistemi che interagiscono tra strutture sociali sono basati su SOA, la natura e la portata delle interazioni riflette il grado di fiducia che c'è tra gli attori; questi rapporti all'interno di un ecosistema SOA vengono resi espliciti e formalizzati prendendo parte a ciò che SOA reference model chiama contesto di esecuzione.

3.3 Ruoli

Oltre agli agenti protagonisti di un'interazione visti nei web service, SOA presenta ulteriori figure come:

Mediatore: Ruolo assunto da un partecipante, volto a facilitare l'interazione e la connettività offerta per la fruizione di servizi.

Proprietario: Partecipante che esercita diritto di proprietà in un servizio.

Generalmente i consumatori di servizi sono partecipanti che avviano le interazioni per soddisfare un bisogno, anche se come si è già visto nell'analisi dei web service non è sempre vero; in oltre diversi attori possono essere coinvolti per un'interazione al supporto di un servizio.

Il tipo di necessità che un servizio è destinato a soddisfare deve essere formalizzato e incapsulato come requisito, per poi presentarsi nel servizio finale come una delle capacità disponibili quando si accede alla descrizione del service; le capacità possono avere ripercussioni nel mondo reale; per esempio: un consumatore richiede un libro e questo viene ordinato.

Si possono offrire variazioni nell'erogazione del servizio per rispondere a requisiti che necessitano del servizio ma chiedono di soddisfare necessità e condizioni diverse da quelle implementate, ci può quindi essere una trattativa tra le parti implicita o esplicita, che continua fino a quando il consumatore non accetta l'offerta e invoca il servizio o non abbandona l'operazione.

3.4 Progettazione di SOA secondo IBM

In questo paragrafo viene analizzata la progettazione di SOA secondo le linee guida IBM, la scelta di analizzare questa versione del modello è fatta in quanto l'elaborato corrente propone una visone dei servizi presenti in un architettura di questo tipo, ovvero IBM WebSphere Commerce.

Le linee guida per creare un'architettura SOA offrono un'ottica per la definizione di servizi business-aligned e loosely coupled[14], la cui realizzazione deve fornire un alto livello di flessibilità in risposta a nuove richieste volte a soddisfare opportunità di business.

Un SOA consiste sostanzialmente in un set di servizi IT business-aligned, volti a soddisfare i processi e le finalità business di un'organizzazione.

I servizi web come detto dispongono di una descrizione utilizzata per ricerche, costruzioni ed invocazioni da parte del service consumer e possono essere raccolti in un'applicazione composta, la quale potrà essere invocata mediante protocolli standard e offrirà risorse sottoforma di valori.

Una vista astratta di SOA raffigura questa come una architettura a strati composta da servizi allineati con i processi business.

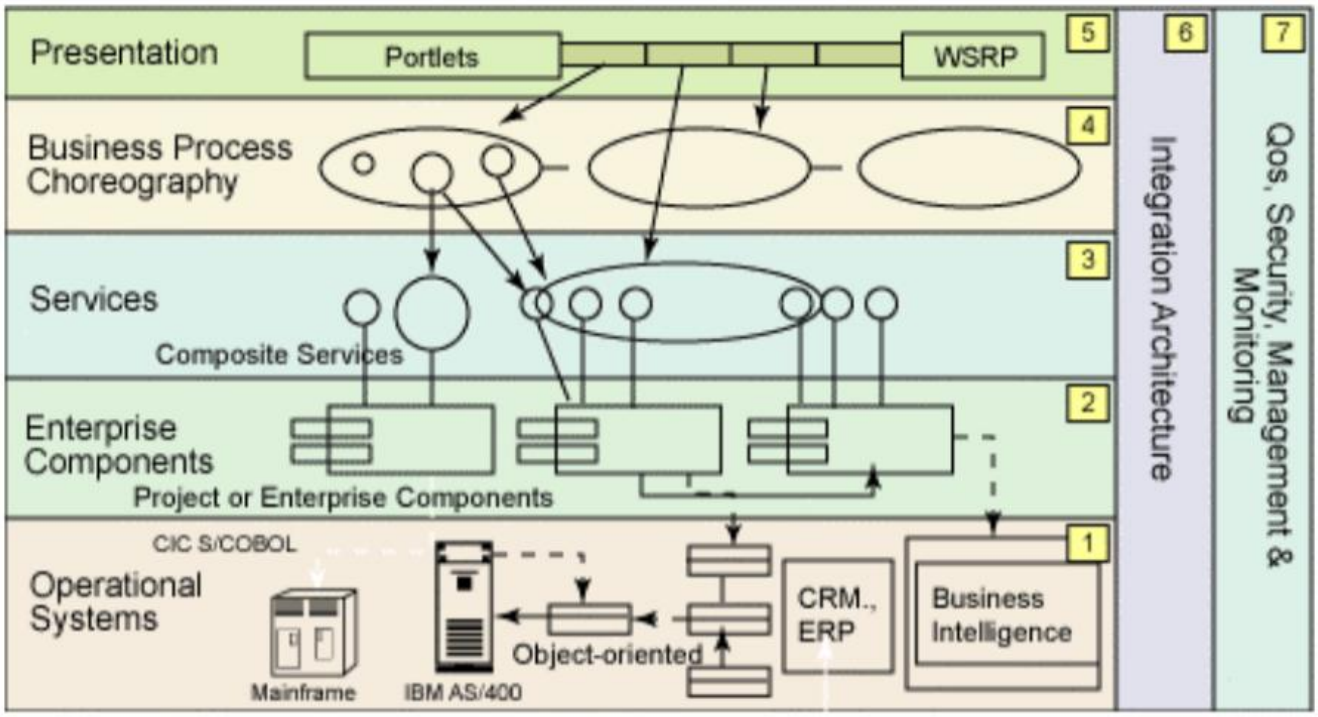

Figura 1: SOA rappresentata a livelli

[14] *Loosely coupled: debolmente accoppiati.*

Layer 1: Operational systems layer

Consiste in applicazioni esistenti costruite su misura, altrimenti dette legacy, tra cui CRM[15] e ERP[16], le implementazioni più vecchie del sistema object-oriented, cosi come le applicazioni di business intelligence.

L'architettura composta a strati di un SOA sfrutta i sistemi esistenti e li integra mediante tecniche service-oriented.

Layer 2: Enterprise component layer

Layer di componenti responsabile della realizzazione di funzionalità e del mantenimento del QoS dei servizi esposti.

Questi componenti sono un set gestito e disciplinato di asset che viene finanziato dall'impresa o dal business unit level.

Come attività su scala aziendale, essi sono tenuti a garantire la conformità agli SLA[17] mediante l'applicazione delle migliori pratiche architettoniche.

Questo layer tipicamente utilizza tecnologie container-based, come server application per implementare componenti, gestione del carico di lavoro, alta disponibilità e load balancing.

Layer 3: Service layer

I servizi che l'azienda sceglie di finalizzare ed esporre risiedono qui.

Essi possono essere scoperti o essere staticamente legati e poi richiamati o coreografati in un servizio composito.

Questo livello di esposizione prevede anche il meccanismo che prendere componenti in scala aziendale, componenti specifici business unit e in alcuni casi project-specific componented per esternare un sottogruppo delle loro interfacce sottoforma di descrizione dei servizi.

---

[15] CRM: Customer Relationship Management, http://it.wikipedia.org/wiki/Customer_relationship_management.

[16] ERP: Enterprise Resource Planning, http://it.wikipedia.org/wiki/Enterprise_resource_planning.

[17] Service Level Agreement: Strumenti contrattuali che definiscono le metriche di servizio, http://it.wikipedia.org/wiki/Service_level_agreement

Cosi i componenti Enterprise forniscono la realizzazione del servizio in fase di esecuzione, utilizzando la funzionalità offerta dalle loro interfacce.
Le interfacce in questo layer vengono esportate come descrizione di servizi, dove sono esposte per l'uso.
Possono esistere isolatamente o come servizi compositi.

Level 4: Business process composition or choreography layer

Le composizioni dei servizi esposti nel precedente livello sono definite in questo layer.
I servizi sono incorporati in un flusso attraverso orchestrazione o coreografia, quindi agiscono insieme come una singola applicazione.
Queste applicazioni supportano specifici casi d'uso e processi business.

Layer 5: Access o presentation layer

Sebbene questo layer sia usualmente fuori dallo scopo della discussione attorno a SOA, sta diventando gradualmente più rilevante.
Si può pensare come ad un futuro layer da tenere in conto per future soluzioni.
E' anche importante notare che SOA disaccoppia l'interfaccia utente dai componenti e che in ultima analisi necessita di fornire una soluzione end-to-end da un canale di accesso ad un servizio.

Level 6: Integration (ESB).

Abilita l'integrazione di servizi attraverso l'introduzione di un insieme di capacità affidabili come instradamento intelligente, protocollo di mediazione e altri meccanismi di trasformazione descritti come l'ESB.
Web Services Description Language (WSDL) specifica un legame, che implica una locazione dove il servizio è fornito contrariamene a un ESB che fornisce un meccanismo d'integrazione indipendente dalla posizione.

Level 7: QoS

Fornisce le capacità richieste per il controllo, la gestione ed il mantenimento di QoS come la sicurezza, performance e la disponibilità.

Questo è un processo in background, che attraverso meccanismi sense-and-respond e strumenti controlla la"salute" delle applicazioni SOA includendo tutte le implementazioni standard importanti di WS-Management e altri importanti protocolli, oltre alle implementazioni standard di quality of service per SOA.

Capitolo 4: WebSphere Commerce

4.1 Introduzione

Il capitolo vuole essere una semplice introduzione alla piattaforma di e-commerce utilizzata presso la ditta TECLA, presa come riferimento in questo elaborato, riassumendo qui parte del percorso di studio intrappreso in azienda.

La trattazione succesiva non può prescindere dalle tematiche che sono qui riportate anche se solo brevemente.

Vengono di seguito illustrate alcune delle funzionalità offerte o supportate da IBM WebSphere Commerce, al fine di rendere l'idea della tipologia e della portata del prodotto.

4.2 Introduzione a WebSphere

4.2.1 Web application

Un web service può essere concretamente implementato utilizzando set di web application, ovvero applicazioni web based distribuite e fruibili per mezzo di una rete.

Una web application è organizzata seguendo una struttura multi-tier, le richieste scaturite dalle interazioni vengono portate dal layer più esterno che opera come interfaccia per altri componenti o utenti, agli strati più interni passando dalle logiche applicative e finendo ai metodi di persistenza.

4.2.2 Application Server e WebSphere Application Server

Una web application necessita di un ambiente su cui essere eseguita che fornisca determinate funzionalità di supporto, offerte appunto da infrastrutture di tipo application server.

Un application server è una tipologia di server modulare, ovvero composta da parti realizzate seguendo standard ben definiti e largamente adottati, questo fornisce

funzionalità di supporto, sviluppo ed esecuzione di applicazioni in contesti distribuiti; il tutto in esecuzione su un'infrastruttura isolata dal sistema operativo e dall'hardware in uso.

Di questa categoria fa parte WebSphere Application server che offre un ambiente basato su tecnologia J2EE e un complesso di servizi orientati alla realizzazione di applicazioni ad architettura multilivello con inclinazione al settore enterprise, offrendo quindi supporto ad applicazioni con un alto grado di complessità e funzionalità di supporto come:

- Modulo web che espone la logica di presentazione statica delle applicazioni ai client , interagisce direttamente con la logica di business.
- Logica di business.
- Gestore degli accessi utenti e sicurezza.
- Gestore accessi a sorgenti dati esterne (Es. DB).
- Gestore transizioni.
- Interfaccia accesso a sistemi legacy.

4.3 WebSphere Commerce

IBM WebSphere Commerce o anche WCS[18] è una piattaforma software per e-commerce che include gestione di marketing, vendite, funzionalità di elaborazione ordini; per l'esecuzione la piattaforma richiede l'ambiente runtime WebSphere Application server.

Si basa su un'architettura di tipo three-tier, ovvero prevede una suddivisione in tre layer, ognuno dei quali dedicato ad un insieme di funzioni raggruppate per livello di astrazione.

L'interfaccia utente di solito rappresentata da un web server, espone contenuti come pagine html, permette di interfacciarsi con la logica funzionale, nascondendo di fatto i dettagli implementativi all'utente.

[18] *WCS: WebSphere Commerce Suite*

La business logic la logica applicativa che rende attiva una web application, si frappone tra l'interfaccia utente e il persistence layer.
Persistence layer è la parte deputata al recupero e al mantenimento dei dati.

La finalità della piattaforma è fornire soluzioni per marketing, campagne pubblicitarie e vendita trammite più canali; si rivolge a imprese di tutte le dimensioni, che supportano modelli commerciali di tipo:
Consumer Direct: Aziende che vendono direttamente i loro prodotti ai consumatori.
B2B: Negozio che tratta tra due aziende o parti, tipicamente il modello B2B vede aziende che acquistano beni da un altra attività di business, come può essere un grossista, un distributore, un rivenditore, ecc. Questa categoria a sua volta ha 2 sottocategorie

- ***Hub*:** Consente ai clienti o partner di accedere a servizi e prodotti di altri.
- ***Sito Esteso*:** Sito che consente al gestore di creare diversi negozi per diversi partner

4.3.1 Funzionalità offerte

Di seguito viene presentata la lista di funzionalità che WCS (WebSphere Commerce Suite) mette a disposizione per realizzare web application, volte a soddisfare le esigenze di business:

- **Asset per app mobili:** Applicazioni modello che possono essere modificate e personalizzate, da cui partire per realizzare il proprio market.
- **Remote Widgets:** Fotogrammi portabili al di fuori del proprio sito, in modo da poter esportare il marchio e prodotti anche su social network, blog, siti associati e affini.
- **IBM Gift Center:** Sistema per gestire e creare registri regalo per clienti ed eventi, garantendo la possibilità di acquistare e pagare questi direttamente dallo store.

- **Coshopping:** Permette di avviare la stessa sessione con Browser diversi da parte di un acquirente.
- **Pagamenti Punch-out:** Modello di pagamento e fatturazione effettuato da un servizio esterno.
- **Product Ranking:** Caratteristica che permette di tenere in evidenza gli articoli più acquistati e popolari per confrontarli con articoli simili.
- **Digital Wallet:** Gestore di attività promozionali.
- **Websphere Commerce Search:** Fornisce funzionalità avanzate di ricerca nei negozi, offre la visualizzazione di suggerimenti e auto-completamento.
- **Subscription Support:** Permette di gestire servizi come abbonamenti, creando un flusso di ordini a catena per automatizzare il processo.
- **Dynamic kit configuration with Sterling Configurator:** Abilita il sito alla vendita di prodotti configurabili e complessi.
- **IBM Product Recommendations dynamic recommendations in starter stores:** Sistema che permette l'inserimento dinamico di prodotti nella pagina principale dello store; si basa sulle cronologie di navigazione e acquisti dei singoli clienti.
- **integrazione con Facebook:** Permette di usufruire di servizi messi a disposizione dal social network, come la condivisione di prodotti o il "like".
- **Responsive Web Design:** Approccio alla creazione di pagine ottimizzate per più tipologie di dispositivo.

Starter Store presenti per business B2B:

- **Elitte Starter Store:** Contiene tutte le caratteristiche del Madison Starter Store, più altre funzioni rivolte a commercio B2B diretto.
- **Extended Site Starter Store:** Base per l'impostazione di una soluzione di siti estesi.

Supporto globale o locale: WebSphere Commerce è progettato per poter lavorare su un mercato globale, dove è necessario fornire supporto multilingua e contenuto mirato a seconda della zona geografica.

Marketing: Le attività di marketing possono essere utilizzate per inoltrare messaggi promozionali tramite mail o direttamente dal sito.

Il Marketing di precisione permette di creare campagne mirate basate sugli interessi degli utenti, queste informazioni vengono ricavate dalla loro navigazione e dai loro acquisti.

Distributed Order Managment integration: I sistemi DOM permettono di tenere traccia del ciclo vitale degli ordini dalla creazione fino alla loro evasione, permettendo di poter cambiare la priorità dell'ordine in base alle esigenze del cliente.

Un'altra caratteristica dei sistemi DOM è la possibilità di automatizzare la richiesta di materiale ai fornitori quando questo manca.

IBM Management Center per WebSphere Commerce: Strumenti di supporto agli utenti business volti alla gestione di marketing e merchandising.

Administration Tools (Data Load utility): Utility che permette di caricare velocemente i dati dal catalogo e dall'inventario al database.

Websphere Commerce Accellerator: Grazie a questo componente si è in grado di mantenere attivo il proprio store anche durante le modifiche.

Workspaces: Si tratta di un ambiente in cui possono essere provate modifiche al sito, senza andare a influenzare però la versione corrente.

Contracts and entitlement: Gestisce aspetti come i metodi di pagamento permessi in uno store, i prodotti acquistabili e i prezzi.

4.3.2 Supporto ai siti estesi (Extended Sites)

IBM Websphere Commerce supporta la possibilità di avere siti estesi, ciò si traduce nella presenza di più portali, ciascuno destinato a utenze con esigenze diverse ma

che condividono la stessa radice; per esempio un negozio che necessita di dover vendere i propri prodotti a livello globale, ha quindi necessità di avere un sito specializzato per ogni area geografica che si vuole coprire, che però mantiene determinate caratteristiche di base.
Questi siti hanno una sorta di asset comune; per esempio un catalogo che va poi modellato ed esteso in base alle caratteristiche della zona in cui dovrà essere operativo (quindi la lingua, gli articoli che espone, la valuta locale, ecc).
Per ogni sito che l'impresa deve presentare al mercato, è possibile creare un sito esteso che contiene tutte le configurazioni necessarie per posizionare in modo univoco il sito per un mercato specifico.

4.4 WebSphere Commerce e Tecla

Tecla, come detto, è partner IBM e offre ai suoi clienti soluzioni di e-commerce complete, basate su tale tecnologia.
Tecla non offre solo competenze, ma è in grado di ritagliare sul cliente soluzioni ad-hoc, che vanno dal digital marketing alla 'customizzazione' del frontend e del backend.
Tutto questo con l'obbiettivo di supportare il business dei propri clienti.
E' proprio nell'ambito della 'customizzazione' della piattaforma per rispondere alle esigenze di alcuni clienti che si inserisce questo lavoro e, più in particolare, quanto segue.

Capitolo 5: WebSphere Commerce come piattaforma di servizi

5.1 Introduzione

Il seguente capitolo illustra come WebSphere Commerce adotti le linee guida di SOA per implementare web service, mostrando in maniera generica come le richieste provenienti da più canali diversi vengano soddisfatte.

Vengono illustrate anche le tecnologie e le risorse messe a disposizione per supportare il basso livello di integrazione che ci deve essere tra i vari componenti dell'infrastruttura e tra il client e il server.

5.2 Adozione dello stile architetturale SOA da parte di WebSphere Commerce

La suite WebSphere Commerce adotta uno stile architetturale SOA basato su Web Service; fornisce un'interfaccia per metodi che mette a disposizione attraverso il network, questa nasconde i dettagli dell'implementazione rendendo indipendente l'uso dalla piattaforma hardware e software utilizzata.

I servizi di WebSphere Commerce sfruttano l'alto grado di integrazione tra essi, messo a disposizione dallo stile architetturale SOA, permettendo alla piattaforma di assumere i ruoli di:

Service consumer: Adattandosi cosi a scenari in cui deve agire da consumatore di servizi; ad esempio per interfacciarsi con sistemi di gestione degli ordini, sistemi di pianificazione delle risorse aziendali, ecc.

Service provider: Quando si permette l'accesso dall'esterno di funzioni business come i web service, allora WebSphere Commerce diventa service provider.

WebSphere Commerce si muove verso un approccio focalizzato sul supporto multicanale, invece di un approccio primariamente basato sul web, offrendo supporto a canali come:

Mobile Applcation: Applicazioni mobili che sfruttano un interfaccia nativa o il web browser.

Web Application: Possono includere storefront tradizionali o dotate di funzionalità web specifiche fornite attraverso i servizi REST.

Social Application: Applicazioni residenti all'interno di ambienti social come può essere Facebook.

Kiosk / Desktop Application: Collegano uno store fisico al corrispettivo on-line.

Questa molteplicità di canali è permessa grazie all'adozione di facade ben definite, che supportano i vari sottosistemi di business logic.

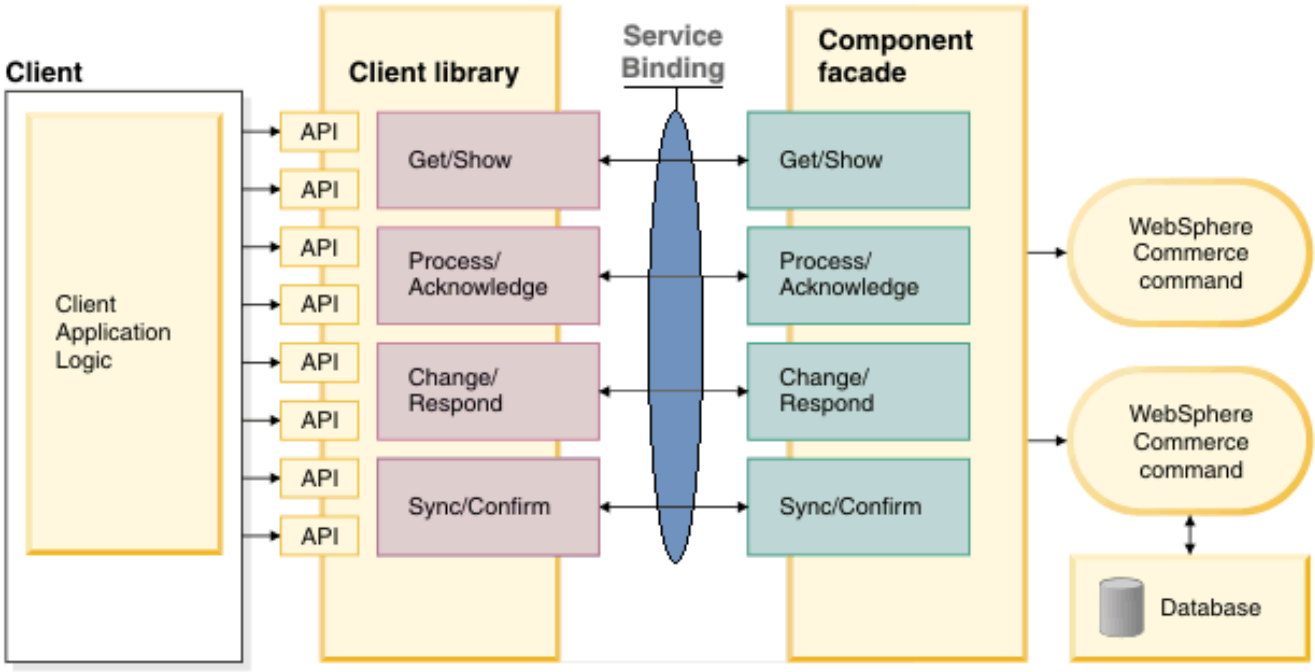

Figura 2: Interazioni che mediante Client Library vengono portate dal service Binding alle Component facade

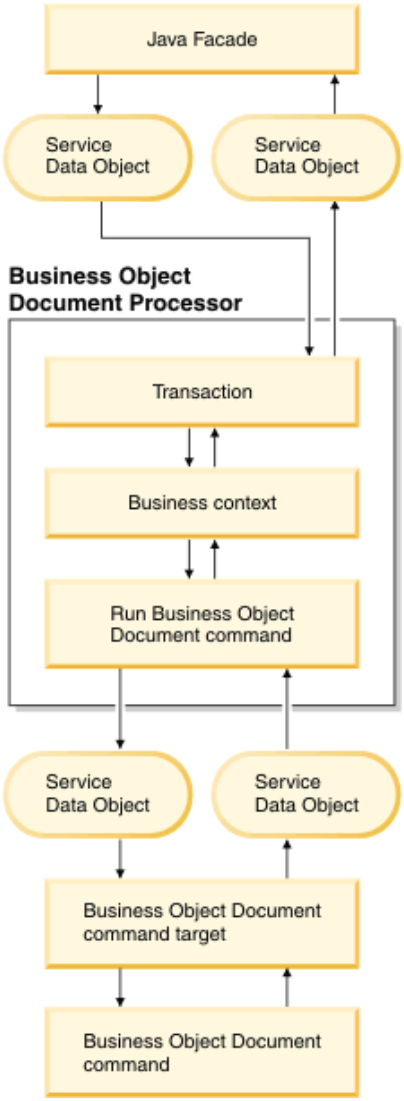

Figura 3: Dettaglio dell'elaborazione innescata da una facade

5.3 Facade

Le facade si collocano tra lo Struts[19] Framework e il command layer, vengono utilizzate per raggruppare una serie di servizi business correlati sotto un'interfaccia esposta; per esempio il catalogo degli ordini.

Vengono implementate come interfacce Java e i servizi corrispondenti come metodi su queste, per poi essere richiamati una volta ricevuto in ingresso il messaggio OAGIS contenente la combinazione verbo, che viene scelto tra Get, Change, Process o Sync e il nome corrispondente; fornendo cosi le logiche di business associate ai servizi richiesti.

L'architettura funzionale è strutturata attorno alla trasmissione di messaggi OAGIS dal client al server e il ritorno della risposta.

Sintatticamente il nome del metodo è il nome del verbo richiesto più il nome del sostantivo; per esempio una facciata potrebbe implementare il verbo Get sul sostantivo persona, quindi il nome del metodo associato a questa azione sulla facade è getPerson.

Questo approccio garantisce che le richieste siano indipendenti dal livello di presentazione, perciò il sottosistema può mantenere la stessa logica indipendentemente dalla provenienza della richiesta, sia da un browser o da un servizio web; in oltre permette di integrare facilmente applicazioni interne o integrarsi con applicazioni esterne adempiendo cosi alle linee guida di SOA.

[19] *Framework Struts: framework per lo sviluppo di applicazioni web, basato su Java EE, http://struts.apache.org/.*

5.4 Client library

La Client library è una libreria Java-compatibile, che fornisce un layer per integrare l'applicazione client con l'architettura dei servizi, senza dover inserire altro codice addizionale.

I client di un servizio di WebSphere Commerce possono utilizzare la libreria client per costruire la richiesta di servizio, questa si occuperà di astrarre tutta la complessità dei servizi Web richiesti, fornendo un meccanismo per creare oggetti che rappresentano la richiesta in formato OAGIS SDO e avviando la comunicazione col servizio, tutte le volte che il client richiederà l'oggetto Java corrispondente all'astrazione di un web service.

Ogni componente fornisce anche un progetto di client library per l'accesso mediante client comprensivo di:

Interfacce del servizio previsto per ciascun sostantivo.

Un file di costanti condivisibili tra client e server.

Un package che contiene:

Una Classe Java astratta con i metodi comuni e le modalità previste dal foundation framework.

Una classe Java Web enabled che estende la classe astratta con implementare i metodi rispettivi.

Le definizioni delle eccezioni dei sostantivi specifici.

Le client library offrono il supporto nativo a funzioni di autenticazione e alle sessioni.

Queste librerie forniscono anche un meccanismo per passare facilmente a comunicazioni tra i servizi Web e messaggi Java locali, a seconda della distribuzione, senza dover modificare il codice del client.

5.5 Service binding

Provvede al meccanismo di trasporto per i dati usando Service Data Object tra client library e servizio, può essere erogato sottoforma di web service o java binding locale.

Il service binding sfrutta la capacità degli SDO di poter serializzare e deserializzare i dati in base al meccanismo di trasporto.

Dal lato client questi vengono portati dalla loro forma Java a XML per essere trasportati attraverso il Web Service, una volta sul server si deserializzano di nuovo in oggetti Java.

Il service binding offre due tipologie di trasporto, passando comunque dalle client library e dalla facade implementata sui componenti richiesti:

Local enterprise bean: Connette il client al'implementazione della facade del componente all'interno della JVM[20] locale.

Quando il client e la facade sono implementati sulla stessa applicazione, il client comunica con l'implementazione della facade attraverso chiamate EJB locali per richiamare il metodo che corrisponde al servizio previsto.

Web Services: connette il client al'implementazione della facade richiesta, quando il client e la facciata si trovano in applicazioni separate, allora il client invia le richieste al servizio usando web service per il trasporto.

Si noti che il client e la facade in questo caso sono in applicazioni distinte, quindi per gestire l'autenticazione e le misure di sicurezza si fa affidamento alle funzioni specifiche del servizio Web utilizzato.

[20] *JVM: Java Virtual Machine*

5.6 WebSphere Commerce framework

Il WebSphere Commerce framework è definito dal server runtime, per eseguire la business logic delle richieste che possono arrivare da utenti o dal sistema che richiamano un web service.

Il WebSphere Commerce Framework gestisce quindi transizioni e sessioni, nello specifico definisce interfacce e implementazioni astratte Java che la business logic deve estendere ed implementare per poter eseguire i comandi dei diversi buisness process.

5.6.1 Gestione della richiesta

Il servlet engine coordina il thread pool dei protocol listener che gestiscono le richieste fatte mediante URL; ogni richiesta in ingresso infatti viene gestita da un thread separato.

Il protocol listener è un componente che prende le richieste passate in ingresso dal livello di trasporto e le invia all'adapter manager, che restituirà l'adapter pertinente alla richiesta e al dispositivo richiedente, al fine di dare una formattazione ottimale alla risposta se necessario.

5.6.2 Controller

Il controller è un componente che gioca un ruolo importante nel modello di programmazione per applicazioni commerce, il suo compito è far rispettare i comandi presenti nella business logic che un'applicazione o un'utente possono utilizzare.

Nello specifico Determina se l'utente deve essere autenticato prima dell'esecuzione del comando, controlla se è richiesta la connessione sicura HTTPS per l'URL, si assicura che i comandi siano eseguiti e esegue il commit delle sessioni.

5.6.3 Adapter framework

Il compito dell'adapter framework è determinare quale adapter deve gestire quale richiesta, ed effettuarne l'associazione.
Gli adapter sono componenti che variano in base al dispositivo usato per fare la richiesta.
Gli adapter effettuano alcuni compiti sulla richiesta prima di passarla al Web controller:

- Istruire il Web controller sul tipo di dispositivo da cui proviene la richiesta.
- Trasformare il formato del messaggio di richiesta in entrata in un insieme di proprietà, che i comandi di WebSphere Commerce possono analizzare.
- Fornire una sessione di persistenza specifica per il dispositivo.

I tipi di adapter di defaul sono:

HTTP browser adapter: Fornisce il supporto per l'invocazione di richieste di WebSphere Commerce command ricevute da browser o REST service mediante HTTP.
HTTP PvC adapter: Classe adapter astratta, può essere usata per sviluppare specific pvc adapter per dispositivi.
Per esempio se si ha necessità di sviluppare un adapter per una particolare applicazione per telefono, si può estendere da questo adapter.
HTTP Program adapter: Fornisce supporto per programmi remoti che richiamano i comandi di WebSphere Commerce.
L'adapter in questione riceve le richieste, usa una mappatura per convertire la richiesta in un oggetto CommandProperty.

Dopo la conversione il program adapter usa l'oggetto CommandProperty per eseguire la richiesta.

5.6.4 Gestione delle risposte

Per la composizione delle risposte WebSphere Commerce Web Service framework si affida al servizio di composizione JSP.

Questo approccio consente di personalizzare la risposta andando a modificare semplicemente il template JSP invece di dover aggiungere del codice Java, lo stesso è impiegato per la composizione delle pagine di errore, in cui la decisione di come debbano essere rappresentate spetta appunto alle JSP incaricate.

In una tipica web application, gli errori sono gestiti quando vengono lanciate le eccezioni corrispondenti.

La vista interna delle eccezioni è usata per comporre la risposta di errore, se questa non viene specificata allora si procede ad utilizzare la vista generica.

Per rendere noto l'errore al client, vengono offerte due metodologie, che lo sviluppatore può scegliere di adottare:

Includere le informazioni di errore nella risposta, in modo che il client possa consultare i dettagli di errore nella risposta, per controllare magari eventuali anomalie presenti nella richiesta.

Utilizza la definizione di fault corrispondente definita in WSDL, questa può essere associata con operazioni di servizio, che possono rilevare errori e reagire di conseguenza, invece di ispezionare l'oggetto di risposta.

5.6.5 Persistence layer

Prima gli Entity beans (variante di EJB) e successivamente gli SDO fisici, permettono di accedere al layer di persistenza, si tratta in ambo i casi di oggetti che rappresentano i dati, permettendone l'accesso senza dover prendere familiarità con lo schema che rappresenta i dati fisici.

5.7 Componenti utilizzati tra i layer di WebSphere Commerce

Di seguito vengono elencate alcune delle tecnologie utilizzate da WebSphere Commerce per la comunicazione standard tra i componenti e servizi al fine di supportarne il disaccoppiamento:

5.7.1 Enterprise JavaBean (EJB)

I bean sono classi java che incorporano al loro interno più oggetti diversi, cosi facendo è possibile passare più oggetti tramite uno solo.

Sono sviluppate seguendo alcune convenzioni: avere un costruttore senza argomenti, gli attributi devono essere accessibili mediante l'uso di funzioni get e set, non deve contenere metodi per la gestione degli eventi, deve in oltre essere in grado di salvare in maniera persistente il suo contenuto e ripristinarlo, ciò ne rende possibile la gestione mediante tool appositi.

Gli EJB sono un tipo di bean che implementa la logica di business di un'applicazione web lato server, per fornire servizi al front-end che oltre alle caratteristiche base dei bean offrono anche:

- Elaborazione delle transizioni.
- Controllo della concorrenza.
- Gestione degli eventi.
- Servizio directory.
- Sicurezza.
- Invocazione di procedure remote.
- Fornire servizi web.

5.7.2 DataBean

I data bean consentono l'accesso alle entità di WebSphere Commerce all'interno delle pagine, vengono utilizzati principalmente dai web designer in quanto consentono di popolare una pagina da parte di informazioni dinamiche al momento della visualizzazione senza dover scendere in dettagli implementativi.

5.7.3 Standard OAGIS e messaggi BOD

Tra i moduli e i layer di WebSphere Commerce si fa largo uso di messaggi BOD, per passare richieste e risposte

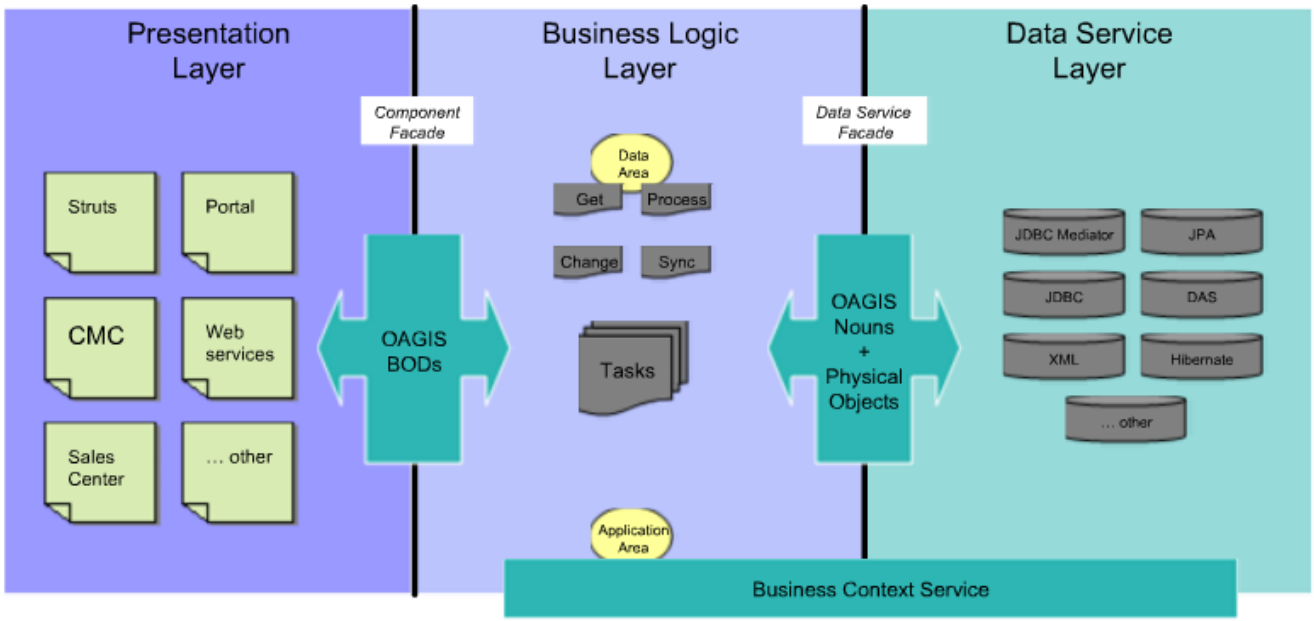

Figura 4: Dettaglio che sottolinea gli ambiti d'uso di BOD

OAGIS è uno standard creato per garantire interoperabilità tra sistemi diversi. Concettualmente OAGIS si basa su gruppi di messaggi, che raggruppati formano scenari; uno scenario è in grado di fornire le informazioni dettagliate per eseguire un determinato compito semplice o complesso.

L'Open Application Group Integration Specification fornisce quindi modelli di scenari, che possono essere utilizzati come punto di partenza per la modellazione di uno scenario che risponda alle esigenze.

Identificando lo scenario che più si avvicina all'esegineze correnti, è possibile identificare il tipo di messaggio di cui si necessita.

Il Business Object Document (BOD) è un tipo di architettura per messaggi business e per documenti enterprise, che vengono scambiati tra applicazioni presenti in catene di fornitura delle imprese.
La struttura dei messaggi BOD contiene un'area per applicazioni e un'area per i dati. L'area per le applicazioni descrive il contesto da associare all'elaborazione, va quindi associata al contesto di un'applicazione specifica, per controllare il processo e indicare il contesto applicativo dove ritornare il risultato.
Nel contesto dei servizi di WebSphere Commerce l'area del'applicazione è dove si archiviano negozio, lingua e altri dati di tipo sessione.
L'area dei dati invece contiene operazioni di servizio, specificate dalla coppia: ‘verbo’, ‘sostantivo‘;
dove il sostantivo specifica l’oggetto su cui eseguire l’operazione, mentre il verbo indica che operazione deve effettuare il servizio.
In caso di richiesta BOD conterrà i dettagli dell'operazione, in caso di risposta invece conterrà informazioni pertinenti alla risposta.
La comunicazione basata su messaggi BOD è bidirezionale e indipendente dal meccanismo utilizzato.
BOD può essere utilizzato con semplici protocolli di trasporto come HTTP e SMTP, ma può anche essere utilizzato in protocolli di trasporto più complessi come SOAP, ebXML trasporto e al routing, o qualsiasi altro sistema di integrazione delle applicazioni aziendali.

5.7.4 Service Data Object (SDO)

SDO tratta di una specifica progettata per rappresentare dati provenienti da fonti diverse, come documenti XML o database.
SDO organizza i dati come grafi contenenti oggetti, che possono essere facilmente aggiornati, in oltre tiene il codice di accesso ai dati separato dal codice dell'applicazione, supporta l'implementazione personalizzata dell'accesso ai dati.

Gli SDO si compongono di:

SDO client: Client SDO invece di utilizzare le API specifiche per interfacciarsi con una sorgente dati, utilizza il framework SDO e i tipi di dato che questo mette a disposizione.

Data mediator service: Il Data mediator Service è responsabile della creazione di un grafo a partire da una sorgente dati e dell'aggiornamento della sorgente dati, in base alle modifiche subite dal grafo.

Data sources: La fonte dati che può non limitarsi a raggruppare le sole fonti di back-end.

Data Object: Si tratta degli oggetti di servizio che rappresentano l'SDO in maniera strutturata, forniscono una visione comune del dato.

Data Graphs: Forniscono una visuale ad albero per i Data object, vengono ricavati a partire dalla sorgente dati.

Un client SDO può navigare all'interno del grafo, modificarne gli oggetti presenti nei nodi e successivamente le modifiche saranno ratificate anche alla sorgente dati.

Change summary: Sono contenuti dai grafi e usati per rappresentare i cambiamenti fatti ad essi, sono poi utilizzati dal DMS[21] per aggiornare in modo efficiente e incrementale le sorgenti dati.

[21] *DMS: Data Management System*

Capitolo 6: I servizi REST di WCS

6.1 Introduzione

Oggigiorno i clienti di Tecla richiedono un sempre maggiore disaccoppiamento tra client, frontend e backend.

gli stessi contenuti devono essere fruibili con modalità anche molto diverse, corredate da funzionalità ed elaborazioni specifiche: dal browser desktop, al tablet, alle app mobile fino all'integrazione con sistemi di terze parti.

I servizi REST della WCS supportano TECLA ed i propri clienti in questo; a volte però lo standard può non bastare.

Per questo occorre 'customizzare' la piattaforma, ma per fare ciò serve una profonda conoscenza dei meccanismi interni.

Nel seguente capitolo viene preso in esame l'iter seguito da una richiesta REST in WebSphere Commerce, partendo dalla richiesta, passando per le API REST, per il Business Object Command framework, fino ad arrivare agli SDO fisici e al motore di ricerca.

L'approccio scelto per l'analisi è di tipo top-down, in quanto l'iter seguito per la sua evasione segue una procedura con un ordine analogo; in oltre questa sequenza permette di partire da un concetto già descritto precedentemente, per poi avanzare nel dettaglio delle operazioni compiute nell'applicazione.

6.2 Iter seguito da una richiesta REST

In questa sezione viene esposta la procedura che porta all' evasione di una richiesta REST, mostrando le interazioni tra i componenti del business logic.

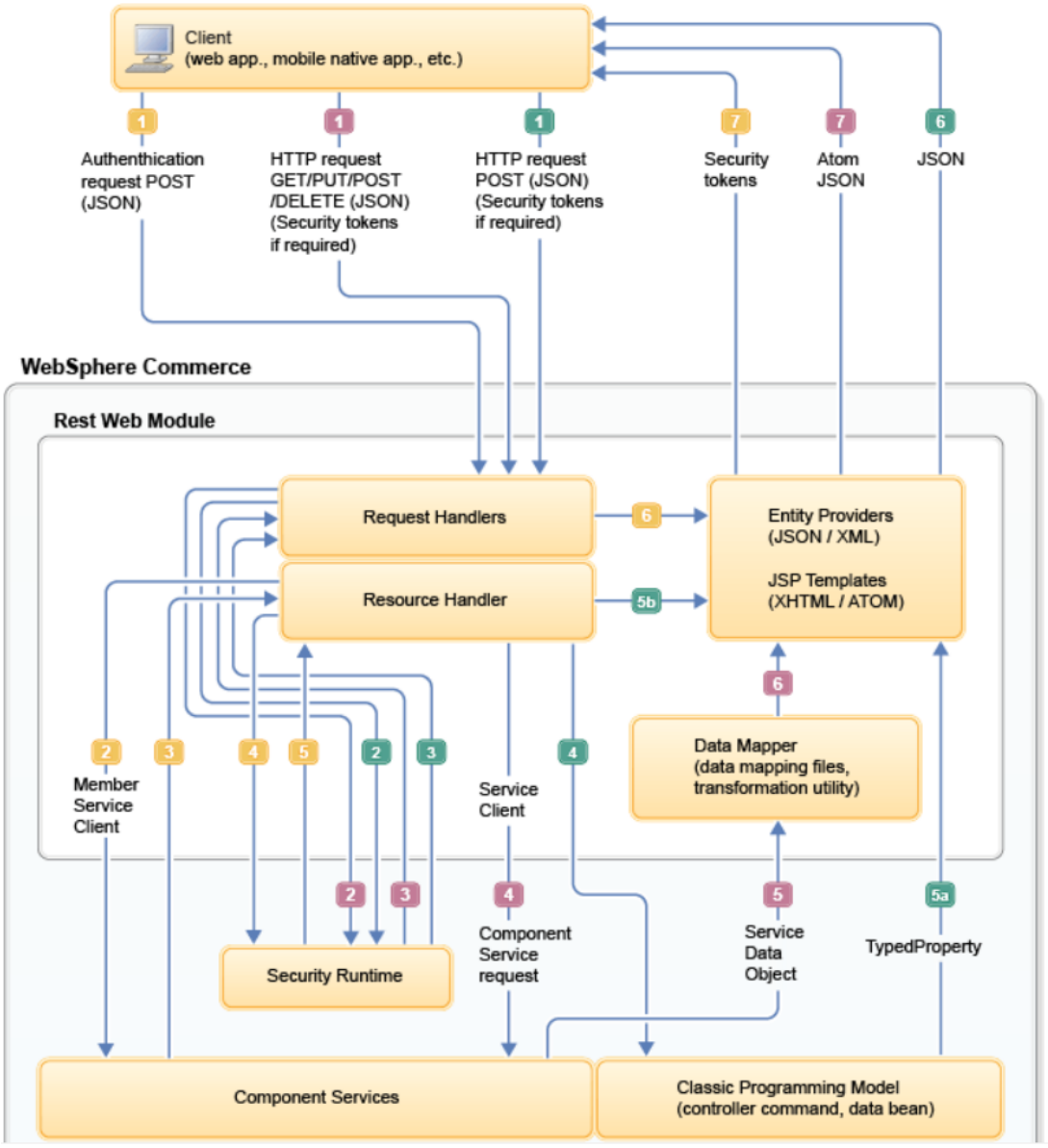

Figura 5: SOA Flussi che processano le richieste REST, in giallo sono riportati quelli che fanno uso di autenticazione, in viola quelli che non ne fanno uso

Flusso con autenticazione:

1. Il client manda una HTTP POST per richiedere l'autenticazione dell'utente registrato o un'ID temporaneo per un ospite.
2. Il framework JAX-RS invoca il Request Handlers che confronta l'URL della richiesta con l'appropriato Resource Handler, successivamente la richiesta http è convertita e inviata al servizio Component Services.

3. Component Services ritorna il risultato contenente le informazioni di autenticazione.
4. Vengono chiamate le runtime di sicurezza per la creazione i token di autenticazione WCToken (usato con connessioni non sicure) e WCTrustedToken (usato SOLO per connessioni sicure).
5. Ritorno dei token.
6. Viene chiamato l'entity Provider per generare la risposta contenente la risorsa richiesta.
7. La risorsa viene inviata al client.

Flusso senza autenticazione:

1. Il client invia la richiesta HTTP contenente nell'header il token di sicurezza WCToken (l'unico che può essere usato in connesioni non sicure)(Nota : Se PUT e DELETE non sono supportate si può utilizzare la proprietà "X-HTTP-Method-Override" presente nell'header della richiesta).
2. Viene invocato il Request Handler, che crea il contesto e chiama il Security Runtime.
3. Verifica dei Token di autenticazione da parte del Security Runtime.
4. Il Request Handler converte la richiesta in formato WebSphere Commerce OAGIS Service e la invia al Component Services.
5. Il server ritorna il risultato in formato BOD.
6. Il file Data Mapper Configuration viene caricato dall'entity provider per convertire la risposta BOD in JSON.
7. La risposta viene ritornata al client.

6.3 Front-end o presentation layer

E' il servizio di interazione che aggrega la logica aziendale per formare un'applicazione.

Questo interagisce con business logic layer tramite messaggi OAGIS e non contiene di per sé nessuna logica di business.

Il livello di presentazione non può interrogare il database direttamente per il recupero di dati, deve per questo e per eseguire qualsiasi altra logica di business interagire con altri componenti.

6.3.1 WebSphere Commerce REST API

WebSphere Commerce REST API sono basate su Apache Wink, framework per servizi REST implementato sulle API JAX-RS[22].

WebSphere Commerce estende le API REST al fine di avere ulteriori parametri oltre a quelli standard per la selettività delle query e la flessibilità della risposta, quindi l'URL della richiesta può includere anche:

responseFormat: Il formato di risposta alla richiesta; per impostazione di default il formato è JSON, ma anche XML è supportato senza dover implementare estensioni (questo parametro ha la precedenza rispetto al campo HTTP Accept request header).

lanId: language ID utilizzato durante l'iterazione, se questo non viene specificato, viene usato il language Id dello store (langId ha la precedenza sul campo locale).

locale: Località da cui il service consumer esegue la richiesta.

currency: La valuta monetaria che si deve usare durante l'iterazione.

catalogId: Identificativo del catalogo in esame nell'iterazione corrente (se questo campo non viene incluso verrà usato il catalogId di default).

forUser: User name dell'utente che ha eseguito la richiesta.

workspace.name: Il nome del workspace in uso.

---

[22] API JAX-RS: sono API forniscono supporto alla creazione di web services, utilizzando il modello architetturale REST, https://jax-rs-spec.java.net/

In oltre sono presenti parametri che fanno uso delle funzioni di ricerca offerte da WebSphere Commerce REST API:

profileName: Se presente permette di specificare un profilo di ricerca diverso da quello di default IBM_ findCategoryByUniqueIds.

responseTemplate: Specifica come le risposte debbano essere strutturate; il valore 1 ad esempio genera una risposta di tipo REST, il valore 0 che è quello di default genera una risposta di tipo BOD.

CatalogEntryView: Mappa gli accessi del catalogo all'interno delle risposte Product REST.

CatalogGroupView: Mappa la sezione categorie all'interno della risposta Category REST.

WebContentView: Mappa la sezione di contenuti web all'interno della risposta SiteContent REST.

6.3.2 Connessioni criptate

Alcuni servizi possono richiedere l'utilizzo di tecnologie per la trasmissione criptata dei dati; visto che le architetture REST sfruttano HTTP per lo scambio di messaggi, è possibile in WebSphere Commerce impostare l'utilizzo di SSL mediante il file di configurazione: Rest.war/WEB-INF/config/com.ibm.commerce.rest/wc-rest-security.xml, che tiene in formato XML l'elenco degli URI delle risorse, che necessitano di essere trasferite con protocolli sicuri.

Es .<sslConfig resource="store/{storeId}/cart/@self/checkout" enabled="true"/>

In caso si provi a trasferire una risorsa il cui indirizzo URL è presente nel file wc-rest-security.xml con HTTP, verrà restituito un messaggio di errore.

6.3.3 Servizio di autenticazione

WebSphere Commerce offre la possibilità di proteggere determinate risorse dall'uso pubblico, rendendole disponibili previa autenticazione dell'utente mediante una delle tre metodologie offerte:

loginidentity : Vengono utilizzate le credenziali Username e Password, per eseguire l'autenticazione di un utente già registrato.

guestidentity : Viene fornita un'identità provvisoria, per un utente ospite non registrato.

ltpaidentity : Token LTPA[23] per autenticare un utente nel caso di autenticazione parziale, in cui questo token è memorizzato in un cookie.

Una volta effettuata l'autenticazione mediante uno dei tre metodi, si otterranno dal server i seguenti valori:

```
"WCToken": "xxxxxxxxxxxxxxxxxx",
"WCTrustedToken": "xxxxxxxxxxxxx",
"personalizationID": "1321550980363-1",
"userId": "2"
```

PersonalizationID dovrà essere passato in tutte le successive chiamate REST, in quanto richiesto da alcuni servizi.

WCToken e WCTrustedToken sono i token di autenticazione forniti che identificheranno il client autenticato nelle prossime interazioni coi servizi offerti dal dominio, il loro impiego cambia a seconda della connessione che si usa.

Nel dettaglio per ogni richiesta seguente all'autenticazione si avrà:

---

[23] Token LTPA: Lightweight Third Party Authentication, http://en.wikipedia.org/wiki/IBM_Lightweight_Third-Party_Authentication

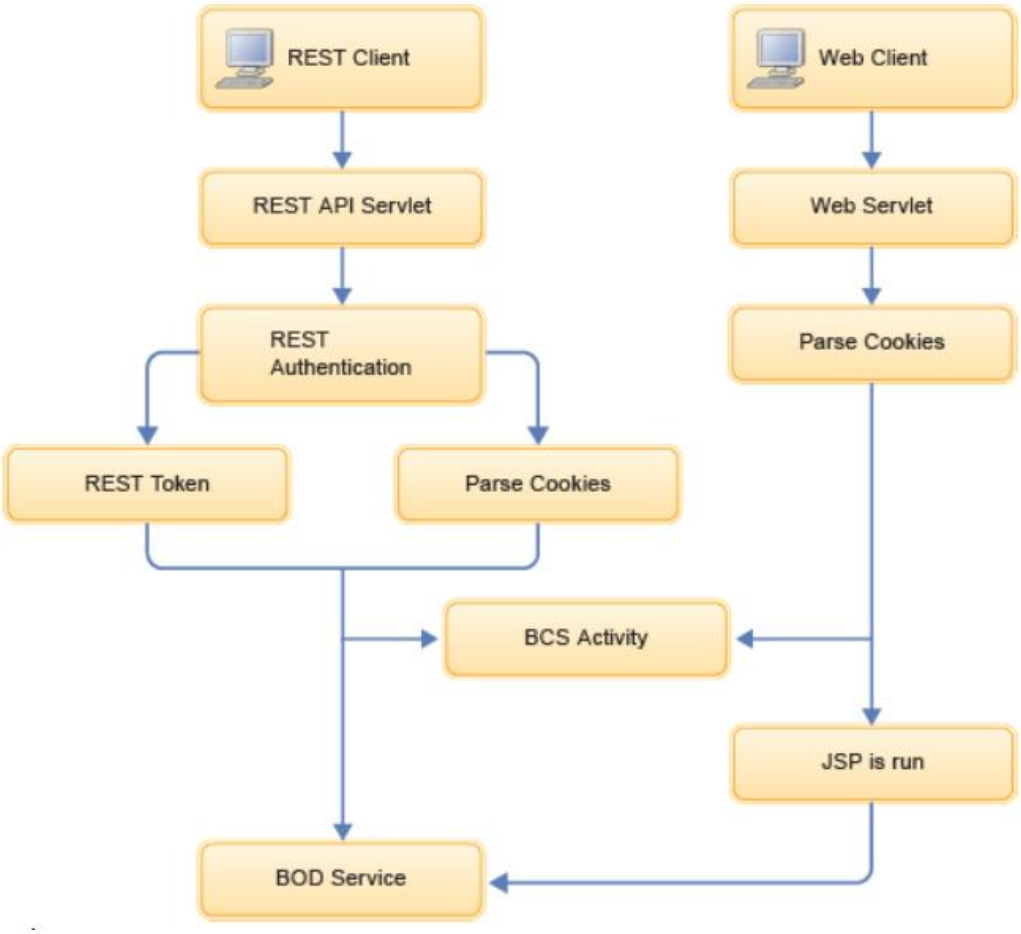

Figura 6: Interazioni per l'autenticazione.

Per le chiamate effettuate su canali sicuri:

1. Se l'header WCTrustedToken è presente viene usato.
2. Se il cookie WC_AUTHENTICATION_* è presente, l'utilizzo dei cookie è permesso, allora si usa il cookie WC_AUTHENTICATION corrispondente allo storeID specifico.
3. Se il cookie WC_PERSISTENT è presente, l'utilizzo dei cookie è abilitato, la sessione è abilitata per il servizio, allora viene usato il cookie WC_PERSISTENT; altrimenti un'eccezione indicherà che l'autenticazione parziale non è permessa.

Per le chiamate effettuate su canali non sicuri:

1. Se l'header WCToken è presente viene usato.
2. Se il cookie WC_USERACTIVITY_* è presente e l'utilizzo dei cookie per il REST è permesso, si usa WC_USERACTIVITY_* corrispondente allo storeID specifico.
   Quindi si imposta il cookie WC_USERACTIVITY nella risposta, con un valore di timeout aggiornato solo se la scadenza è entro la soglia configurata, per evitare aggiornamenti su ogni richiesta.
3. Se il cookie WC_PERSISTENT è presente e l'utilizzo dei cookie è permess, viene usato il cookie WC_PERSISTENT sempre che la sessione persistente sia abilitata per il servizio, altrimenti un'eccezione indicherà che l'autenticazione parziale non è permessa.

6.3.4 Caching

Il caching è una tecnica ampiamente utilizzata per migliorare le prestazioni del server, memorizzando valori di chiamate già utilizzate in una posizione di facile accesso, per possibili riusi in chiamate successive.
I REST service offrono la possibilità di effettuare il caching lato client e lato server.
Il caching lato server viene mantenuto mediante l'utilizzo di dynacache, ovvero un sofisticato tipo di tabella hash, con lo scopo di fornire un servizio di cache in-memory, in grado di recuperare dati sulla base di alcune corrispondenze.
Il servizio opera all'interno di un server applicativo JVM, intercettando le chiamate agli oggetti per memorizzarne il contenuto in cache e renderlo poi disponibile.
Visto che ad ogni richiesta dati è associata una serie di parametri di input, questi vengono combinati per generare una chiave univoca chiamata cache-id .
Per ogni chiamata viene usata cache-id per interrogare la cache, se una richiesta genera una chiave già usata in precedenza la risposta verrà servita dalla cache.

Questa cache viene memorizzata nell'heap della JVM, ma Dynacache permette l'overflow su disco, gestendo gli elementi da spostare con un algoritmo di tipo Least Recently Used (LRU).

Dynacache offre due istanze di cache, una indipendente dall'altra :

L'istanza servlet in cui vengono memorizzzati servlet, JSP, Struts, Tiles, command objects e richieste SOAP; consente ad applicazioni come WebSphere Portal Server e WebSphere Application Server di memorizzare i dati in cache separate.

L'istanza oggetti viene utilizzata per archiviare, distribuire e condividere oggetti Java.

DistribuitedObjectCache e Distribuited Map API vengono fornite per far si che le applicazioni possano interagire con l'istanza cache degli oggetti.

Il caching lato client è gestito mediante l'applicazione client, il server può influenzarne il comportamento utilizzando direttive inserite nell'header di risposta.

Le direttive supportate sono:

Expires che identifica il tempo di validità del campo.

Cache-Control supportata solo quando la risorsa sia pubblica o private è cacheable; caratteristica definita nel file XML: WEB-INF/config/com.ibm.commerce.rest/wc-rest-clientCaching.xml.

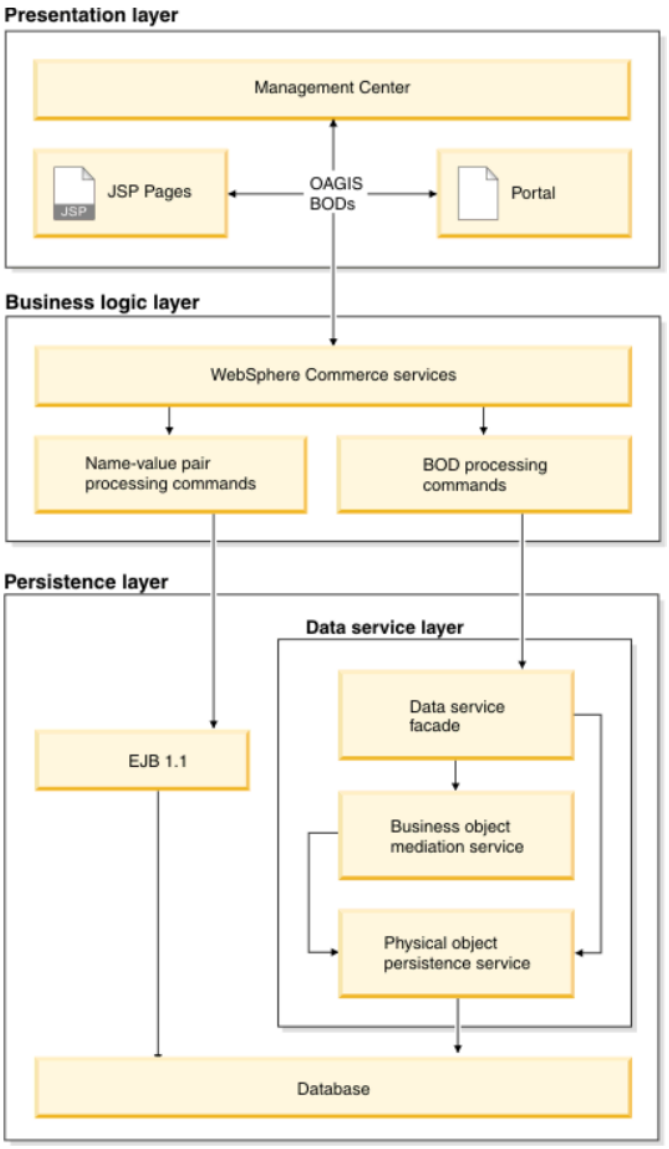

Figura 7: Dettaglio delle interazioni tra il livelli dell'architettura.

6.4 Business Logic Layer

E' il layer che offre la possibilità di eseguire le logiche di business, servizio di recupero dati frapponendosi come intermediario tra il presentation layer e il persistence layer.

E' organizzato in moduli di servizio, che vengono sfruttati dal presentation layer per visualizzare i dati o invocare business process mediante messaggi OAGIS.

I messaggi OAGIS possono essere trasformati in coppie nome-valore, per essere elaborati come tali, in modo da facilitare l'integrazione con versioni precedenti di WebSphere Commerce, o trasformati in oggetti Java di tipo Service Data Object

(SDO), per essere utilizzati dai comandi per il mapping con il Business Object Mediator.

Con questo approccio la business logic e in generale tutto il business logic layer restano svincolati dalla tecnologia di persistenza dei dati utilizzata, infatti questi dialogheranno con il persistence layer utilizzando SDO.

I componenti principali del Business Logic Layer possono essere raggruppati nelle seguenti categorie :

Context Providers: Cercano elementi nella chiamata ai servizi REST come lo storeID, il languageID o l'identità dell'user, in base a ciò creano il contesto appropriato, utilizzato poi per aggiornare o recuperare (BOD); vengono inoltre invocati ad ogni richiesta per fornire al contesto i giusti resource handler.

Resource Handlers: Sono il punto di accesso alle risorse, contengono informazioni utili a gestire le richieste delle risorse, come il percorso, il contesto ,ecc; sono in grado di rappresentare risorse semplici o composte ed effettuano le conversioni tra risposte BOD e il formato richiesto dal client o viceversa.

Helpers: Insieme agli handler astraggono il layer BOD, permettendo l'uso di codice riutilizzabile tra i vari handler.

Esistono vari tipi di helper, da quelli utilizzabili per la configurazione di uno store, quelli per la creazione di URI a quelli per l'applicazione di regole di sicurezza e per la gestione delle risorse.

Data Mappers: File di configurazione personalizzabili, utilizzati nella conversione di risorse da / a BOD.

Entity Providers: Quello presente di default consente la rappresentazione di risposte in XML o JSON, tuttavia ne possono essere aggiunti altri per rappresentazioni dati personalizzate.

Resource Templates: Sistema che permette la resa di rappresentazioni personalizzate come XHTML utilizzando JSP.

6.4.1 Business Object Command Framework

Implementato a partire dal FEP 3.0.1[24], offre la possibilità di elaborare i messaggi OAGIS che vengono utilizzati dalle interfacce per effettuare richieste, recuperare dati o invocare logica di business tenendo slegati tra loro il presentation layer, il logic layer e il persistence layer che nelle versioni precedenti presentavano dipendenze implementative.

Il BOD command framework permette di elaborare messaggi in formato OAGIS, utilizzati come interfaccia per effettuare richieste, recuperare dati o invocare business logic.

Le caratteristiche principali sono:

- BOD command interagisce con service data object (SDO), invece di usare coppie nome-valore(come succedeva prima della sua implementazione).
- BOD possono rappresentare una richiesta complessa, in grado di svolgere molteplici azioni invece di una.
- BOD command interagiscono con un'interfaccia persistente chiamata data service layer, utilizzando un oggetto chiamato Business Object Mediator, sono indipendenti dalla tecnologia di persistenza utilizzata.

6.5 Persistence layer

E' lo strato che ospita i servizi per il recupero dei dati, si occupa quindi di mappare SDO per implementarne persistenza, recupero e aggiornamento.

Le specifiche degli asset di persistenza come possono essere le query SQL sono isolate all'interno di DSL.

Per facilitare l'integrazione con versioni precedenti di WebSphere Commerce questo offre due metodologie di accesso diverse: mediante l'uso di comandi di elaborazione in formato coppia nome-valore (metodo retrocomaptibile con le versioni precedenti di WebSphere Commerce) o impiegando il Business Object

[24] *FEP: Feature Extended Pack pacchetti che aggiungono funzionalità e caratteristiche*

Mediator (BOM), che accetta e restituisce dati in formato di SDO logici (metodo corrente).

6.5.1 Business Object Mediator

Si occupa di trasformare SDO logici in SDO fisici e viceversa, fornendo un'interfaccia verso il data service layer.

BOM supporta operazioni di creazione, lettura, modifica, eliminazione su SDO logici, in tal modo business logic layer può lavorare su questi senza dover accedere direttamente gli schemi fisici.

Gli SDO logici sono il meccanismo con cui i componenti e servizi si scambiano dati, mentre gli SDO fisici sono oggetti Java utilizzati nel persistance layer e rappresentano appunto lo schema fisico; il Business Object Mediator fornisce la mappatura tra gli SDO logici e la loro controparte presente nel persistence layer, ovvero gli SDO fisici.

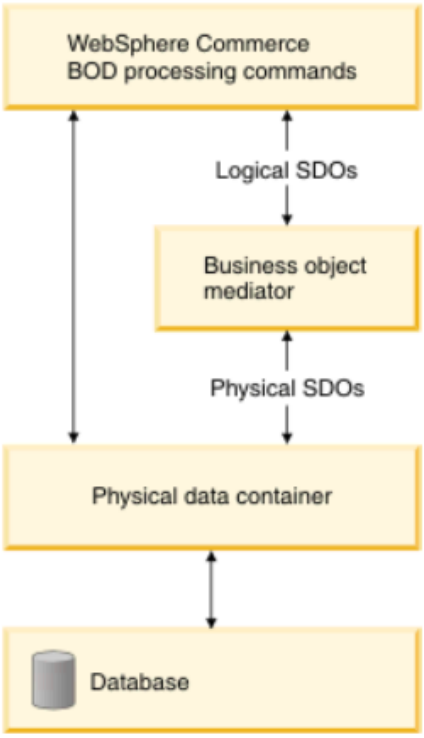

Figura 8: Dettagli delle interazioni con il Physical data container.

Il Business Object Mediator per effettuare operazioni sui dati fisici fa uso dei servizi del Physical Data Container.

Il Physical Data Container gestisce le iterazioni con lo schema fisico e gli SDO fisici, può infatti crearli, memorizzarli in array e salvarli sul DB.

Il Data Service Layer mette a disposizione 2 modi per accedere ai dati fisici : il più comune è quello messo a disposizione mediante Business Object Mediator, tuttavia i Business Object Commands possono aver la necessità di accedere a dati non mappati a livello logico, come possono essere tabelle adibite ai log o alle informazioni di auditing, in questi casi il business logic layer permette di accedere direttamente al DB mediante il Physical Data Container.

6.5.2 I mediator

I mediator sono le classi che permettono le trasformazioni tra rappresentazioni logiche e fisiche, ogni modulo di servizio fornisce la propria implementazione di mediator.

I mediator si possono dividere in 2 tipologie: i mediator di lettura e i mediator di scrittura.

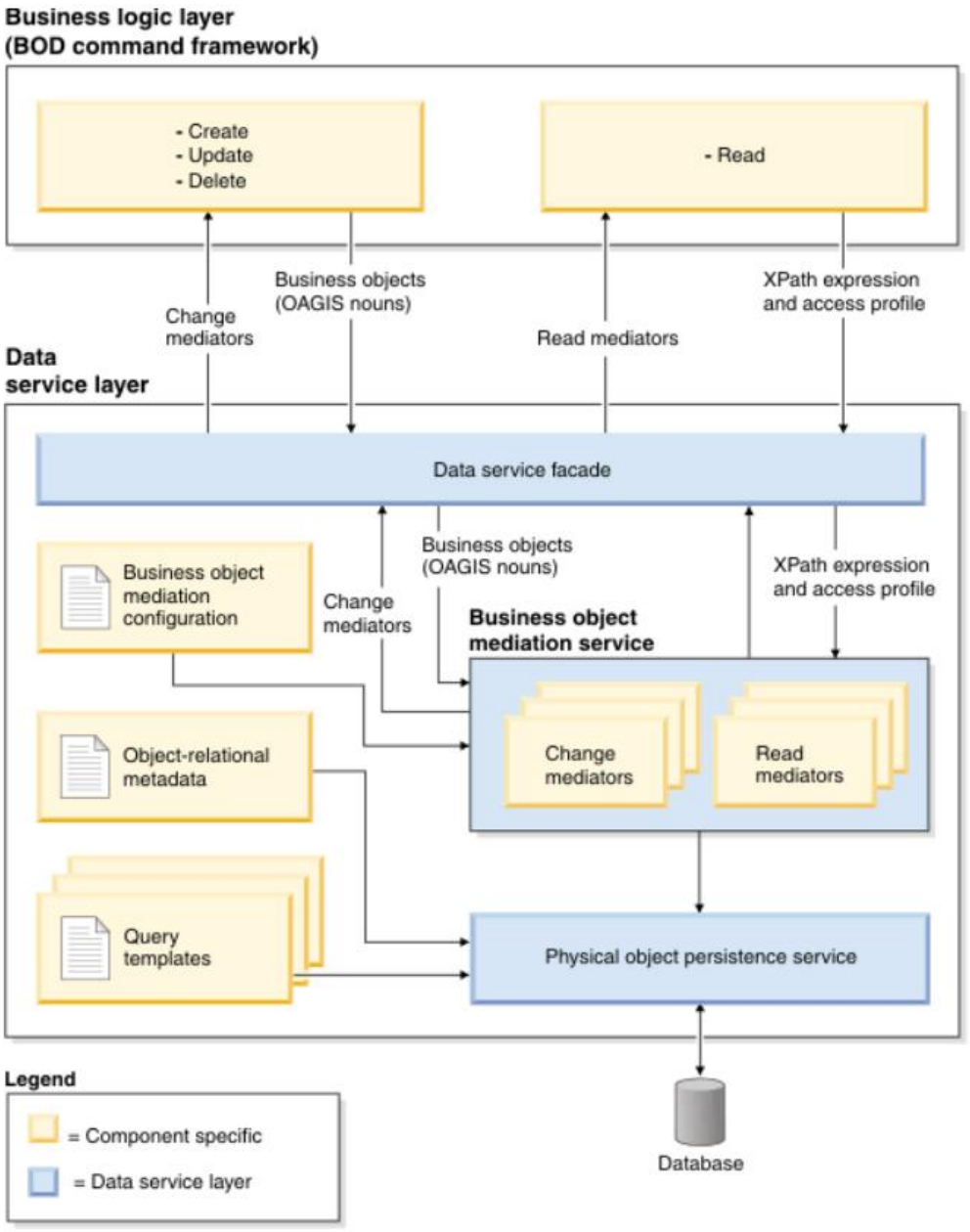

Figura 9: Dettaglio delle interazioni dei mediator e del Data Service Layer.

I mediator di lettura vengono utilizzati per elaborare le richieste OAGIS di tipo GET, da cui il data service layer estrae le query dei dati richiesti dal business logic layer; la query consiste in un espressione XPath[25] e in un profilo di accesso.
Il Data Service Layer ritorna questa query al Business Object Mediator che a sua volta la passa al Persistance Service.
A questo punto vengono generate dalla richiesta una o più istruzioni SQL, una volta eseguite i risultati vengono mappati in SDO fisici e restituiti al Business Object Mediator.
La configurazione di Business Object Mediator descrive come istanziare i mediator necessari, che vengono restituiti al business logic layer.
Da notare che è restituito il mediatore e non solo l'SDO, questo infatti contiene i dati fisici e la logica per convertire l'SDO fisico in SDO logico.
I mediator di scrittura gestiscono l'Oagis Change dei processi e delle richieste di sincronizzazione.
Il Service Data Layer riceve i noun OAGIS e li passa Business Object Mediator service, questo crea poi le istanze appropriate dei change mediator che a loro volta vanno a prelevare la rappresentazione fisica dei sostantivi dal physical object mediation dei service chiamati.
Il Business Object Mediator restituisce poi i mediator al business logic layer; questi sono chiamati con specifiche azioni per creare, aggiornare o cancellare sostantivi e loro parti, che la logica all'interno dei mediator traduce in operazioni su SDO fisici.
Dopo aver effettuato i cambiamenti, il business logic layer istruisce il cambiamento per salvare gli SDO fisici aggiornati.
Il mediator chiama un physical object persistance service per aggiornare il DB.

---

[25] XPath: Linguaggio basato su XML, volto ad identificare i nodi all'interno di un documento, http://www.w3.org/TR/xpath/

6.5.3 Data Service layer

Il Data Service layer è stato introdotto per fornire accesso ai dati indipendente dallo schema fisico, mediante un'interfaccia consistente chiamata data Service Facade, rimanendo indipendente dal framework che effettua la mappatura oggetto/entità relazionale.

Il framework per astrarre il mapping è usato per trasformare i dati ritornati dal database in una collezione di oggetti Java, SDO fisici, su cui data service layer lavora, operando una trasformazione bidirezionale tra SDO fisici e SDO logic, permettendo di eseguire operazioni CRUD[26] partendo dagli SDO logici.

In alternativa il data service layer permette di eseguire operazioni CRUD direttamente sui dati fisici, scavalcando cosi lo schema logico.

Il data service layer è composto da tre parti: Business object mediation service, physical persistance service, data service facade.

Il punto d'ingresso in DSL è Data Service Facade, che fornisce le interfacce per lavorare con dati fisici e logici.

Per le operazioni di lettura il data service facade riceve una query dal business logic layer, consistente in un'espressione XPath e un profilo di accesso ricavati dal verbo OAGIS GET; la query è passata al BOM che a sua volta la passa al persistance service.

Il persistance service genere le query corrispondenti alle operazioni richieste utilizzando dei template che mappano le query XPath in gruppi di query SQL.

Dopo l'interrogazione del DB viene ritornato l'SDO fisico al BOM che contiene la configurazione per istanziare i mediator necessari, che verranno poi ritornati come risultato al business logic layer.

Per le operazioni che richiedono un cambiamento dei dati invece, il data service facade riceve i sostantivi OAGIS come input, e li passa al business object mediation services che istanzia gli appropriati change mediator per caricare la

[26] CRUD: Acronimo usato per riassumere le 4 operazioni di Create, Read, Update, Delete.

rappresentazione fisica dei sostantivi dal physical object mediation service, poi il BOM ritorna i mediatori al business logic layer.
I mediator sono chiamati con specifiche azioni sui sostantivi o su loro parti, queste operazioni sono tradotte dalla logica all'interno dei mediator in operazioni su SDO fisici.
Successivamente alle modifiche il business logic layer istruisce i change noun mediator per salvare l'SDO fisico aggiornato, in fine il mediator chiama il physical object persistence service per aggiornare il database.

6.5.4 Physical Data Container

Il Physical Data Container supporta operazioni di creazione, lettura, aggiornamento, cancellazione, su SDO fisici mappati direttamente sullo schema fisico.
Esso permette al business logic o al client di lavorare cho SDO fisici se questi non sono mappati come modello logico.

6.6 WebSphere Commerce Search

Fornisce funzionalità di ricerca nello starter store, utilizzando un motore di ricerca esterno corredato di funzioni come: suggerimento di termini, speling correction, ecc.
Le runtime di ricerca possono essere invocate dallo storefront direttamente attraverso servizi REST.
Con questo aproccio il traffico destinato alla ricerca è passato dal server di WebSphere a quello di ricerca.
webSphere Commerce search è basato su indici di ricerca, dove un'indice è rappresentato da una grande tabella flessibile che contiene campi di ricerca, ottimizzata per le performance della ricerca.
Gli indici di ricerca devono essere costruiti prima di poter eseguire qualsisi ricerca.
Gli indici di ricerca sono costruiti su uno o più documenti , dove un documento contiene campi.

Un campo consiste in un nome, un contenuto, dei metadati che indicano la corretta gestione a WebSphere Commerce.

WebShere Commerce search usa uno schema di rappresentazione simile ad uno schema di DB.

Lo schema di ricerca definisce la struttura degli indici di ricerca.

6.6.1 Architettura del Search server

Il server consiste in un gruppo di servizi REST, un search runtime framework e un WebSphere Commerce foundation service incaricato di fornire l'accesso al database.

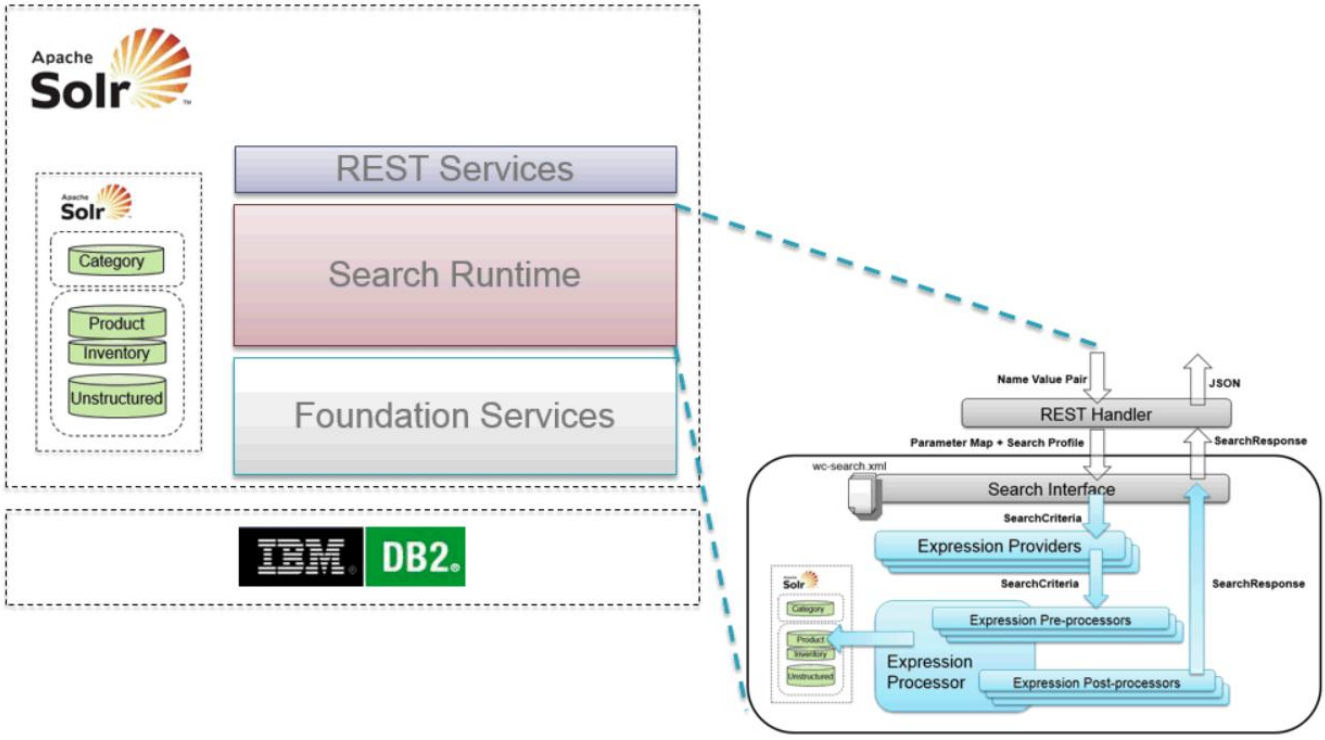

Figura 10: Layer del search server.

Il REST Service è un'applicazione Apache Wink, che mappa l'URL REST e il corrispondente resource handler che una volta interpellato chiamerà il search runtime.

Il foundation service è il nucleo in grado di fornire servizi come query JDBC, usato per accedere al DB di WebSphere Commerce.

Il search runtime framework è guidato principalmente da un programming pattern simili ad una catena di montaggio, dove ogni componente di questa contribuisce con il proprio pezzo di espressione alla query.

Il runtime di ricerca è un framework di programmazione estensibile, che permette l'esecuzione di espressioni di ricerca utilizzando proprietà definite in un profilo di ricerca.

Una volta che 'espressione di ricerca è composta viene inoltrata al Solr runtime embedded per l'elaborazione.

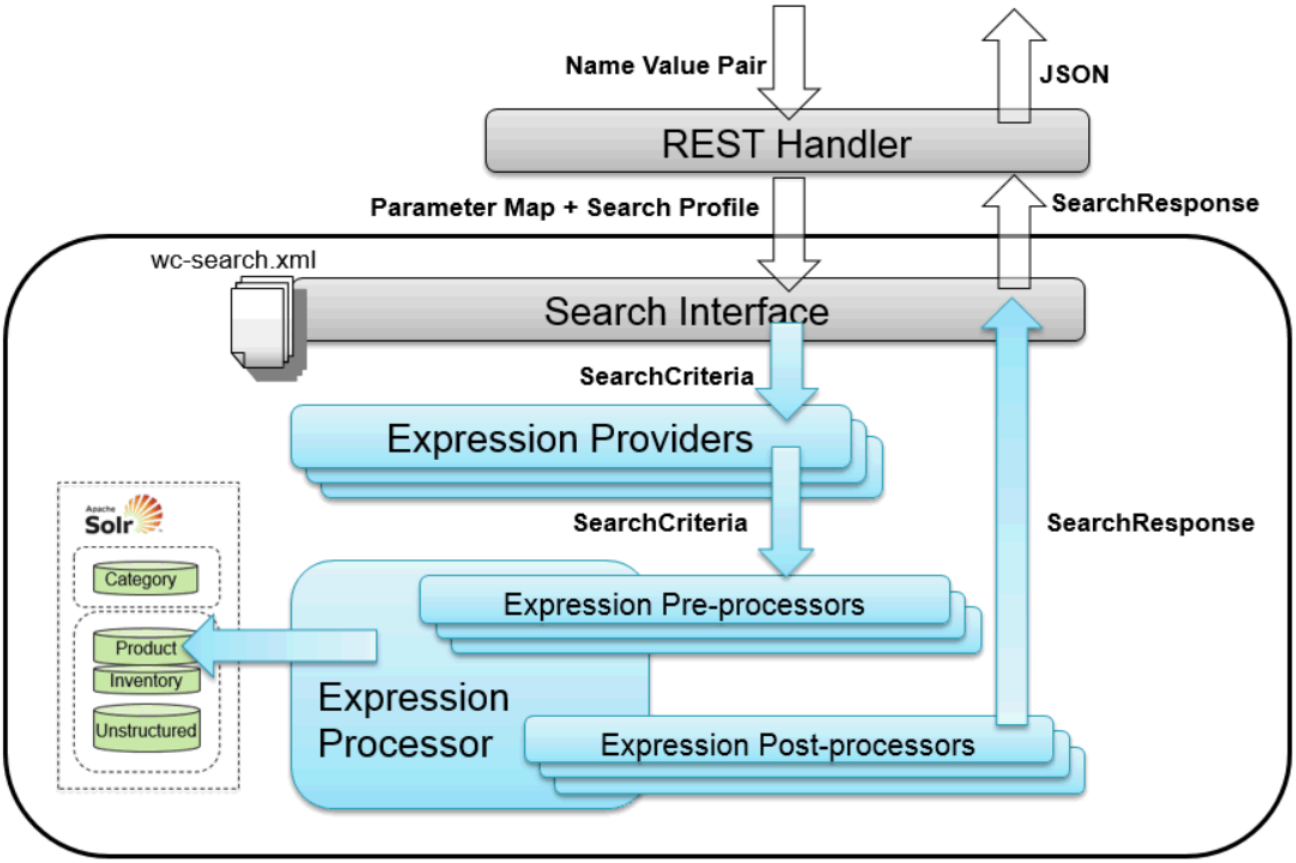

Figura 11: Vista del search runtime.

Il search runtime contiene due componenti principali:

Il Search Expression Provider dipende dalla natura della richiesta,che può essere: profili di ricerca, componenti business coinvolti; come ad esempio: marketing per ricerche basate su regole di merchandising o contratti per il diritto.

Ogni componente business incaricato di contribuire ad una porzione di espressione di ricerca viene combinato con l'espressione principale generata dal servizio di navigazione catalogo.

Successivamente l'espressione di ricerca viene eseguita dal search processor.

Una factory class SolrRESTSearchExpressionProvider gestisce tutti i fornitori di espressioni in fase di esecuzione.

L'implementazione di questa factory class è definita nel file SearchExpressonProviderFactory.properties sul search server.

SolrRESTSearchExpressionProvider esegue di default le seguenti operazione in ordine:

1. Convalida profilo di ricerca chiamando SolrSearchProfileNameValidator.
2. Convalida del corrispondente nome del'indice chiamando SolrSearchIndexNameValidator.
3. Convalida delle informazioni del workspace, chiamando SolrSearchWorkSpaceValidator che le mette a disposizione del processore.
4. Convalida dell'espressione di ricerca al fine di garantire che la query non sia vuota e non contenga caratteri speciali chiamando SolrSearchExpressionValidator.
5. Avvia una lista di query expression providers definite nel profilo di ricerca dove ogni provider contribuisce ad una parte dell'espressione di ricerca.

Search Expression Processor: Un'unità centrale per l'integrazione con il motore di ricerca. La sua responsabiltià è far girare le espressioni date a Solr ,sulla base delle ricerche fornite dal profilo di ricerca e catturare la risposta del motore di ricerca.

Il SolrRESTSearchExpressionProcessor è un'implementazione di default dell'expression processor che viene definita nel file SearchExpressionProcessorFactory.properties sul server di ricerca.

Il SolrRESTSearchExpressionProcessor predefinito compone l'espressione finale Solr e imposta tutti i parametri bootstrap necessari per l'esecuzione della ricerca runtime Solr.

Il set di risultati viene riformattato in un oggetto JSON da restituire al chiamante.

l'expression processor esegue le seguenti operazioni nel seguente ordine:

1. inserisce un parametro opzione di debug nell'oggetto SolrQuery chiamando SolrSearchQueryPreprocessor.
2. Genera un'elenco di campi indice da includere nel risultato di ricerca impostato dalla chiamata SolrSearchResultFieldQueryPreprocessor.
3. Include un punteggio di rilevanza e ulteriori informazioni di analisi nell'oggetto SolrQuery con la chiamata SolrSearchPreviewQueryPreprocessor.
4. Abilita spell correction e include le sue opzioni dei parametri associati nell'oggetto SolrQuery chiamando SolrSearchSpellCorrectionQueryPreprocessor.
5. Abilita l'evidenziazione e include le sue opzioni dei parametri associati nell'oggetto SolrQuery chiamando SolrSearchHighlighterQueryPreprocessor.
6. inserisce le opzioni dei parametri di impaginazione nell'oggetto SolrQuery chiamando SolrSearchPaginationQueryPreprocessor.
7. Inserisce l'ordinamento delle opzioni dei parametri nell'ogetto SolrQuery chiamando SolrSearchSortingQueryPreprocessor.
8. Compone un elenco di campi facet ,insieme con le impostazioni corrette da inserire nel set dei risultati di ricerca chiamando SolrSearchFacetQueryPreprocessor.
9. Inserisce la query dismax correlata chiamando SolrsearchEDismaxQueryPreProcessor come eDismax handler (deve avvenire prima della query principale poichè il parametro di controllo della query inserita potrebbe dover essere rimosso).

10. Inserisce il parametro opzione formato di risposta chiamando SolrSearchResponseFormatQueryPreprocessor.
11. Inserisce la stringa principale della query di ricerca nell'oggetto SolrQuery chiamando SolrSearchMainueryPreprocessor.
12. Inserisce ulteriori parametri personalizzati SolrQuery nell'espressione finale della query Solr, chiamando SolrSearchCustomQueryPreprocessor.
13. Avvia un elenco di search quey preprocessor, definite nel profilo di ricerca corrente, per permettere le modifiche all'oggetto SolrQuery, prima di inviarlo a Solr per l'elaborazione.
14. Invia una richiesta a Solr, per elaborare l'oggetto SolrQuery.
15. Avvia una lista di search query postprocessor definiti nel profilo di ricerca corrente, per permettere le modifiche ai dati fisici dell'oggetto SearchResponse; subito dopo il QueryResponse viene restituito dal server Solr.

6.6.2 Apache Solr

Solr è motore di ricerca open source basato sul progetto Apache Lucene, tra le caratteristiche di maggior rilievo includono un motore di ricerca testuale, indicizzazione in tempo reale, dynamic clustering, integrazione database, ecc.
E' scritto in Java e si appoggia alle librerie di ricerca Lucene che forniscono funzionalità di indicizzazione e ricerca testuale, viene eseguito come server di ricerca testi stand-alone all'interno di un servlet container, questo costruisce degli indici di ricerca su campi mappati a priori sulla sorgente, velocizzando il reperimento di dati, usa poi interazioni basate richieste REST Http /XML e JSON, che lo rendono facile da interrogare da qualsiasi linguaggio o ambiente.
Tra le carateristiche di maggior rilievo si hanno:

- Funzionalità di ricerca full-text avanzata.
- Ottimizato per un alto traffico Web.
- Interfacce basate su standard XML, JSON e HTTP.

- Statistiche server esposte su JMX per il monitoraggio.
- Linearmente scalabile, replica automatica degli indici, rilevamento guasti e recovery.
- Indicizzazione real-time.
- Configurazione XML flessibile e adattabile
- architettura estensibile con plug-in.

Capitolo 7: Personalizzazione di un servizio esistente

7.1 Introduzione

I capitoli precedenti hanno illustrato lo stile architetturale SOA realizzato mediante web services e come questo viene implementato dalla piattaforma IBM WebSphere Commerce; si sono introdotti i servizi REST esposti dalla piattaforma e si è analizzato nel dettaglio come una richiesta REST venga processata dai diversi strati di logica implementati nella WCS.

Quanto scritto nei capitoli precedenti e brevemente riassunto qui sopra rappresenta il percorso di studio e di analisi condotto in TECLA e preparatorio alla trattazione ben più tecnica che viene presentata in questo capitolo finale.

Quanto riportato in questo capitolo, cuore dell'esperienza maturata durante il tirocinio in TECLA, nasce da un'esplicita esigenza aziendale derivante da richieste maturate sui progetti di alcuni clienti e si inserisce nel ciclo di lavoro tipico dell'azienda stessa ovvero la customizzazione di una funzionalità standard della piattaforma IBM. Il caso particolare è l'analisi, lo studio e la realizzazione dell'estensione di uno dei servizi REST esposti dalla piattaforma in modo da venire incontro alle particolari esigenze applicative di alcuni clienti.

Essendo una suite destinata all'e-commerce l'asset standard di IBM WebSphere Commerce presenta servizi RESTles orientati all'uso di risorse commerciali, come la gestione del carrello, dei coupon, dei codici promozionali, degli ordini, dei pagamenti, ecc.

Particolare attenzione in questa sezione è posta sulle richieste dei dettagli prodotti, che possono essere fatte usando come chiave di ricerca un termine, ID del prodotto, ID della categoria, ecc.

L'oggetto JSON 'CatalogEntryView' è l'oggetto inaricato di rappresentare un'articolo commerciale, quando questo è restituito come risposta ad una richiesta REST fatta

mediante i servizi che il capitolo si pone di personalizzare, aggiungendo agli attributi già presenti 'field1',valore numerico presente solo nella corrispondente tabela del DB, per essere utilizzato dalle logiche di frontend.
La modifica è stata svolta con la supervisione del tutor aziendale e del supporto tecnico IBM, ha richiesto un intervento non solo sul presentation layer ma anche su business e persistence layer.
La procedura esposta inizierà ad implementare le modifiche partendo da Solr, modificando le impostazioni della mappatura e dell'indicizzazione, poi verrà esteso il profilo di ricerca, la mappatura della sogente dell'oggetto, la configurazione del business object mediator e in fine la mappatura dell'URL; adottando quindi un approccio bottom-up.
L'approccio bottom-up oltre a garantire la possibilità di verificare gli esiti dei passaggi, permette di mantenere coerenza tra le varie nomenclature utilizzate nelle mappature tra la risorsa REST e la colonna del DB.

7.2 Solr e il preprocessore

Il primo passo consiste nel modificare i file di configurazione del preprocessore di Solr, questo si occupa di estrarre i dati da una sorgente esterna, "appiattirli" e metterli in una tabella provvisoria da cui Solr attingerà per creare l'indice.
In caso si debba eseguire la procedura su una tabella da creare, è necessario costruire anche un file: "wc-dataimport-preprocess-nometabella.xml" nella cartella: "C:\IBM\WCDE_ENT70\search\pre-processConfig\MC_catalogID\Cloudscape\", questo specifica le definizioni di tabelle, i metadati ed i riferimenti ai dati presenti nel Db che devono essere trasferiti sulle tabelle temporanee; nel caso preso in esame il campo 'field1' si trova nella tabella 'catentry', già presente all'interno del db, quindi questo passaggio non è necessario.
Il file: 'wc-data-config.xml' presente in:
'C:\IBM\WCDE_ENT70\search\solr\home\MC_10001\en_US\CatalogEntry\conf\',

contiene le query che prendono i dati dalle varie tabelle del db e le inseriscono in altre tabelle temporanee con chiave primaria CATENTRY.CATENTRY_ID, effettuando per ottenere questo risultato molteplici operazioni di join tra le varie provenienze; qui si aggiunge, per tutte le query presenti nel file, in corrispondenza di select la voce CATENTRY.FIELD1.

Il file wc-data-config.xml contiene anche i nomi delle variabili con cui verranno identificati i vari valori risultanti dalla query; nel caso dell'esempio si avrà: ' <field column="FIELD1" name ="field1"/>', dove column è il nome di una delle colonne risultanti dalla query, a cui ci si riferisce, name è il nome della colonna di destinazione; necessita attenzione l'elenco dei campi <field>, che devono seguire lo stesso ordine di dichiarazione usato per elencare le colonne nella select della query corrispondente.

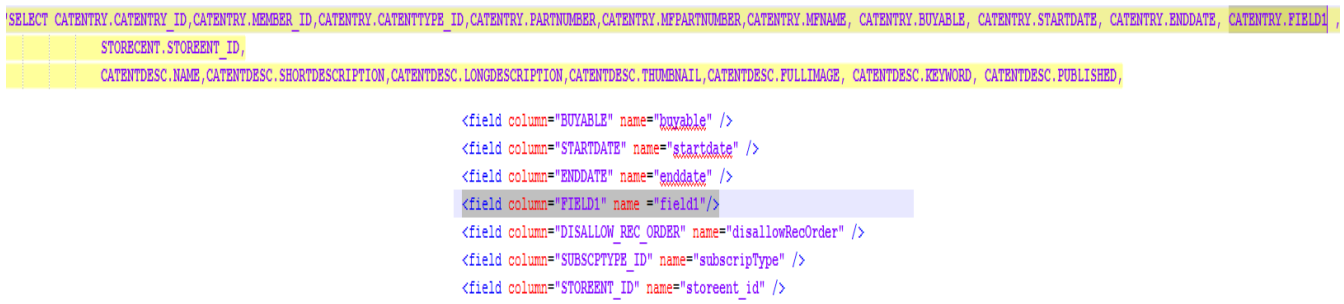

Figura 12: Screenshot su una query del file wc-data-config.xml che evidenzia la relazione tra l'ordine di dichiarazione dei campi della select e i tag field.

Il file schema.xml presente in: 'C:\IBM\WCDE_ENT70\search\solr\home\MC_10001\en_US\CatalogEntry\conf\schema.xml', si occupa di tenere traccia dei tipi di dato utilizzati da Solr per rappresentare i dati indicizzati e le impostazioni specifiche per ognuno di questi.

La prima sezione del file indica quali tipi di dato sono utili per rappresentare i dati presenti nella corrente indicizzazione, presentando campi in formato tipo: '<fieldType name="int" class="solr.TrieIntField" precisionStep="0" omitNorms="true" positionIncrementGap="0"/>', dove:

name: indica il nome del tipo di dato da utilizzare per referenziarlo.

class: Indica la classe da utilizzare per memorizzare il tipo di dato in questione.

precisionStep: Indica la precisione del valore.

omitNorms: Disabilita le norme associate al campo.

positionIncrementGap: nei campi multivalore specifica quanto questi siano distanti tra loro.

E' possibile specificare anche tag come <analyzer>, <tokenizer>, <filter> che permettono di pulire un campo; per esempio da tag Html.

La seconda sezione del file definisce le impostazioni specifiche per ogni tipo di campo, qui si inserirà tra le voci già presenti: ' <field name="field1" type="int" indexed="true" stored="true" multiValued="false" />', che definisce le impostazioni e le caratteristiche da utilizzare per indicizzare field1.

```
<!--
Catentry's basic attributes: map to table CATENTRY
-->
<field name="catentry_id" type="string" indexed="true" stored="true" required="true" multiValued="false"/>
<field name="member_id" type="long" indexed="true" stored="true" multiValued="false"/>
<field name="mfName" type="wc_text" indexed="true" stored="true"  multiValued="false"/>
<field name="buyable" type="int" indexed="true" stored="true" multiValued="false"/>

<field name="partNumber_ntk" type="wc_keywordTextLowerCase" indexed="true" stored="true"  multiValued="false"/>
<field name="mfPartNumber_ntk" type="wc_keywordTextLowerCase" indexed="true" stored="true" multiValued="false"/>
<field name="mfName_ntk_cs" type="wc_keywordText" indexed="true" stored="true" multiValued="false" />
<field name="mfName_ntk" type="wc_keywordTextLowerCase" indexed="true" stored="true" multiValued="false" />
<field name="catenttype_id_ntk_cs" type="wc_keywordText" indexed="true" stored="true"  multiValued="false"/>
<field name="startdate" type="date" indexed="true" stored="true"  multiValued="false"/>
<field name="enddate" type="date" indexed="true" stored="true"  multiValued="false"/>
<field name="field1" type="int" indexed="true" stored="true" multiValued="false" />
```

Figura 13: Sezione del file Schema.xml estesa.

Dove gli attribuiti del tag field specificano:

name : Il nome del campo specificato in precedenza.

Type: il tipo di dato che deve essere restituito (Nota: all'inizio di questo file possono essere specificati dei tipi di dato personalizzati).

indexed=true|false: Da impostare a true se il campo deve essere indicizzato ; in caso affermativo questo potrà essere visibile tramite ricerca o ordinato.

stored=true|false: Se impostato a true il valore del campo è recuperabile durante una ricerca utilizzando la funzione MoreLikeThis.

multiValued=true|false: Se impostato a true questo campo contiene valori multipli nel documento e può quindi apparire più volte.

La terza sezione del file copia le variabili definite e abilitate ad essere indicizzate nel campo defaultSearch, che è l'oggetto interessato dalle ricerche scaturite mediante query generiche (quelle in cui non è specificato esplicitamente il tipo di campo su cui eseguire la ricerca).

Viene aggiunto a quelli già presenti <copyField source="field1" dest="defaultSearch">, volto ad indicare a Solr che si vogliono copiare tutti i dati presenti nel campo specificato dall'attributo source, al campo specificato dall'attributo dest.

Es:

```
<!-- Copy fields for default search field -->
<copyField source="name" dest="defaultSearch"/>
<copyField source="shortDescription" dest="defaultSearch"/>
<copyField source="partNumber_ntk" dest="defaultSearch"/>
<copyField source="keyword" dest="defaultSearch"/>
<copyField source="cas_f*" dest="defaultSearch"/>
<copyField source="cai_f*" dest="defaultSearch"/>
<copyField source="caf_f*" dest="defaultSearch"/>
<copyField source="nameOverride" dest="defaultSearch"/>
<copyField source="shortDescriptionOverride" dest="defaultSearch"/>
<copyField source="keywordOverride" dest="defaultSearch"/>
<copyField source="categoryname" dest="defaultSearch"/>
<copyField source="field1" dest="defaultSearch"/>
<!--copyField source="Field*" dest="defaultSearch"/-->
```

7.2.1 Le utility di Solr

Per applicare le modifiche viste è necessario ora lanciare 2 utility da riga di comando, con i diritti di amministratore.

La prima è 'di-preprocess.bat', presente in 'C:\IBM\WCDE_ENT70\bin\', che si occupa di estrarre i dati dal db e creare le tabelle provvisorie, va lanciata con il server di WebSphere arrestato e la sintassi per la sua invocazione è 'di-preprocess.bat C:\IBM\WCDE_ENT70\search\pre-processConfig\MC_catalogID\Cloudscape'; il percorso dato come parametro segnala all'utility dove si trovano i file del preprocessore che indicano le tabelle e i metadati del DB.

La seconda è di-buildindex.bat, il cui scopo è ricreare l'indice di ricerca di Solr, partendo dalle tabelle provvisorie create, va lanciata col server di WebSphere attivo e con la sintassi: ‘di-buildindex.bat -masterCatalogId catalogID’; l'utility ricrea da zero gli indici o li aggiorna tramite le query delta(tipologia di query presente nel file wc-data-config.xml specializzate nell’aggiornamento), sfrutta le tabelle temporanee create dal preprocessore per creare gli indici e per connettersi al database di WebSphere Commerce utilizza il servizio Data Import Handler e una connessione JDBC[27].

Arrivati a questo punto è possibile verificare se gli step eseguiti sono andati a buon fine, interrogando direttamente Solr mediante l'accesso diretto fornito dal server di ricerca utilizzando il browser.

Tramite URL va inserita la query: ' http://localhost/solr/MC_10001_CatalogEntry_en_US/select?q=field1:*', dove MC_ è seguito dal catalog Id, la tabella su cui eseguire la ricerca, la lingua correntemente utilizzata ed infine il corpo della query che chiede al search server di mostrare tutti i campi con field1.

Se la procedura è stata eseguita correttamente verrà restituito dal browser un documento xml in cui sono presenti gli oggetti della tabella catalogo con il campo appena mappato simile al seguente:

[27] *JDBC: Java DataBase Connectivity, http://it.wikipedia.org/wiki/JDBC*

Figura 14: Screenshot dell'output di una query elaborata da Solr, si è evidenziata la presenza di field1.

7.3 Profili di ricerca

WebSphere Commerce usa i profili di ricerca per influenzare la ricerca a livello di pagina.

I profili di ricerca sono dei gruppi di impostazioni che coinvolgono i nomi degli indici, i campi di ricerca dell'indice, la paginazione, l'ordinamento e le varie caratteristiche estetiche quali sottolineatura, aspetto, correzioni del testo.

I profili vengono definiti nel file wc-search.xml, che adotta la seguente struttura:

<server>: Sezione dedicata ai parametri di connesione del server di ricerca.

<AdvancedConfiguration>: Questa impostazione usa Apache Common HTTP client per connettere al server remoto di ricerca secondo i seguenti campi:

<URL>: Indirizzo del server radice di Solr.

<soTimeout>: Valore espresso in millisecondi, indica il tempo di attesa prima del timeout che il client utilizza mentre aspetta i dati.

<connectionTimeout>: Tempo espresso in millisecondi da attendere durante lo stabilimento di una connessione col client, prima di decretare il timeout.

<defaultMaxConnectionsPerHost>: il massimo numero di connessioni permesse ad un host.

<maxTotalConnections>: Numero massimo di connessioni permesse.

<maxRetries>: Numero di tentativi di connessioni possibili prima di decretare il fallimento dell'operazione.

<retryTimeInterval>: Tempo espresso in millisecondi, da attendere tra ogni tentativo di connessione.

<followRedirects>: indica se il reindirizzamento HTTP automatco è abilitato o meno.

<allowCompression>: Se abilitato il client è in grado di fare uso migliore della banda a sua disposizione.

<BasicConfiguration>: Questa impostazione fornisce la stessa interfaccia di un server di ricerca integrato a livello locale, senza richiedere una connessione HTTP.

<Index>: proprietà associate all'indice di ricerca.

<object>: La classe java utilizzata per rappresentare l'oggetto nativo di risposta della ricerca.

<deltaUpdate>, <fullBuild>: Permettono di impostare il trigger di sincronizzazione del'indice dalla vetrina:

deltaUpdate: indica se lo storefront può attivare la costruzione del indice delta (aggiornamento dell'indice).

fulBuild: indica se lo storefront può attivare la costruzione completa dell'indice (ricostruisce totalmente l'indice).

<Cores>: Elenco di impostazioni di base del server di ricerca.

<Profiles>: Gruppi di impostazioni per il runtime di ricerca, come il nome dell'indice di ricerca, i campi degli indici di ricerca, i fornitori di espressioni ecc. Da questi si può cambiare l'esperienza di ricerca a livello di pagina.

Per estendere il profilo nell'esempio corrente, si raggiungerà il file wc-search.xml presente nella cartella WC/xml/config/com.ibm.commerce.catalog-ext/ e si inserirà nel file:

```
<_config:profile name="X_findCatalogEntry"
        extends="IBM_findCatalogEntryAll">
        <_config:result>
        <_config:field name="field1" />
        </_config:result>
</_config:profile>

<_config:profile name="IBM_fetchRelatedCatalogEntryDetailedInfo"
        extends="IBM_fetchRelatedCatalogEntrySummaryInfo">
        <_config:result inherits="true">
        <_config:field name="field1" />
        </_config:result>
</_config:profile>
```

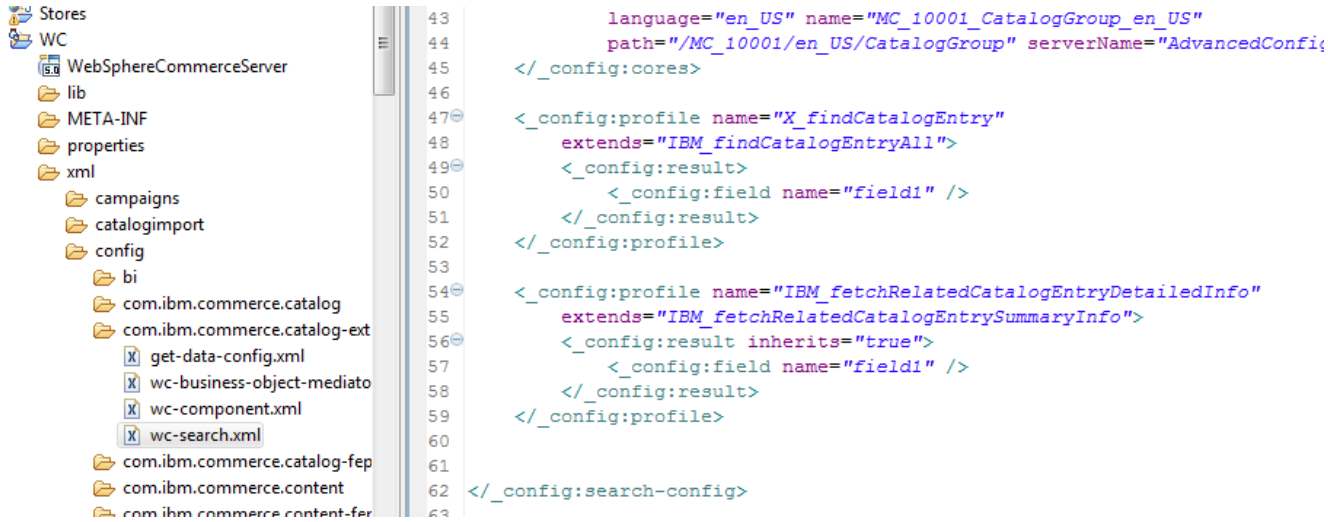

Figura 15: Screeshot su WebSphere, che mostra la sezione e il percorso del file, in cui è stato aggiunto il frammento XML.

In modo da specificare due nuovi profili di ricerca, x_findCatalogEntry che estende il profilo IBM_findCatalogEntryAll e IBM_fetchRelatedCatalogEntryDetailedInfo che estende IBM_fetchRelatedCatalogEntrySummaryInfo.
Il file 'WC/xml/config/com.ibm.commerce.catalog-ext/wc-search.xml' potrebbe essere da creare; infatti le cartelle con suffisso -ext nel nome denotano caratteristiche aggiuntive da sommare alle corrispondenti senza suffisso, di default le cartelle con suffisso '–ext' non sono presenti.

7.4 Mappatura

Il file 'wc-component.xml' è incaricato di conservare la mappatura del percorso dell'oggetto alla sua origine; per esempio potrebbe essere la colonna della tabella nel Db.

Il file si trova in WC /xml/config/com.ibm.commerce.catalog-ext/, qui si aggiungerà:

```
<?xml version="1.0" encoding="UTF-8"?>
<_config:DevelopmentComponentConfiguration
xmlns:_config="http://www.ibm.com/xmlns/prod/commerce/foundation/config"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ibm.com/xmlns/prod/commerce/foundation/confi
g ../xsd/wc-component.xsd ">
   <_config:extendedconfiguration>
      <_config:configgrouping name="AccessProfileToSearchProfileMapping">
         <_config:property name="IBM_Store_CatalogEntrySearch"
value="X_findCatalogEntry"/>
      </_config:configgrouping>
   </_config:extendedconfiguration>
</_config:DevelopmentComponentConfiguration>
```

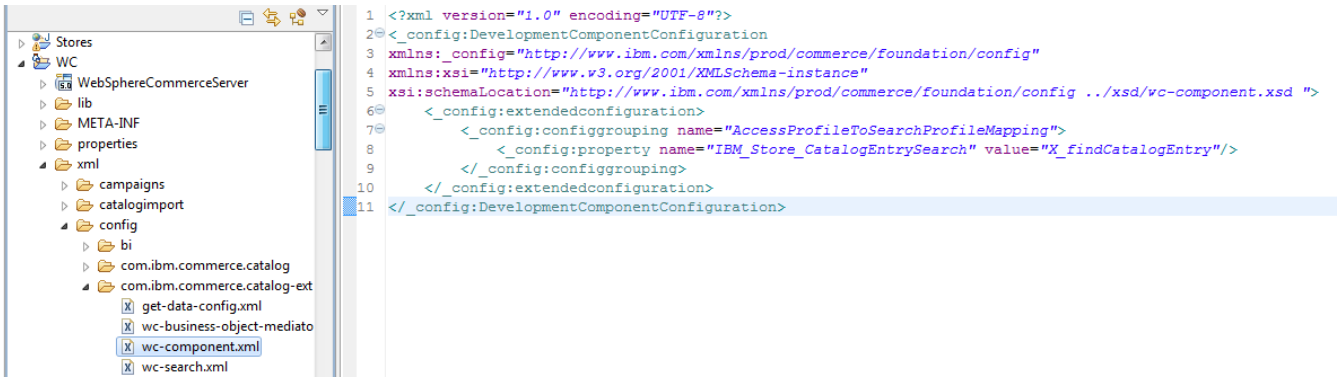

Figura 16: Screeshot su WebSphere, che mostra la sezione e il percorso del file wc-component.xml, in cui è stato aggiunto il frammento XML.

7.5 Estendere la configurazione del Business Object Mediator

Come già descritto il Business Object Mediator opera una trasformazione bidirezionale tra SDO fisici e SDO logici.

Per adempire a questo incarico fornisce un'interfaccia verso il Data Service Layer.

Il file di configurazione wc-business-object-mediator.xml offre le seguenti configurazioni: mapping tra i noun logici ed il corrispondente tipo di dato fisico personalizzato, facilitando il flusso di dati personalizzati, memorizzati sul database del servizio; espone le proprietà logiche come coppia nome valore nell'elemento UserData e istruisce il Business object Mediator all'utilizzo dei metodi appropriati della classe Java dell'SDO fisico per aggiungere informazioni dall'user data element al modello logico.

In caso si voglia estendere un sostantivo, sarà necessario modificare il mapping tra la classe utilizzata per rappresentare l'oggetto logico e quella utilizzata per l'oggetto fisico.

Per la modifica proposta dall'esempio del capitolo si aggiungerà al file di estensione della configurazione 'WC_eardir/xml/config/com.ibm.commerce.catalog-ext/wc-business-object-mediator.xml' il seguente contenuto:

```
<?xml version="1.0" encoding="UTF-8"?>
        <_config:BusinessObjectMediatorConfiguration
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://www.ibm.com/xmlns/prod/commerce/foundation/config ../xsd/wc-business-object-mediator.xsd" xmlns:_config="http://www.ibm.com/xmlns/prod/commerce/foundation/config">
```

```
                <_config:object logicalType="com.ibm.commerce.catalog.facade.datatypes.CatalogNavigationViewType"

        physicalType="com.ibm.commerce.catalog.facade.server.services.search.metadata.solr.SolrCatalogNavigationViewImpl">
                        <_config:mediator
interfaceName="com.ibm.commerce.foundation.server.services.dataaccess.bom.mediator.ReadBusinessObjectMediator"

        className="com.ibm.commerce.catalog.facade.server.services.dataaccess.bom.mediator.solr.SolrReadCatalogNavigation
ViewMediator">
                        </_config:mediator>
                </_config:object>

                 <_config:object logicalType="com.ibm.commerce.catalog.facade.datatypes.CatalogNavigationViewType"

        physicalType="com.ibm.commerce.catalog.facade.server.services.search.metadata.solr.SolrCatalogNavigationViewImpl">
                <_config:mediator
interfaceName="com.ibm.commerce.foundation.server.services.dataaccess.bom.mediator.ReadBusinessObjectMediator"

        className="com.ibm.commerce.catalog.facade.server.services.dataaccess.bom.mediator.solr.SolrReadCatalogNavigation
ViewMediator">
                        <_config:mediator-properties>
                        <_config:mediator-property name="CatalogEntryView/UserData[(Name='field1')]" value="field1" />
                        </_config:mediator-properties>
                </_config:mediator>
        </_config:object>

        </_config:BusinessObjectMediatorConfiguration>
```

7.6 Mappare l'URL

Il file ‘wc-rest-resourceconfig.xml’ aggrega le varie modifiche fatte fin'ora; al suo interno infatti si trovano le definizioni dei servizi esposti con l'url che si vuole utilizzare per tornare il dato, la descrizione, il profilo di accesso e il profilo di ricerca.

Nel file ‘REST/WebContent/WEB-INF/config/com.ibm.commerce.rest-ext/wc-rest-resourceconfig.xml’ inserire:

```
<ResourceConfig>
   <Resource name="productview">
      <GetUri uri="store/{storeId}/productview/byId/{productId}" description="Get
product by unique ID"
        accessProfile="IBM_Store_CatalogEntrySearch"
searchProfile="X_findCatalogEntry"/>
   </Resource>
</ResourceConfig>
```

7.7 Verifica del risultato

Per verificare il risultato si può interrogare l'application server direttamente con una chiamata REST volta consultare i dettagli di un articolo.

Si può, come nell'esempio, chiedere al server i dettagli relativi ad un prodotto specificato dall'id:

http://localhost/wcs/resources/store/10151/productview/byId/10125, dove 10125 è l'Id del prodotto di cui si intende visualizzare i dettagli.

Il server ritornerà un oggetto Json come il seguente:


```
{
	"CatalogEntryView": [
		{
			"MerchandisingAssociations": [
				{
					"name": "T545 15.1\" WHITE HYBRID FLAT PANEL MONITOR",
					"partNumber": "6653HW2",
					"quantity": "1.0",
					"shortDescription": "T545 15.1\" WHITE HYBRID FLAT PANEL MONITOR",
					"thumbnail": "\/wcsstore\/CAS\/images\/F_T545_white_sm.gif",
					"type": "REPLACEMENT",
					"uniqueID": "10115"
				}
			],
			"buyable": "true",
			"fullImage": "\/wcsstore\/CAS\/images\/C_E54_black_140.gif",
			"fullImageAltDescription": "Image for E54 15.0IN CRT 13.8V BL MPR from StoreITB2C",
			"manufacturer": "IBM",
			"metaDescription": "E54 15.0IN CRT 13.8V BL MPR",
			"metaKeyword": " IBM",
			"name": "E54 15.0IN CRT 13.8V BL MPR",
			"parentCategoryID": "10011",
			"parentProductID": "10107",
			"partNumber": "633147N",
			"productType": "ItemBean",
			"resourceId":
"http:\/\/localhost:80\/wcs\/resources\/store\/10151\/productview\/byId\/10109",
			"shortDescription": "E54 15.0IN CRT 13.8V BL MPR",
			"storeID": "10051",
			"thumbnail": "\/wcsstore\/CAS\/images\/C_E54_black_sm.gif",
			"title": "E54 15.0IN CRT 13.8V BL MPR | StoreITB2C",
			"uniqueID": "10109",
			"xcatentry_field1": "1"
		}
	],
	"MetaData": [
		{
			"metaData": "1",
			"metaKey": "price"
		}
	],
	"recordSetComplete": "true",
```

```
        "recordSetCount": "1",
        "recordSetStartNumber": "0",
        "recordSetTotal": "1",
        "resourceId": "http:\/\/localhost:80\/wcs\/resources\/store\/10151\/productview\/byId\/10109",
        "resourceName": "productview"
}
```


Il campo che si è aggiunto è xcatenrty_field1, dove field1 è il nome specificato nella mappatura Solr, mentre xcatentry_ viene specificato nel file 'rest-productview-clientobject.xml', collocato in: REST/WebContent/WEB-INF/config/bodMapping/, come attributo del seguente tag:

```
<!-- UserData Section -->
<_config:URLParameter name="CatalogEntryView/xcatentry_"
nounElement="/CatalogEntryView/UserData/UserDataField" return="true"
type="UserData"/>
```

Volendo questo prefisso può essere modificato a piacimento, l'importante è lasciare la x inziale, convenzione imposta per differenziare le impostazioni native dalle personalizzazioni al fine di non creare sovrapposizioni dei nomi.

7.8 Difficoltà riscontrate

Durante la procedura si sono riscontrate alcune difficoltà .

La prima ha riguardato la poca chiarezza presente nella documentazione del prodotto che espone in maniera poco chiara e frammentata la procedura da seguire per arrivare al risultato.

Il secondo invece è un problema di carattere tecnico , in quanto la procedura è efficiente solo per oggetti in cui il campo fied1 è valorizzato , se field1 non ha valore il server restituirà un'eccezione: java.lang.ClassCastException: java.util.ArrayList incompatible with java.lang.String.

Per cercare un rimedio a ciò è stato contattato il supporto tecnico IBM che ha preso in carico la segnalazione, fornendo la disponiblità di un tecnico per garantire continuità all'operazione.

Mediante e-mail è stato possibile interagire con l'addetto dell'IBM, a cui sono stati mandati i file di configurazione modificati e i log del server WebSphere, questi sono

stati esaminati per fornire diverse prove e soluzioni da applicare per cercare rimedio al problema.

Capitolo 8: Conclusioni

Il tirocinio fatto per la stesura della tesi ha consistito nell'affrontare un percorso formativo su tematiche quotidianamente trattate all'interno di TECLA, offrendo una maturazione delle conscenze professionali dell'argomento utili ad un eventuale impiego all'interno dell'azienda.

Il percorso affrontato ha permesso di approfondire tematiche riguardanti le piattaforme orientate ai servizi, sottolineando l'alta flessibilità e il grado d'integrazione che queste offrono, specie nel mondo enterprise dove sono largamente adottate.

Successivamente è stato analizzato come una soluzione software in comercio possa essere implementata basandosi sulle architetture precedentemente studiate, offrendo quindi un riscontro pratico delle nozioni astratte analizzate.

Si è mostrato come la soluzione in esame rispetti i requisiti imposti dai principi elencati nei modelli architetturali.

In fine seguendo le esigenze dell'azienda TECLA, è stato proposto un'esempio di come può essere personalizzata la risposta di un servizio REST, andando ad inserire un campo precedentemente non previsto nell'oggetto Json passato come risposta dal server al client.

Un possibile approfondimento potrebbe essere fatto a proposito dell'estensione di un servizio REST, un esigenza possibile infatti vede l'aggiunta di un nuovo servizio all'interno della web application, usando oggetti non sono previsti nell'asset standard della soluzione in uso.

La modifica fatta su WCS andrebbe a customizzare maggiormente l'asset presente, estendosi in molteplici maniere a seconda della portata del nuovo servizio.

Nel caso più complesso il servizio richiederebbe dati non presenti sulla base dati, renderebbe quindi necessario la modifica e l'estensione del DB, l'estensione delle

impostazioni di Solr, l'implementazione di un nuovo BOM per la mappatura fisica dei dati, un nuovo profilo di ricerca, e nuove mappature.

Personalmente sono molto grato all'azienda TECLA che ringrazio per l'opportunita datami.

Nel mio percorso sono stato aiutato sin da subito per potermi inserire nel contesto aziendale, dove ho potuto osservare come vengono gestiti progetti di ampia portata e dove mi sono state date le risorse e il supporto per la compilazione dell'elaborato.

Capitolo 9: Bibliografia

Web Services Architecture

http://www.w3.org/TR/ws-arch/

OASIS Reference Architecture Foundation for Service Oriented Architecture Version 1.0

http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/cs01/soa-ra-v1.0-cs01.pdf

Apache Solr Schema.XML

http://wiki.apache.org/solr/SchemaXml

IBM Service-oriented modeling and architecture

http://www.immagic.com/eLibrary/ARCHIVES/GENERAL/IBM/I041109A.pdf

WebSphere Commerce product overview

http://www-01.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.admin.doc/concepts/covoverall.htm?lang=en

Business models

http://pic.dhe.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.admin.doc/concepts/cbmbusinessmodels.htm

Extended sites

http://pic.dhe.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.admin.business.doc/concepts/cbmextendedsites.htm

WebSphere Commerce common architecture

http://www-01.ibm.com/support/knowledgecenter/api/content/SSZLC2_7.0.0/com.ibm.commerce.developer.doc/concepts/csdsoftwarecomp.htm

Web services and WebSphere Commerce

http://www-01.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.webservices.doc/concepts/cwvwebservicestop.htm?lang=en

Overview: WebSphere Commerce search interactions

http://pic.dhe.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=%2Fcom.ibm.commerce.developer.doc%2Fconcepts%2Fcsdsearchinteractionov.htm

WebSphere Commerce framework overview

http://www-01.ibm.com/support/knowledgecenter/api/content/SSZLC2_7.0.0/com.ibm.commerce.developer.doc/concepts/csdruntime.htm

Representational State Transfer (REST) services

http://pic.dhe.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=%2Fcom.ibm.commerce.webservices.doc%2Fconcepts%2Fcwvrest.htm

REST services

http://publib.boulder.ibm.com/infocenter/ieduasst/v1r1m0/topic/com.ibm.iea.wcs/wcs/7.0.0.4/ProgrammingModel/RESTServices.pdf

Understanding the WebSphere Commerce Web service framework

http://www-01.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.webservices.doc/concepts/cwvwc55webservicesguide10.htm?lang=en

REST configuration properties in the component configuration file (wc-component.xml)

http://pic.dhe.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=%2Fcom.ibm.commerce.webservices.doc%2Fconcepts%2Fcsdwccomponentxmlrest.htm

WebSphere Commerce REST API

http://pic.dhe.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=%2Fcom.ibm.commerce.starterstores.doc%2Fconcepts%2Fcwvrestapi.htm

Securing REST services using Secure Sockets Layer (SSL)

http://www-01.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.webservices.doc/tasks/twvrestssl.htm?lang=en

Caching strategies for REST services

http://pic.dhe.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=%2Fcom.ibm.commerce.webservices.doc%2Frefs%2Frwvrestcache.htm

DynaCache technical overview

http://www-01.ibm.com/support/knowledgecenter/linuxonibm/liaag/cache/pubwasdynacachoverview.htm?lang=en

WebSphere Commerce search index schema

http://pic.dhe.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=%2Fcom.ibm.commerce.developer.doc%2Fconcepts%2Fcsdsearchindex.htm

Preprocessing the WebSphere Commerce search index data

http://pic.dhe.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=%2Fcom.ibm.commerce.developer.doc%2Ftasks%2Ftsdsearchbuildpre.htm

WebSphere Commerce use of Open Applications Group (OAGIS) messaging

http://pic.dhe.ibm.com/infocenter/wchelp/v6r0m0/index.jsp?topic=%2Fcom.ibm.commerce.webservices.doc%2Fconcepts%2Fcwvoagis.htm

Business Object Mediator

http://pic.dhe.ibm.com/infocenter/wchelp/v6r0m0/index.jsp?topic=%2Fcom.ibm.commerce.developer.soa.doc%2Fconcepts%2Fcsdbom.htm

WebSphere Commerce BOD command framework

http://pic.dhe.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=%2Fcom.ibm.commerce.developer.soa.doc%2Fconcepts%2Fcsdsoaprogmodel.htm

Mapping REST resources to Business Object Document (BOD) nouns

http://www-01.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.webservices.doc/tasks/twvrestmapping.htm?lang=en

Business Object Document Message Architecture

http://www.oagi.org/oagis/9.0/Documentation/Architecture.html

Data service layer

http://pic.dhe.ibm.com/infocenter/wchelp/v6r0m0/index.jsp?topic=%2Fcom.ibm.commerce.developer.soa.doc%2Fconcepts%2Fcsddsl.htm

Physical data container

http://pic.dhe.ibm.com/infocenter/wchelp/v6r0m0/index.jsp?topic=%2Fcom.ibm.commerce.developer.soa.doc%2Fconcepts%2Fcsdpdc.htm

WebSphere Commerce search

http://www-01.ibm.com/support/knowledgecenter/api/content/SSZLC2_7.0.0/com.ibm.commerce.developer.doc/concepts/csdsearch.htm

IBM RedBook

http://www.redbooks.ibm.com/redbooks/pdfs/sg246969.pdf

Introduzione ai Web Service

http://www.html.it/pag/19595/introduzione-ai-web-service/

I principi dell'architettura RESTful

http://www.html.it/pag/19596/i-principi-dellarchitettura-restful/

Differenze tra Web service REST e SOAP

http://www.html.it/pag/19612/differenze-tra-web-service-rest-e-soap/

Architetture SOA

http://www.html.it/pag/17183/architetture-soa/