Università degli Studi di Padova

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

tesi di laurea

Progettazione e sviluppo di uno strumento automatico per la valutazione della conformità dei geoDBMS alla Simple Features Specification di OGC

Relatore: Massimo Rumor

Laureando: Francesco Bergo

23 Febbraio 2012

a Ruenna

Abstract

The aim of this project is to develop a Java application to test OGC Simple Features Specification compliance of a given set of geoDBMSs. Compliance tests will be performed using Hibernate ORM framework, when possible, in order to ensure project reuse among different geodatabases. The objective will be achieved by designing a testing environment based on the standard specifications. The application execution carries out a detailed report about compliance of tested geoDBMSs, offering users a solid base for software selection decisions.

Sommario

Obiettivo di questo progetto é sviluppare un'applicazione Java per testare la compatibilitá alle OGC Simple Features Specifications di un dato insieme di geomDBMS. I test di compatibilitá saranno eseguiti usando Hibernate ORM, quando possibile, al fine di assicurare il riuso del progetto su differenti geodatabase. L'obiettivo sará raggiunto progettando un ambiente di test basato sulle specifiche dello standard. L'esecuzione dell'applicazione garantirá un report dettagliato sulla compatibilitá del geoDBMS testato, offrendo agli utenti una base solida per le decisioni sulla scelta del software.

Autore: Francesco Bergo

Indice

1 Progetto
1.1 Azienda Ospitante . . . . .
1.2 Obiettivi del Tirocinio . . . . .
1.3 Standard OGC Simple Feature . . . . .
1.3.1 Simple Feature 1.1 [1] [2] . . . . .
1.3.2 Simple Feature 1.2 [3] [4] . . . . .
1.4 Hibernate ORM . . . . .

2 Analisi
2.1 Analisi dei Requisiti . . . . .
2.1.1 Requisiti Generici . . . . .
2.1.2 Test Driven Development . . . . .
2.1.3 Requisiti dell' azienda ospitante . . . . .
2.2 Analisi di Mercato . . . . .
2.2.1 Prodotti package . . . . .
2.2.2 Framework per la lettura dei dati . . . . .
2.2.3 Framework per l'esecuzione dei test . . . . .
2.2.4 Framework per il logging . . . . .
2.2.5 Framework per la produzione di report . . . . .

3 Progettazione
3.1 Casi d'Uso . . . . .
3.1.1 Preparazione . . . . .
3.1.2 Gestione del Database . . . . .

4 Sviluppo software
4.1 Strumenti Utilizzati . . . . .
4.1.1 Eclipse . . . . .
4.1.2 Maven . . . . .
4.1.3 Git . . . . .
4.1.4 PostgreSQL . . . . .
4.1.5 SQLite . . . . .
4.1.6 Simple Feature SQL 1.1 Compliance Test Suite . . . . .

4.2 Framework Utilizzati . . . .
4.2.1 Lettura dati: XStream . . . .
4.2.2 Unit Testing . . . .
4.2.3 ORM: Hibernate . . . .
4.2.4 Logging: log4j . . . .
4.2.5 Reporting: FreeMarker . . . .
4.2.6 Gestione delle Opzioni: Commons CLI . . . .
4.3 Sviluppo Software . . . .
4.3.1 Preparazione . . . .
4.3.2 Creazione e popolamento del database . . . .
4.3.3 Esecuzione dei Test . . . .

5 Esecuzione test e Risultati
5.1 Esecuzione Test . . . .
5.1.1 Creazione e popolamento database . . . .
5.1.2 Esecuzione dei Test . . . .
5.2 Risultati . . . .

6 Conclusioni e Sviluppi Futuri

Elenco delle figure

Capitolo 1

Progetto

1.1 Azienda Ospitante

3DGIS è una startup italiana, costruita su di un'esperienza pluriennale nel campo della progettazione e dello sviluppo di sistemi informativi territoriali e, in generale, di soluzioni nell' area ITC e GeoITC. È la combinazione di diverse professionalità: ingegneri informatici, grafici e architetti esperti in problematiche di dominio, con esperienza di analisi, progettazione e sviluppo di applicazioni GIS e Web GIS 2D e 3D. I suoi prodotti sono sviluppati sia in ambiente Open Source, che su piattaforme commerciali come ESRI, Oracle e Microsoft SQL Server a partire dall'analisi preliminare per l'individuazione delle problematiche ed esigenze, fino alla messa a regime e manutenzione straordinaria del sistema. Le modalità con cui lavora sono sempre di tipo collaborativo, in modo da coinvolgere il cliente in tutte le fasi e renderlo praticamente indipendente alla consegna del prodotto.

1.2 Obiettivi del Tirocinio

Si intende realizzare un ambiente per testare la Simple Feature compliance delle estensioni spaziali di un RDBMS secondo gli standard di OGC. Tale ambiente dovrà essere facilmente configurabile e riusabile, pertanto sarà necessaria una conoscenza sufficientemente approfondita di SQL, dello standard OGC Simple Feature, di Hibernate, e l'applicazione di tali conoscenze ad un applicativo scritto in Java per la compatibilità degli RDBMS utilizzati dall' azienda. La Simple Feature compliance richiede che gli oggetti di tipo Geometrico abbiano determinate proprietà, e rispondano correttamente alle interrogazioni che usano funzioni di tipo geospaziale. È pertanto necessario che l'estensione spaziale del database in questione sappia gestire il tipo di dato geometrico, fornisca tali funzioni per le interrogazioni, e risponda ad esse adeguatamente. OGC fornisce le specifiche dei suoi standard tramite una serie di dettagliati documenti disponibili per la consultazione. Le versioni dello standard Simple Feature prese in considerazione sono

la 1.1[1] [2] e la 1.2[3] [4]. Per testare le varie estensioni spaziali si dovrà disporre di un database già formato, e in base a questo verificare che i dati vengano memorizzati correttamente, rispondendo adeguatamente alle interrogazioni necessarie, secondo lo standard OGC. Si tratta dunque di un test di stato del database, ovvero di come il geoDBMS gestisce i dati di tipo spaziale, ma anche e soprattutto delle funzionalità, ovvero dell'implementazione delle funzioni spaziali, di cui si parlerà in dettaglio nella prossima sezione. Le operazioni di creazione del database, popolamento e interrogazione, dovranno essere quanto più possibile automatizzate tramite Hibernate, pur lasciando all'utente la dovuta libertà di testare secondo le proprie necessità. In più si richiede un sistema di reporting di quanto ottenuto durante i test.

1.3 Standard OGC Simple Feature

OGC è un'associazione internazionale di numerose organizzazioni, pubbliche e private, partecipanti allo sviluppo di standard d'interfaccia disponibili al pubblico. Le soluzioni di OGC sono focalizzate all'interoperabilità nell' utilizzo e nell' elaborazione del dato Geometrico, soprattutto nel Web. L'interoperabilità per simili applicazioni è necessaria, in particolare affinchè esse siano efficienti ed efficaci. Affinchè ciò avvenga è indispensabile la presenza di uno standard. Pertanto qualunque ente od organizzazione decida di operare o progettare applicazioni per un Geographic Information System (GIS) è tenuta a confrontarsi e ad essere compatibile con lo standard internazionale di OGC, che ha sviluppato nel tempo un gran numero di standard, ognuno focalizzato su un determinato tipo di informazione o di tecnologia. OGC si prefigge tramite il Simple Feature di standardizzare la gestione del dato Geometrico, compreso il caso dei database, e dunque dei RDBMS. Il Simple Feature è pertanto ciò che sta alla base di tutto il complesso sviluppato da OGC. Lo standard si presenta in due versioni: la 1.1[1] [2] e la 1.2[3] [4]. La prima risale al 1999 ed è attualmente la più usata in ambito software. Tuttavia il suo utilizzo è attualmente deprecato da OGC, in funzione della più completa e aggiornata 1.2.

1.3.1 Simple Feature 1.1 [1] [2]

È opportuno innanzitutto partire dalla figura 1.1.

Come si può notare, tutte le Feature rientrano nella classe Geometry, pertanto vi sarà una serie di metodi comuni a tutte, ed altri specifici per ognuna, al più ereditati dalle sottoclassi. Per esempio tutte le Geometrie risponderanno alla funzione AsText(), che restituisce una rappresentazione secondo il formato Well Known Text (WKT) della Geometria stessa; mentre tutti e soli i punti risponderanno ai metodi X() o Y(), ovvero le funzioni per ricavare ascissa e ordinata rispettivamente; ma anche le altre Geometrie composte da punti, al più indirettamente, risponderanno a tali metodi, purchè si ricavi da esse una Geometria di

1.3 Standard OGC Simple Feature

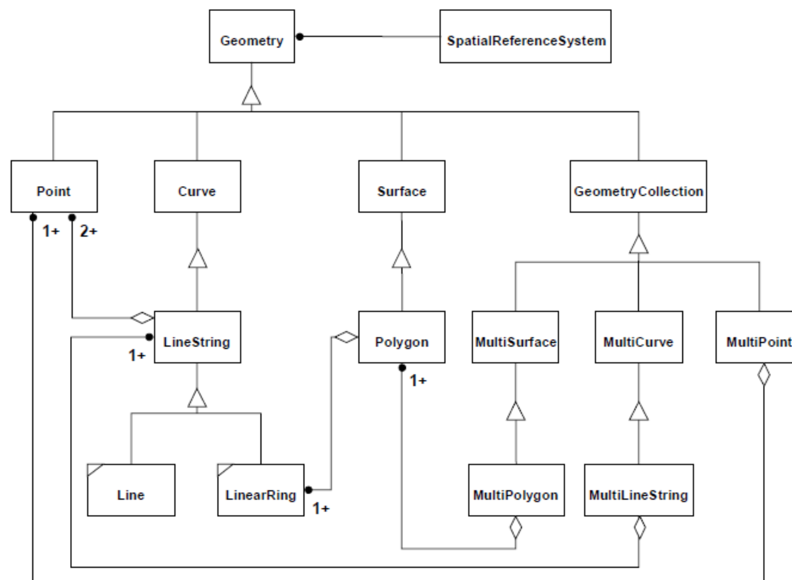

Figura 1.1: Geometrie SFS 1.1

tipo puntuale. Nello specifico una Linestring non può rispondere direttamente al metodo X(), mentre può farlo uno dei punti che la determina, preventivamente ricavato tramite PointN(). I metodi di base per tutte le Geometrie sono:

- Dimension ( ):Integer—restituisce la dimensione della Geometria. tali dimensioni dovranno rispettare i vincoli dello spazio in cui la Geometria è collocata.
- GeometryType ( ):String—restituisce come stringa il tipo di Geometria.
- SRID ( ):Integer— restituisce lo Spatial Reference System ID della Geometria.
- Envelope( ):Geometry— restituisce come Geometria il Bounding Box minimo dell' oggetto Geometrico. Il poligono è definito dai vertici del bounding box ((MINX, MINY), (MAXX, MINY), (MAXX, MAXY), (MINX, MAXY), (MINX, MINY)).
- AsText( ):String —Restituisce la rappresentazione di tipo Well Known Text (WKT) di questa Geometria.
- AsBinary( ):Binary—Restituisce la rappresentazione di tipo Well Known Binary (WKB) di questa Geometria.
- IsEmpty( ):Integer —Restituisce 1 (TRUE) se questa Geometria è vuota.
- IsSimple( ):Integer —Restituisce 1 (TRUE) se questa Geometria non ha punti anomali, ovvero intersezioni o tangenze con se stessa. Ogni tipo Geometrico ha specifiche condizioni per essere definito come Simple.
- Boundary( ):Geometry —Restituisce la chiusura di questa geometria. Questa funzione è specificata con maggior dettaglio nel documento dello standard.

Mentre i metodi destinati all' analisi spaziale sono:

- Equals(anotherGeometry:Geometry):Integer — Restituisce 1 (TRUE) se questa Geometria è 'spazialmente equivalente' ad anotherGeometry.
- Disjoint(anotherGeometry:Geometry):Integer— Restituisce 1 (TRUE) se questa Geometria è 'spazialmente disgiunta' da anotherGeometry.
- Intersects(anotherGeometry:Geometry):Integer— Restituisce 1 (TRUE) se questa Geometria è 'spazialmente intersecante' anotherGeometry.
- Touches(anotherGeometry:Geometry):Integer— Restituisce 1 (TRUE) se questa Geometria è 'spazialmente tangente' ad anotherGeometry.

- Crosses(anotherGeometry:Geometry):Integer— Restituisce 1 (TRUE) se questa Geometria è 'spazialmente intersecante' ad anotherGeometry.
- Within(anotherGeometry:Geometry):Integer — Restituisce 1 (TRUE) se questa Geometria è 'spazialmente dentro' ad anotherGeometry.
- Contains(anotherGeometry:Geometry):Integer — Restituisce 1 (TRUE) se questa Geometria è 'spazialmente contenente' anotherGeometry.
- Overlaps(anotherGeometry:Geometry):Integer — Restituisce 1 (TRUE) se questa Geometria è 'spazialmente sovrapposta' ad anotherGeometry.
- Relate(anotherGeometry:Geometry, intersectionPatternMatrix:String):Integer— Restituisce 1 (TRUE) se questa Geometria è 'spazialmente correlata ad anotherGeometry.
- Distance(anotherGeometry:Geometry):Double— Restituisce il minimo delle distanze fra due punti delle due Geometrie.
- Buffer(distance:Double):Geometry— Restituisce la Geometria composta dai punti a distanza da questa Geometria uguale o minore a quella specificata.
- ConvexHull( ):Geometry— Restituisce l'inviluppo convesso di questa Geometria.
- Intersection(anotherGeometry:Geometry):Geometry— Restituisce la Geometria che rappresenta l'intersezione fra le due.
- Union(anotherGeometry:Geometry):Geometry— Restituisce la Geometria che rappresenta l'unione fra le due.
- Difference(anotherGeometry:Geometry):Geometry— Restituisce la Geometria che rappresenta la differenza fra le due.
- SymDifference(anotherGeometry:Geometry):Geometry— Restituisce la Geometria che rappresenta la differenza simmetrica fra le due.

Di seguito si illustreranno i metodi specifici per ogni tipo di Geometria.

GeometryCollection

Com'è facilmente intuibile dal nome stesso, la GeometryCollection rappresenta una collezione di una o più Geometrie. Presenta i seguenti metodi:

- NumGeometries( ):Integer— restituisce il numero di geometrie di questa collezione.
- GeometryN(N:integer):Geometry— resitutisce l' N-esima geometria della collezione.

Point

Il punto è una Geometria adimensionale, e rappresenta le coordinate di un luogo.

- X( ):Double — restituisce il valore della coordinata x del punto.
- Y( ):Double — restituisce il valore della coordinata y del punto.

MultiPoint

Molto semplicemente, un MultiPoint è una collezione di punti.

Curve

Una Curve è una Geometria monodimensionale, tipicamente rappresentata come sequenza di punti, e il tipo di interpolazione fra essi.

- Length( ):Double— resituisce la lunghezza della Curve, in base al riferimento spaziale.
- StartPoint( ):Point— restituisce il punto iniziale della Curve.
- EndPoint( ):Point— restituisce il punto finale della Curve.
- IsClosed( ):Integer— restituisce 1 (TRUE) se la Curve è chiusa, ossia se punto iniziale e punto finale coincidono.
- IsRing( ):Integer— restituisce 1 (TRUE) se la Curve è chiusa e semplice, ossia non passa due volte per lo stesso punto.

LineString, Line, LinearRing

Una LineString è una Curve con interpolazione lineare, mentre una Line è una LineString rappresentata esattamente da due punti. Una LinearRing è una LineString chiusa e semplice (analogamente a quanto descritto per il metodo IsRing() della Curve).

- NumPoints( ):Integer—The number of points in this LineString.
- PointN(N:Integer):Point—Returns the specified point N in this Linestring.

MultiCurve

Una MultiCurve è una GeometryCollection formata esclusivamente da Curve.

- IsClosed( ):Integer— restituisce 1 (TRUE) se ogni Curve facente parte dell' oggetto è anch'essa chiusa.
- Length( ):Double— restituisce la somma delle lunghezze delle curve facenti parte dell' oggetto.

MultiLineString

Una MultiLineString è una MultiCurve i cui elementi sono LineString.

Surface

Una Surface è una Geometria bidimensionale.

- Area( ):Double— restituisce l'area della superficie, misurata rispetto al riferimento spaziale dell 'oggetto. Centroid( ):Point— restituisce il centroide matematico della Surface. PointOnSurface( ):Point— restituisce un punto sulla Surface.

Polygon

Un Poligono è una Surface definita da un Boundary esterno, e 0 o più boundary interni (definiti Holes, buchi).

- ExteriorRing( ):LineString— restituisce l'anello esterno del Polygon.
- NumInteriorRing( ):Integer— restituisce il numero di anelli interni.
- InteriorRingN(N:Integer):LineString— restiuice l' N-esimo anello interno.

MultiSurface

Una MultiSurface è una collezione di superfici.

- Area( ):Double— restituisce l'area della MultiSurface.
- Centroid( ):Point— restituisce il centroide matematico della MultiSurface.
- PointOnSurface( ):Point— restituisce un punto della MultiSurface.

MultiPolygon

Un MultiPolygon è una MultiSurface formata esclusivamente da poligoni.

La lista di metodi riportata ha il solo scopo di elencare ciò che si intende testare in questo progetto. Per definizioni più dettagliate si faccia riferimento allo standard OGC.

Dal punto di vista di SQL, le tabelle delle Feature possono essere organizzate in vario modo, purchè vi siano gli opportuni riferimenti fra le Feature, le geometrie che costituiscono una loro proprietà, i tipi di Geometria che le rappresenta, e lo Spatial Reference System che modellizza la mappa in questione. Ogni RDBMS che gestisce anche dati di tipo Geometrico fornisce funzionalità speciali ed adattamenti di default a questa esigenza. L'attività di test che ci si è prefissa consiste appunto nel verificare che tali funzioni siano implementate nel RDBMS oggetto di studio, ovvero che l'RDBMS sappia gestire il tipo di dato Geometrico.

1.3.2 Simple Feature 1.2 [3] [4]

La versione 1.2 dello standard Simple Feature conferma tutte le caratteristiche della versione precedente, ma in più ha introdotto importanti innovazioni, innanzitutto due dimensioni aggiuntive, le cui quantità sono restituite dai metodi Z() e M(). La dimensione Z è pensata per rappresentare la terza dimensione, ovvero l'altezza, mentre invece M sta per measure, ed indica una misura associata ad una determinata Geometria. Da notare che tali dimensioni sono facoltative, ovvero non necessariamente una feature di tipo punto facente riferimento a questo standard deve avere un valore definito per Z() e M()(al contrario di X() e Y() che vanno sempre specificati), ma tali metodi devono essere comunque implementati, restituendo al più valori nulli. Inoltre, come si può vedere dalla figura **??**, sono stati introdotti alcuni nuovi tipi di Geometria, ovvero la PolyhedralSurface e le Triangle e Triangulated Irregular Network (TIN) ad essa collegate.

Da notare che il fatto che si vada a testare la versione 1.1 anzichè la 1.2 dello standard è determinato dal database e da come è popolato, ovvero per testare accuratamente la versione 1.2 sarà necessario disporre di un database in quattro dimensioni: gli eventuali valori nulli restituiti dai relativi metodi applicati a un database popolato da oggetti bidimensionali non sarebbero sufficienti a determinare una piena compliance.

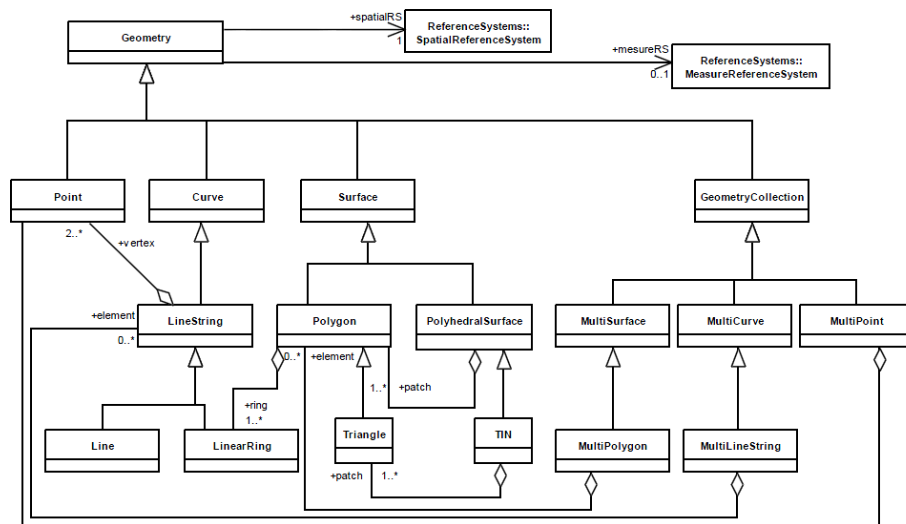

Figura 1.2: Geometrie SFS 1.2

1.4 Hibernate ORM

Un ORM svolge essenzialmente una funzione di mappatura fra un database di tipo relazionale, e un linguaggio di programmazione di oggetti, ovvero garantisce la persistenza in un database, di tali oggetti. Negli anni, il concetto di ORM si è sviluppato attorno al concetto più generico di integrazione di sistemi elaborati secondo la programmazione ad oggetti, e sistemi di tipo RDBMS. In questo modo non solo si ottengono i benefici di entrambi i paradigmi, ovvero la persistenza degli oggetti, ma si migliora l'efficienza e l'efficacia del sistema così realizzato nel suo complesso, per esempio tramite un processo di caching e/o di ottimizzazione delle transazioni. All'aumentare della complessità del sistema da gestire, aumentano anche i benefici dell'utilizzo di un ORM. L'utilizzo di uno strato ulteriore fra oggetti ed RDBMS contribuisce ad aumentare la portabilità complessiva del software, oltre che a migliorarne la facilità di gestione. Hibernate è un ORM open source. In quanto tale, Hibernate si pone, per quanto riguarda il rapporto fra le componenti della Test Suite, fra l'RDBMS e l'ambiente di test vero e proprio (che si ricorda dovrà essere scritto in Java). In particolare Hibernate lavora tramite i cosiddetti POJO , ossia oggetti Java piuttosto semplici, dotati di campi più o meno numerosi, e di metodi di tipo setters e getters che vanno ad interagire con essi. Tali POJO rappresentano le entità che andranno gestite nel database. Hibernate si collega al database tramite un opportuno file di configurazione XML che verrà richiamato nel codice Java, e consente all' utente di specificare come i POJO dovranno poi essere mappati verso il RDBMS, ossia tramite file XML o annotazioni di vario tipo. Questa feature di Hibernate può essere sfruttata nel progetto per la gestione delle opzioni disponibili per l'utilizzo dell' applicativo.

Capitolo 2

Analisi

2.1 Analisi dei Requisiti

2.1.1 Requisiti Generici

Uno dei requisiti più importanti è l'usabilitá del software da produrre, ovvero si vuole semplificare quanto più possibile i metodi di interazione con l'utente, nello specifico per preparare il database e le query che dovranno essere eseguite. A tale scopo si vuole poter comunicare con la test suite esclusivamente tramite file di testo oppure XML, formattati secondo le necessitá. La test suite dovrá essere in grado di estrapolare da tali file XML i dati necessari ad eseguire le operazioni di test. Sorge a questo punto il bisogno di avere a disposizione una parte di codice in grado di leggere file xml, e di popolare di conseguenza le strutture dati che occorrono per le fasi successive. Il test poi sará composto da una serie di unit test. La lista degli unit test fornirá il risultato complessivo del testing, da cui poi si evincerá o meno la compliance voluta. L'esecuzione degli unit test dovrá essere accompagnata da un opportuno sistema di logging. Il logging è certamente importante per la scrittura del codice, in quanto aiuta ad individuare eventuali problemi, ma anche per l'utente finale dell' applicazione, per la tracciabilitá delle operazioni che vengono eseguite. A livello di utenza sará opportuno minimizzare il logging a ció che è indispensabile, trascurando quello che non sarebbe realmente informativo. Una volta pronta la lista coi risultati degli unit test, sará necessario fornire il dovuto report. Come requisito di default si dovrá fornire un report html, ma dovrá anche essere possibile ricevere un report in altro formato, ad esempio XML, partendo da un template con cui il codice andrá ad interagire, e che eventualmente l'utente potrá modificare secondo le proprie esigenze. Questo approccio consente (senza imporlo) di dividere il programmatore Java dal web designer (che opera, ad esempio in HTML). Il web designer produce un template con l'unico presupposto di sapere cosa restituiscono i vari metodi del codice Java, ma in nessun modo è legato al loro funzionamento (purché corretto). Il contesto tipico in cui si andrá ad eseguire l' applicativo è evocando il programma

da riga di comando, ricevendone poi l'output. Prima di eseguire il programma si dovranno fornire i rispettivi file XML di configurazione, nonché un database su cui andare ad effettuare le interrogazioni. Si potranno scegliere le varie opzioni fornite dal software tramite argomenti sulla riga di comando.

2.1.2 Test Driven Development

Come giá accennato, il test si comporrá di una serie di Unit Test. Lo Unit Test è un test atomico, nel senso che verifica una funzionalitá piuttosto semplice, al punto che non se ne potrebbe testare una parte piú piccola. Questa procedura fornisce notevoli garanzie durante lo sviluppo di software, in quanto si è costantemente consapevoli di cosa funziona e cosa no. Il Test Driven Development consiste appunto nell'effettuare test durante tutta la fase di progettazione e sviluppo, anziché quando il prodotto é già almeno in forma di prototipo. Secondo tale filosofia, sono appunto i test e i loro risultati a guidare l'andamento della produzione di codice. Gli Unit Test sono la forma ideale per procedere in questo senso, in quanto, per via delle loro caratteristiche, possono essere pensati ed effettuati in qualunque momento e in qualunque punto del codice. Non si seguirá in questo progetto la filosofia del Test Driven Development in senso stretto, ciononostante si seguiranno i principi dello Unit Test per verificare la OGC Simple Feature compliance. Piú precisamente si userá un dump fornito da OGC, che costruisce e popola la mappa rappresentata in 2.1.

2.1.3 Requisiti dell' azienda ospitante

Oltre ai requisiti precedentemente specificati, l'azienda ospitante richiede che i test vengano eseguiti, dove possibile, tramite Hibernate, essendo uno degli strumenti utilizzati dall'azienda stessa.

2.2 Analisi di Mercato

2.2.1 Prodotti package

OGC supporta il test di vari suoi standard tramite il TEAM Engine, ovvero un tool messo a disposizione online che automatizza i test per la compatibilità ai vari standard. Tuttavia il Simple Feature è uno standard basilare, che viene verificato a livello di Data Tier (secondo l'architettura SOA a tre livelli degli web services), pertanto OGC non fornisce software giá pronto a riguardo. Sono comunque disponibili file che contengano i dump per la creazione e il popolamento di un database, secondo un luogo geografico immaginario, e le query per interrogare il database e verificare che rispettino i risultati attesi, secondo la compliance. Qualora una o più query non verificassero ciò che ci si aspetta, ci sarebbero incongruenze con la OGC Simple Feature compliance. Da notare che i dump forniti vanno comunque adattati al RDBMS che si intende usare, per esempio

2.2 Analisi di Mercato

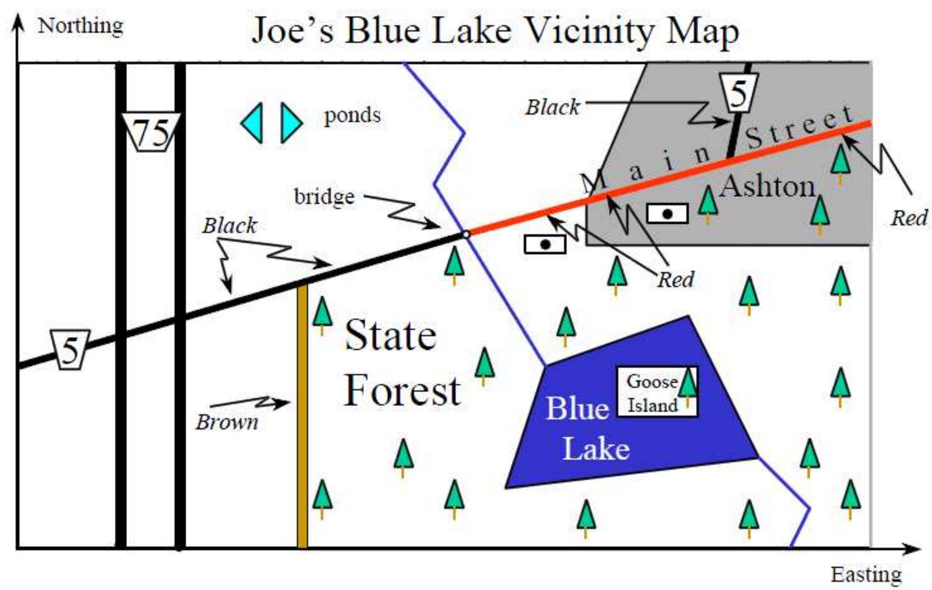

Figura 2.1: Mappa Blue Lake

con l'aggiunta delle opportune chiamate AddGeometryColumn() in caso le si voglia testare su postGIS. Esistono poi sul mercato vari framework di test, ma sono tutti piuttosto generici, e da soli insufficienti per il raggiungimento del nostro scopo. Vale la pena nominare dbUnit, un framework open source per Java in grado di creare un database a partire da un file XML, e di confrontarne lo stato, dopo l'esecuzione di determinate query, con uno stato che ci si attende, anch'esso fornito come XML. Questo framework tuttavia è in grado di testare unicamente stati differenti di un database, mentre l'analisi che ci si è prefissa è molto più orientata alla verifica del risultato di varie funzioni.

A fronte di tutto ciò si evince che sarà necessario produrre da sè la test suite nella sua interezza, eventualmente sfruttando alcuni framework disponibili sul mercato, per l'esecuzione di operazioni specifiche.

2.2.2 Framework per la lettura dei dati

I dati dovranno essere letti a partire da file xml, che sono sufficientemente facili da scrivere per l'utente. A tal proposito esistono vari framework per java, in grado di leggere file xml e inizializzare di conseguenza, o addirittura creare, i corrispondenti oggetti java. Sono stati presi in considerazione i seguenti: JAXP, JiBX, XStream.

JAXP

JAXP sta per Java API for XML Parsing, e costituisce lo standard della libreria di Java per il parsing di file XML. JAXP é infatti incluso nelle ultime versioni del Java SE (Standard Edition). In quanto tale, JAXP fornisce una serie di funzionalità di base, garantendo dunque una certa versatilità. Tuttavia, proprio per questa sua versatilità, JAXP può diventare piuttosto complicato da usare nella programmazione, e pertanto è una possibile fonte di errori e rallentamenti per la produzione. Nello specifico JAXP è in grado di elaborare XML schema e produrre i corrispondenti oggetti Java. Inoltre è in grado di istanziarne degli esemplari a partire da file XML. Va detto che JAXP va ben al di là delle richieste di funzionalitá e performance del presente progetto.

JiBX

JiBX è molto simile a JAXP in quanto a flessibilitá e performance, e si pone come una diretta alternativa. Anch'esso è in grado di creare oggetti Java a partire da un XML schema e viceversa, ed eventualmente di inizializzare tali oggetti. In particolare JiBX è sempre stato molto orientato alle perfomance.

XStream

XStream é un altro framework per il parsing di file XML. Sfrutta una serie di Converter in grado (tramite i metodi di tipo marshal e unmarshal) di passare da

XML a oggetti java e viceversa. Tali operazioni avvengono secondo il principio della reflection. XStream, a differenza degli altri due, non è ottimizzato per le performance, ma si presta comunque ai nostri propositi, essendo che il database da testare è pensato per rimanere invariato su una serie di test, e in ogni caso il tempo impiegato per la lettura da XML é una piccola frazione del tempo totale di esecuzione dei test. Inoltre XStream è in grado di generare e inizializzare una gran diversità di oggetti Java partendo da file XML, senza dover passare per l'opportuno XML schema, semplificando notevolmente la progettazione e la produzione del software, nonchè la configurazione dell'applicativo da parte dell'utente finale, rispetto alle alternative precedentemente esaminate.

2.2.3 Framework per l'esecuzione dei test

Sono presenti sul mercato vari framework per l'esecuzione dei test in Java, e più specificatamente degli unit test che si andranno ad eseguire. Il più famoso e utilizzatoé indubbiamente JUnit, ma va menzionato anche testNG. Entrambi hanno una serie di estensioni per l'esecuzione di funzionalità particolari.

JUnit

JUnit costituisce, grazie al fatto di essere stato supportato da anni da una consistente community, lo standard de facto per lo unit testing. In particolare gli sviluppatori operano secondo la filosofia del test driven development. Questo implica che i test sono parte integrante dello sviluppo, e non la fase conclusiva. JUnit è cresciuto per assecondare questo modus operandi. Ciò ha portato il framework a un notevole livello di qualità, fintanto che se ne rispettano le regole di utilizzo. Nel tempo JUnit ha anche raggiunto una certa flessibilità, non da ultima l'integrazione con altri framework per la produzione di software (come ad esempio Eclipse o Maven).

TestNG

testNG è molto simile a JUnit, dato che ad esso si ispira. Precisamente, TestNG nasce in reazione ad alcuni particolari di JUnit, che secondo Cedric Beust, fondatore di TestNG, portavano ad uno standard incoerente. TestNG ha quindi una serie di varianti che lo rendono più facilmente usabile e più flessibile nella progettazione dei test, ed è per questo meno dipendente da uno standard specifico. Nel tempo, TestNG ha ottenuto una discreta community a suo supporto. Inoltre questo framework offre un sistema di reporting (riutilizzabile fra l'altro per ripetere test già eseguiti) che non necessita di tool esterni. Anche TestNG si integra con altri framework per la programmazione come Eclipse e Maven.

2.2.4 Framework per il logging

La maggior parte dei framework per il logging in Java si appoggi a slf4j, Simple Logging Facade For Java, ossia un framework di facciata per il logging. Slf4j può allacciarsi a diversi sistemi di output, compreso il System.err. Questo garantisce una notevole flessibilità su quale framework di logging si dovrà scegliere per scrivere la parte principale dell' applicativo. Le conseguenze della scelta si saranno esclusivamente sulle classi che andranno a rappresentare i logger. Esistono alternative a slf4j, come Apache Commons Logging, ma slf4j é un prerequisito per Hibernate, e non avrebbe senso introdurre per questo progetto un sistema diverso di facciata per il logging.

log4j

Log4j è il framework di logging più usato in assoluto. Come tale si hanno buone garanzie di una numerosa e valida community. Gli sviluppatori di log4j partono dalla convinzione che il logging sia innanzitutto un aiuto nell' ordinare il codice (anziché il contrario). Inoltre, se opportunamente gestito, il logging può rappresentare una parte minima del carico computazionale dell' applicativo. Per questo log4j fornisce la possibilità di configurare il logging, ovvero decidere quali fra gli statement di logging inseriti nel codice andranno effettivamente elaborati e trasmessi. Tale decisione avviene direttamente durante il runtime (non potrebbe essere altrimenti per via delle caratteristiche fondamentali di Java). In questo modo, anche un logging piuttosto pesante a livello di codice, può essere notevolmente alleggerito in fase di esecuzione. A tale scopo log4j organizza il logging su diversi livelli, ognuno di importanza via via crescente. I livelli sono ereditabili fra le varie istanze di un oggetto di tipo Logger, purché rispettino la gerarchia: DEBUG $<$ INFO $<$ WARN $<$ ERROR $<$ FATAL. Il programmatore è tenuto ad usare nel modo più opportuno tali livelli. Log4j fornisce addirittura la possibilità, benché sconsigliata, di generare livelli personalizzati. Il file di configurazione consente di filtrare i livelli che si vuole elaborare. In ogni caso l'elaborazione del logging è progettata in modo da essere quanto più possibile efficiente in termini computazionali. log4j è molto flessibile, soprattutto per la gestione dei flussi di logging. Tale flessibilità è accompagnata da un trascurabile overhead di prestazioni e conoscenze da parte del programmatore.

JDK logging

JDK logging è il framework di default della distribuzione Java. Fornisce tutte le funzionalità principali per il logging, come ad esempio un sistema di livelli simile a quello di log4j. Tuttavia rispetto a quest'ultimo, la gestione del flusso di logging è notevolmente semplificata.

2.2.5 Framework per la produzione di report

Esistono vari framework per automatizzare il reporting, indipendentemente dalle capacità dei framework precedentemente presentati. In particolare sono stati presi in considerazione Velocity e FreeMarker.

Velocity

Velocity fa parte della Apache Software Foundation. È il framework più usato nell'ambito della produzione di testi a partire da un template, e ne costituisce una sorta di standard. Velocity vanta infatti una community numerosa e consolidata, nonchè un discreto supporto per l'integrazione con altri framework, e ha sempre puntato molto sulla semplicità d' uso e sull' efficienza.

FreeMarker

FreeMarker è anch' esso un tool per la produzione di testi partendo da un template. Inoltre FreeMarker è uno strumento di facile utilizzo ma che offre comunque funzionalità piuttosto complesse, laddove fosse necessario. FreeMarker fornisce di default funzionalità che con Velocity sarebbero realizzabili unicamente tramite più o meno complessi workaround, e questo ne costituisce un punto di forza.

Capitolo 3

Progettazione

3.1 Casi d'Uso

Di seguito si mostreranno più in dettaglio i casi d'uso relativi a questo progetto.

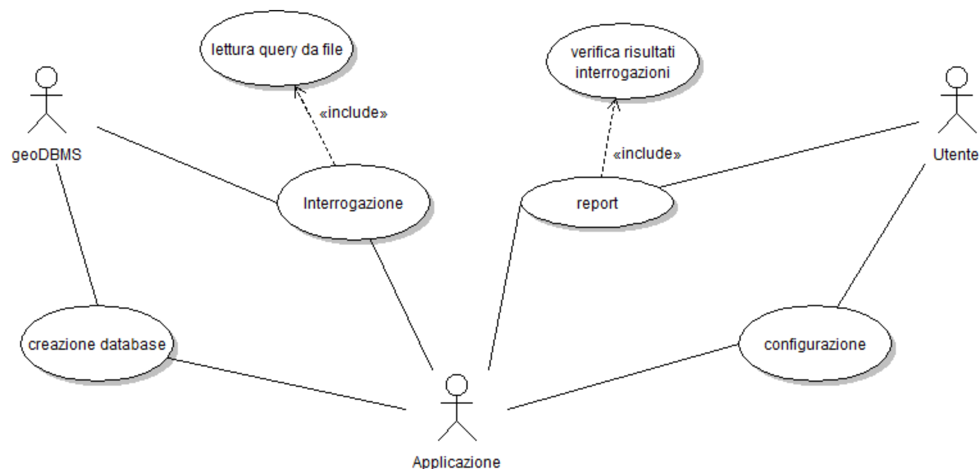

Figura 3.1: Caso d'uso

Come si può vedere dalla figura 3.1, l'utente deve avere una serie di opzioni disponibili in vari momenti dell' esecuzione dell' applicativo. Si può dividere l'esecuzione dell' applicativo in 3 fasi principali: preparazione file di configurazione, gestione del database, ed esecuzione dei test.

3.1.1 Preparazione

La fase di preparazione comprende la gestione di tutti i file che corrispondono all'interazione dell'utente con l'applicativo. Questi file potrebbero essere ad esempio degli XML e/o dei file di testo. L'unica limitazione sarà dovuta al

fatto che per renderli leggibili al software, tali file dovranno seguire un template ben preciso, a cui l'utente dovrà attenersi. I vari template dipenderanno dall'implementazione dell'applicativo e dai vari framework utilizzati.

3.1.2 Gestione del Database

Secondo il requisito di riusabilità, il software dovrà garantire la possibilità di scegliere quale database testare. Questo si tradurrà in ulteriori file di configurazione, e nelle opportune componenti software in grado di gestirli. Dovrà inoltre essere possibile scegliere se effettuare i test tramite Hibernate oppure no, e in caso positivo, se usare query di tipo SQL, oppure il dialetto di Hibernate, ovvero HQL. Si dovrà innanzitutto poter specificare i test da eseguire, composti da Unit Test, ovvero query. Qui si deve rendere disponibile, nuovamente, la possibilità di intervenire tramite Hibernate, ossia specificare se le query da eseguire sono di tipo SQL oppure seguono il dialetto HQL. Successivamente tali query verranno lette da file ed eseguite in sequenza dall' applicativo. Infine si dovrà poter decidere che tipo di report ricevere, se file di testo, XML o HTML.

Capitolo 4

Sviluppo software

4.1 Strumenti Utilizzati

Di seguito si fornirá una breve presentazione dei vari strumenti utilizzati per lo sviluppo.

4.1.1 Eclipse

Eclipse é uno degli IDE (Integrated Development Environment) disponibili per Java. É stato scelto questo anziché, ad esempio, Net Beans, in quanto Eclipse é l' IDE usato all' interno dell' azienda. Oltre alle numerose utilitá che facilitano non di poco la vita del programmatore, Eclipse dá la possibilitá di integrarsi con una notevole lista di plug-ins, per l'aggiunta di ulteriori funzionalitá. Esistono varie distribuzioni di Eclipse, ognuna con le proprie peculiaritá. Per lo sviluppo di questo progetto é stato scelto Indigo Eclipse.

4.1.2 Maven

Maven é uno strumento per la gestione e lo sviluppo di progetti, e agisce secondo il concetto di POM (Project Object Model). É possibile definire la struttura del progetto tramite il file pom.xml. Maven si occupa di configurare di conseguenza il workspace, impostando opportunamente le librerie richieste. In questo modo si facilita anche la gestione del progetto all' interno di un team di piú persone. Per questo progetto sará sufficiente fornire nel pom.xml i parametri necessari affinché Maven sia in grado di caricare dai rispettivi repository le librerie dei vari framework. Un' altra particolaritá notevole di Maven é la gestione automatica delle dipendenze fra le varie librerie. Non é infatti necessario predisporre delle informazioni per caricare librerie a loro volta richiamate da altre che si intende usare. Questo, a fronte di un breve periodo di apprendimento per il corretto utilizzo del framework, consente al programmatore di risparmiarsi notevoli impacci nella gestione delle librerie, che facilmente si moltiplicano all' interno anche di piccoli progetti, e le dipendenze reciproche diventano pertanto assai complicate.

4.1.3 Git

Git é un sistema distribuito di controllo versione, e la sua utilitá si dimostra giá in piccoli progetti, ma brilla particolarmente per progetti piú complessi. I vantaggi di Git sono soprattutto nella gestione delle varie versioni del software, nella visualizzazione dei cambiamenti avvenuti, e in generale della storicizzazione della sua evoluzione. In piú Git, salvando esclusivamente i cambiamenti che avvengono fra versioni differenti, unisce efficienza nella gestione dello spazio all' efficacia nel mantenimento contemporaneo delle suddette versioni del software, potendo tornare indietro in qualunque momento. É possibile inoltre effettuare delle operazioni di branch per effettuare modifiche non definitive e sperimentare col codice in tutta tranquillitá, tornando eventualmente tramite merging a una linea di sviluppo unitaria.

4.1.4 PostgreSQL

PostgreSQL é uno dei due rdbms open source usati per i test. É indubbiamente uno dei piú famosi e usati sul mercato, dati i suoi 15 anni di sviluppo, e presenta un' architettura ormai consolidata, garantendo affidabilitá, integritá, correttezza e ACID compliance. Nel caso specifico di questo progetto, é possibile interagire con PostgreSQL da Java tramite il JDBC, ossia una libreria di Java in grado di interfacciarsi direttamente al database.

PostGIS

PostGIS è l'estensione di PostgreSQL adibita alla gestione dei dati spaziali, ed è dunque ciò che si è interessati a testare. PostGIS offre funzionalità in grado di estendere un database già esistente, ad esempio tramite il metodo AddGeometryColumn, che aggiunge un attributo di tipo Geometrico/Geografico ad una data entità, e aggiornando di conseguenza una relazione specifica, atta a registrare tutti questi attributi come entità a sè stanti, in riferimento ad un sistema di riferimento spaziale (a sua volta memorizzati in un' opportuna relazione). È disponibile un'estensione al JDBC di PostgreSQL per PostGIS.

4.1.5 SQLite

SQLite è un RDBMS open source con proprie peculiarità rispetto alla maggior parte degli altri rdbms disponibili sul mercato, compresto PostgreSQL. In particolare sqlite non risponde ad un' architettura di tipo client-server, bensì il database è rappresentato da un unico file, a cui si può avere accesso diretto. Ciononostante sqlite è noto per la sua affidabilità. Altra particolarità di sqlite è la sua leggerezza, come dice il nome stesso. Proprio per via della sua architettura semplice, sqlite crea database che occupano uno spazio minimo all' interno del supporto di memorizzazione utilizzato. Questa feature lo ha reso particolarmente adatto a strumenti con limitate risorse hardware ma comunque con esigenze di

un database solido, come ad esempio gli smartphone o i lettori mp3. Downside di tutte questi pregi è la mancanza di determinate funzionalità all' interno di sqlite rispetto allo standard SQL92, come ad esempio la mancanza di supporto diretto per FULL OUTER JOIN e RIGHT OUTER JOIN. Per sqlite è stato scelto Xerial come JDBC.

SpatiaLite

Spatialite è la rispettiva estensione spaziale di sqlite, ed è sviluppato e gestito da Alessandro Furieri. Come sqlite, anche Spatialite ha restrizioni (che vedremo nello specifico) rispetto allo standard che punta a seguire (lo standard OGC per quanto riguarda Spatialite).

4.1.6 Simple Feature SQL 1.1 Compliance Test Suite

Come giá precedentemente menzionato, si tratta di una suite fornita direttamente da OGC per testare la compliance allo standard Simple Feature. Comprende tutte le query necessarie (al più adattate all' estensione spaziale specifica) sia per costruire un database basato su un luogo immaginario, sia per eseguire le interrogazioni che costituiscono i singoli unit test.

4.2 Framework Utilizzati

4.2.1 Lettura dati: XStream

Per la lettura di dati a partire da file XML è stato scelto XStream, in quanto l'efficienza computazionale non è un grave problema per questo progetto, e XStream è, fra gli strumenti disponibili, il più facile da usare, in particolare per il fatto di non dover passare attraverso gli XML schema, essendo sufficiente partire direttamente dagli XML. XStream provvede in automatico a creare gli oggetti Java e a inizializzarli con i dati forniti.

4.2.2 Unit Testing

Per lo Unit Testing si è scelto di non usare nessuno degli strumenti presi in esame. Lo sforzo necessario per adattare tali strumenti agli scopi del progetto sarebbe infatti superiore al progettare e realizzare da sè gli opportuni unit test. Il problema principale sta nel fatto che gli unit test che si andranno ad eseguire sono tutti molto simili tra loro, ovvero consistono tutti nell' esecuzione di query SQL (o al più HQL, secondo il dialetto di Hibernate). Questo vuol dire che non avrebbe senso sviluppare a priori, secondo quello che sarebbe lo standard de facto, il codice degli Unit Test, ossia non sussistono come entità a sè stanti, ma sono piuttosto esemplari che hanno vita unicamente all' interno della sessione di test; tant'è che tali query verranno caricate tramite XStream (sono cioè fornite

in serie in una lista di stringhe). Tradotto in termini di programmazione, è più conveniente inizializzare in fase di esecuzione (all' interno di un ciclo) gli esemplari della classe che andrà a rappresentare gli Unit Test. Per questo si è deciso di non usare nessuno dei framework di Unit Test presenti sul mercato, ma di procedere invece costruendo le classi opportune, come si vedrà in seguito.

4.2.3 ORM: Hibernate

L'utilizzo di Hibernate è uno dei requisiti imposti dall' azienda ospitante. Dato che si stanno testando oggetti geometrici, è necessaria un' estensione di Hibernate, ossia Hibernate-Spatial, che fornisce i driver opportuni per agganciarsi ai rispettivi JDBC delle varie estensioni spaziali dei RDBMS.

4.2.4 Logging: log4j

Per il logging è stato scelto log4j, sia perchè più versatile e flessibile, sia perchè il framework utilizzato all' interno dell' azienda per il debug del software. Inoltre log4j fornisce garanzie non indifferenti grazie alla sua community. Le complicazioni di log4j rispetto alle librerie di default di Java sono state valutate irrilevanti. Ci si appoggia comunque a slf4j, anche perchè quest' ultima libreria, come già detto, è uno dei requisiti di funzionamento di Hibernate, che è a sua volta uno dei requisiti di progetto.

4.2.5 Reporting: FreeMarker

Per il reporting è stato scelto FreeMarker, dato che per operazioni semplici presenta una semplicità d'uso paragonabile a Velocity, e garantisce una robustezza tale da non dover incorrere in spiacevoli workaround durante la realizzazione del software (sia del codice Java, che del template in formato ftl). FreeMarker produce l' output (ad esempio un file html) elaborando assieme il template e il codice Java ad esso associato.

4.2.6 Gestione delle Opzioni: Commons CLI

Le opzioni dell' applicativo saranno gestite tramite argomenti sulla riga di comando, ed è stata scelta la libreria Commons CLI per assolvere a questo compito. Commons CLI modellizza le varie opzioni come oggetti, e consente di fare da intermediario fra il software Java e l'utente.

4.3 Sviluppo Software

Si è deciso di sviluppare il software in ambiente Windows, mentre il database postGIS è stato collocato in un server ubuntu virtualizzato. Per Spatialite non è necessaria un' operazione analoga, in quanto la gestione del rispettivo database

è incentrata su un unico file. Basandosi sulla Test Suite fornita da OGC, si è provveduto innanzitutto a creare e mappare nel database i POJO corrispondenti alle Feature che interverranno nei test. Questa operazione di mappatura (effettuata secondo le specifiche di Hibernate) permette la gestione dei dati da relazionali a oggetti, ed è necessaria ogni qualvolta si decida di testare attraverso Hibernate. Oltre a ciò si dovrà fornire i driver di Hibernate per il JDBC del RDBMS che si intende testare. Da ultimo sarà necessario configurare opportunamente il file Properties.txt, contenente tutti i parametri usati dall'applicativo.

4.3.1 Preparazione

Si è provveduto a generare i file XML necessari all' interazione con l'applicativo oltre che il file Properties.txt necessario per la configurazione. Fra gli XML sono compresi i file di configurazione per Hibernate. Ne avremo uno per ogni RDBMS. Tali file saranno costruiti secondo la Test Suite di OGC, ma costituiranno anche un template per l'utente, qualora si scelga di effettuare i test su una base di dati diversa. I file XML avranno il loro indirizzo specificato nel Properties.txt.

4.3.2 Creazione e popolamento del database

Tramite argomenti da riga di comando,sarà possibile evocare la classe principale (TestEngine, contenente il metodo main) con varie opzioni, compresa la creazione e il conseguente popolamento delle tabelle nel database. Si ricorda che il popolamento avverrà, a meno che non si configuri altrimenti l'applicazione, secondo la mappa Blue Lake, presentata nei precedenti capitoli, e opportunamente adattata alle esigenze. Sarà sufficiente fornire l' indirizzo del file Properties.txt assieme al rispettivo argomento di create table e/o quello di inserimento. La creazione può avvenire sia direttamente tramite JDBC, oppure passando attraverso Hibernate. In entrambi i casi si forniranno i dati necessari tramite XStream (dunque file XML opportunamente redatti). Il codice è organizzato in modo da ricevere una lista di liste di Feature: questa scelta è dovuta alla volontà di lasciare all' utente dell' applicativo la libertà di testare qualunque ambiente sia composto da Feature, indipendentemente dal tipo e dalla quantità. Allo stesso modo è possibile effettuare le operazioni di inserimento, purchè si fornisca all' applicativo il rispettivo argomento su riga di comando. La stessa classe è anche in grado di eseguire il drop delle tabelle. Queste operazioni hanno come presupposto che tutti i preparativi siano stati effettuati correttamente, e che gli XML siano scritti secondo le struttura dati del codice (per esempio le query che compongono la Test Suite sono organizzate come lista di TestData). La Test Suite è accompagnata dai relativi file XML necessari, pertanto l'utente può semplicemente sfruttare questi come template, riempiendoli coi dati del proprio ambiente di test. Da sottolineare l'importanza del file Properties.txt, in quanto è ciò che permette all'utente di decidere (assieme ad alcuni argomenti su riga di comando, tramite

Commons CLI) le opzioni che l' applicativo rende disponibili, ovvero in particolare le informazioni relativi ai driver e ai database che si vuole testare ad ogni esecuzione.

4.3.3 Esecuzione dei Test

È necessaria un' ulteriore operazione prima di eseguire i test, ovvero provvedere il file XML contentente le informazioni per eseguire le query. Tali informazioni comprendono la query stessa rappresentata come stringa, il fatto che sia HQL o SQL (in base a come si intende eseguirla), e il risultato che ci si attende. Tutte queste informazioni sono disponibili nella Test Suite di OGC (se si intende eseguire i test basandosi su questo schema relazionale e non uno personalizzato), sarà sufficiente riportarle in un opportuno file XML, di modo che XStream sia in grado di trasporle in Java. A questo punto si potrà eseguire il TestEngine fornendo l'argomento relativo all'esecuzione dei test. Il codice provvederà a leggere il file XML, eseguire le query una alla volta, e confrontarne l'esito con il risultato atteso. In automatico verrà creato il report (ad esempio HTML) partendo dal template.ftl richiesto da FreeMarker. Da qui si potrà vedere quali query non hanno risposto in maniera adeguata, e di conseguenza capire quali funzionalità del RDBMS che si sta testando non corrispondono allo standard OGC.

Capitolo 5

Esecuzione test e Risultati

5.1 Esecuzione Test

5.1.1 Creazione e popolamento database

Come enunciato nei capitoli precedenti, si dovranno fornire le query di creazione e popolamento del database in file XML opportunamente redatti, ovvero con tag che permettano ad XStream di leggerli correttamente. Le query SQL su cui si basa la creazione, secondo la mappa Blue Lake, sono le seguenti:

```
CREATE TABLE geometry_columns
(
  f_table_catalog character varying(256) NOT NULL,
  f_table_schema character varying(256) NOT NULL,
  f_table_name character varying(256) NOT NULL,
  f_geometry_column character varying(256) NOT NULL,
  coord_dimension integer NOT NULL,
  srid integer NOT NULL,
  "type" character varying(30) NOT NULL,
  CONSTRAINT geometry_columns_pk PRIMARY KEY (f_table_catalog,
   f_table_schema, f_table_name, f_geometry_column)
)
--WITH (
 -- OIDS=TRUE
--);
--ALTER TABLE geometry_columns OWNER TO postgres;


-- create spatial_ref_sys table only if missing from the GIS database.
CREATE TABLE spatial_ref_sys (
       srid       INTEGER NOT NULL PRIMARY KEY,
       auth_name  VARCHAR(256),
```

```
       auth_srid  INTEGER,
       srtext     VARCHAR(2048)
       --srtext     VARCHAR(2000)
);

-- specific srid for the test suite.
INSERT INTO spatial_ref_sys VALUES(101, 'POSC', 32214,
'PROJCS["UTM_ZONE_14N", GEOGCS["World Geodetic System 72",
DATUM["WGS_72",  SPHEROID["NWL_10D", 6378135, 298.26]],
PRIMEM["Greenwich", 0], UNIT["Meter", 1.0]],
PROJECTION["Transverse_Mercator"],
PARAMETER["False_Easting", 500000.0],
PARAMETER["False_Northing", 0.0],
PARAMETER["Central_Meridian", -99.0],
PARAMETER["Scale_Factor", 0.9996],
PARAMETER["Latitude_of_origin", 0.0],
UNIT["Meter", 1.0]]'
);

CREATE TABLE lakes (
       fid              INTEGER NOT NULL PRIMARY KEY,
       name             VARCHAR(64)--,
     --  shore            POLYGON
);

SELECT AddGeometryColumn('lakes', 'shore', 101,
 'POLYGON', 2);

CREATE TABLE road_segments (
       fid              INTEGER NOT NULL PRIMARY KEY,
       name             VARCHAR(64),
       aliases          VARCHAR(64),
       num_lanes        INTEGER--,
       --centerline       LINESTRING
);

SELECT AddGeometryColumn('road_segments', 'centerline', 101,
 'LINESTRING', 2);

CREATE TABLE divided_routes (
       fid              INTEGER NOT NULL PRIMARY KEY,
       name             VARCHAR(64),
       num_lanes        INTEGER--,
```

```
       --centerlines       MULTILINESTRING
);

SELECT AddGeometryColumn('divided_routes', 'centerlines', 101,
 'MULTILINESTRING', 2);

CREATE TABLE forests (
       fid            INTEGER NOT NULL PRIMARY KEY,
       name           VARCHAR(64)--,
       --boundary       MULTIPOLYGON
);

SELECT AddGeometryColumn('forests', 'boundary', 101,
 'MULTIPOLYGON', 2);


CREATE TABLE bridges (
       fid           INTEGER NOT NULL PRIMARY KEY,
       name          VARCHAR(64)--,
       --position      POINT
);

SELECT AddGeometryColumn('bridges', 'position', 101,
'POINT', 2);


CREATE TABLE streams (
       fid             INTEGER NOT NULL PRIMARY KEY,
       name            VARCHAR(64)--,
       --centerline      LINESTRING
);

SELECT AddGeometryColumn('streams', 'centerline', 101,
 'LINESTRING', 2);


CREATE TABLE buildings (
       fid             INTEGER NOT NULL PRIMARY KEY,
       address         VARCHAR(64)--,
       --position        POINT,
       --footprint       POLYGON
);
```

```
SELECT AddGeometryColumn('buildings', 'position', 101,
 'POINT', 2);
SELECT AddGeometryColumn('buildings', 'footprint', 101,
'POLYGON', 2);


CREATE TABLE ponds (
       fid             INTEGER NOT NULL PRIMARY KEY,
       name            VARCHAR(64),
       type            VARCHAR(64)--,
       --shores          MULTIPOYLGON
);

SELECT AddGeometryColumn('ponds', 'shores', 101,
 'MULTIPOLYGON', 2);

CREATE TABLE named_places (
       fid             INTEGER NOT NULL PRIMARY KEY,
       name            VARCHAR(64)--,
       --boundary        POLYGON
);

SELECT AddGeometryColumn('named_places', 'boundary',
 101, 'POLYGON', 2);

CREATE TABLE map_neatlines (
       fid             INTEGER NOT NULL PRIMARY KEY--,
       --neatline        POLYGON
);

SELECT AddGeometryColumn('map_neatlines', 'neatline',
 101, 'POLYGON', 2);
```

Mentre le query su cui si basa il popolamento del database sono le seguenti:

```
INSERT INTO lakes VALUES (101, 'BLUE LAKE',
    PolygonFromText
    ('POLYGON((52 18,66 23,73 9,48 6,52 18),
    (59 18,67 18,67 13,59 13,59 18))', 101)
);
INSERT INTO road_segments VALUES(102, 'Route 5', NULL, 2,
    LineStringFromText
    ('LINESTRING( 0 18, 10 21, 16 23, 28 26, 44 31 )' ,101)
```

```
);
INSERT INTO road_segments VALUES(103, 'Route 5', 'Main Street', 4,
    LineStringFromText
    ('LINESTRING( 44 31, 56 34, 70 38 )' ,101)
);
INSERT INTO road_segments VALUES(104, 'Route 5', NULL, 2,
    LineStringFromText
    ('LINESTRING( 70 38, 72 48 )' ,101)
);
INSERT INTO road_segments VALUES(105, 'Main Street', NULL, 4,
    LineStringFromText
    ('LINESTRING( 70 38, 84 42 )' ,101)
);
INSERT INTO road_segments VALUES(106, 'Dirt Road by Green Forest', NULL, 1,
    LineStringFromText
    ('LINESTRING( 28 26, 28 0 )',101)
);
INSERT INTO divided_routes VALUES(119, 'Route 75', 4,
    MultiLineStringFromText
    ('MULTILINESTRING((10 48,10 21,10 0),(16 0,16 23,16 48))', 101)
);
INSERT INTO forests VALUES(109, 'Green Forest',
    MultiPolygonFromText
    ('MULTIPOLYGON(((28 26,28 0,84 0,84 42,28 26),
    (52 18,66 23,73 9,48 6,52 18)),
    ((59 18,67 18,67 13,59 13,59 18)))', 101)
);
INSERT INTO bridges VALUES(110, 'Cam Bridge',
    PointFromText('POINT( 44 31 )', 101)
);
INSERT INTO streams VALUES(111, 'Cam Stream',
    LineStringFromText
    ('LINESTRING( 38 48, 44 41, 41 36, 44 31, 52 18 )', 101)
);
INSERT INTO streams VALUES(112, NULL,
    LineStringFromText
    ('LINESTRING( 76 0, 78 4, 73 9 )', 101)
);
INSERT INTO buildings VALUES(113, '123 Main Street',
    PointFromText('POINT( 52 30 )', 101),
    PolygonFromText
    ('POLYGON( ( 50 31, 54 31, 54 29, 50 29, 50 31) )', 101)
);
```

```
INSERT INTO buildings VALUES(114, '215 Main Street',
    PointFromText('POINT( 64 33 )', 101),
    PolygonFromText
    ('POLYGON( ( 66 34, 62 34, 62 32, 66 32, 66 34) )', 101)
);
INSERT INTO ponds VALUES(120, NULL, 'Stock Pond',
    MultiPolygonFromText
    ('MULTIPOLYGON( ( ( 24 44, 22 42, 24 40, 24 44) ),
    ( ( 26 44, 26 40, 28 42, 26 44) ) )', 101)
);
INSERT INTO named_places VALUES(117, 'Ashton',
    PolygonFromText
    ('POLYGON((62 48,84 48,84 30,56 30,56 34,62 48))', 101)
);
INSERT INTO named_places VALUES(118, 'Goose Island',
    PolygonFromText
    ('POLYGON((67 13,67 18,59 18,59 13,67 13))', 101)
);
INSERT INTO map_neatlines VALUES(115,
    PolygonFromText
    ('POLYGON((0 0,0 48,84 48,84 0,0 0))', 101)
);
```

Le chiamate qui presentate sono relative ad una versione 2D della mappa Blue Lake, ovvero sufficienti per testare lo standard SFS 1.1. Per una mappa 3D e/o 4D (dove la terza dimensione è l'altezza, e la quarta è una misura) sarà sufficiente aggiungere gli opportuni campi in ogni oggetto, e specificare un valore anche per essi. Può bastare anche un valore 0 (o altro) per ogni oggetto, ottenendo ad esempio una mappa 3D identica a quella 2D, con la differenza di essere elevata ad un'altezza specifica. In alternativa si può complicare ulteriormente, progettando una mappa tridimensionale (e/o quadridimensionale) con valori verosimili.Per il test completo di SFS 1.2 è necessario inserire Feature del tipo PolyhedralSurface, Triangle e TIN, e progettare le relative query.

I rispettivi file XML dovranno semplicemente contenere le query e le loro parti negli opportuni tag, ovvero rispecchiando le varibili dei POJO che rappresentano le Feature. Si ricorda che i POJO in questione sono semplicemente una rappresentazione Java delle Feature qui mostrate come SQL, e l'applicativo fornisce file XML con un template di default, costruito per i POJO di Blue Lake. Si noti che le parti di creazione e popolamento database possono essere eseguite in tempi diversi, grazie alle funzionalità messe a disposizione dall'applicativo. Si noti inoltre che le chiamate ad AddGeometryColumn sono qui usate solo perchè PostGIS e SpatiaLite mettono a disposizione tale funzionalità. Si sarebbe comunque potuto organizzare le tabelle in modo diverso, gestendo le geometrie

con opportuni vincoli di entità referenziale, senza tuttavia violare le dispozioni dello standard.

5.1.2 Esecuzione dei Test

Il database è ora pronto per essere testato. Di seguito si fornisce la serie di query, rappresentata qui come SQL, usata come base per questa sessione di test.

```
SELECT f_table_name
FROM geometry_columns;
--
SELECT f_geometry_column
FROM geometry_columns
WHERE f_table_name = 'streams';
--

SELECT coord_dimension
FROM geometry_columns
WHERE f_table_name = 'streams';
--
SELECT srid
FROM geometry_columns
WHERE f_table_name = 'streams';
--
SELECT srtext
FROM SPATIAL_REF_SYS
WHERE SRID = 101;
--

SELECT Dimension(shore)
FROM lakes
WHERE name = 'BLUE LAKE';
--
SELECT GeometryType(centerlines)
FROM divided_routes
WHERE name = 'Route 75';
--
SELECT AsText(boundary)
FROM named_places
WHERE name = 'Goose Island';
--
SELECT AsText(PolygonFromWKB(AsBinary(boundary)))
FROM named_places
WHERE name = 'Goose Island';
```

```
--
SELECT SRID(boundary)
FROM named_places
WHERE name = 'Goose Island';
--
SELECT IsEmpty(centerline)
FROM road_segments
WHERE name = 'Route 5' AND aliases = 'Main Street';
--
SELECT IsSimple(shore)
FROM lakes
WHERE name = 'BLUE LAKE';
--
SELECT AsText(Boundary(boundary))
FROM named_places
WHERE name = 'Goose Island';
--
SELECT AsText(Envelope(boundary))
FROM named_places
WHERE name = 'Goose Island';
--
SELECT X(position)
FROM bridges
WHERE name = 'Cam Bridge';
--
SELECT Y(position)
FROM bridges
WHERE name = 'Cam Bridge';
--
SELECT AsText(StartPoint(centerline))
FROM road_segments
WHERE fid = 102;
--
SELECT AsText(EndPoint(centerline))
FROM road_segments
WHERE fid = 102;
--
SELECT IsClosed(Boundary(boundary))
FROM named_places
WHERE name = 'Goose Island';
--
SELECT IsRing(Boundary(boundary))
FROM named_places
```

```
WHERE name = 'Goose Island';
--
SELECT Length(centerline)
FROM road_segments
WHERE fid = 106;
--
SELECT NumPoints(centerline)
FROM road_segments
WHERE fid = 102;
--
SELECT AsText(PointN(centerline, 1))
FROM road_segments
WHERE fid = 102;
--
SELECT AsText(Centroid(boundary))
FROM named_places
WHERE name = 'Goose Island';
--
SELECT Contains(boundary, PointOnSurface(boundary))
FROM named_places
WHERE name = 'Goose Island';
--
SELECT Area(boundary)
FROM named_places
WHERE name = 'Goose Island';
--
SELECT AsText(ExteriorRing(shore))
FROM lakes
WHERE name = 'BLUE LAKE';
--
SELECT NumInteriorRings(shore)
FROM lakes
WHERE name = 'BLUE LAKE';
--
SELECT AsText(InteriorRingN(shore, 1))
FROM lakes
WHERE name = 'BLUE LAKE';
--
SELECT NumGeometries(centerlines)
FROM divided_routes
WHERE name = 'Route 75';
--
SELECT AsText(GeometryN(centerlines, 2))
```

```
FROM divided_routes
WHERE name = 'Route 75';
--
SELECT IsClosed(centerlines)
FROM divided_routes
WHERE name = 'Route 75';
--
SELECT Length(centerlines)
FROM divided_routes
WHERE name = 'Route 75';
--
SELECT AsText(Centroid(shores))
FROM ponds
WHERE fid = 120;
--
SELECT Contains(shores, PointOnSurface(shores))
FROM ponds
WHERE fid = 120;
--
SELECT Area(shores)
FROM ponds
WHERE fid = 120;
--
SELECT Equals(boundary, PolygonFromText
('POLYGON( ( 67 13, 67 18, 59 18, 59 13, 67 13) )',101))
FROM named_places
WHERE name = 'Goose Island';
--
SELECT Disjoint(centerlines, boundary)
FROM divided_routes, named_places
WHERE divided_routes.name = 'Route 75' AND
named_places.name = 'Ashton';
--
SELECT Touches(centerline, shore)
FROM streams, lakes
WHERE streams.name = 'Cam Stream' AND
lakes.name = 'BLUE LAKE';
--
SELECT Within(boundary, footprint)
FROM named_places, buildings
WHERE named_places.name = 'Ashton' AND
 buildings.address = '215 Main Street';
--
```

```
SELECT Overlaps(forests.boundary, named_places.boundary)
FROM forests, named_places
WHERE forests.name = 'Green Forest'
AND named_places.name = 'Ashton';
--
SELECT Crosses(road_segments.centerline, divided_routes.centerlines)
FROM road_segments, divided_routes
WHERE road_segments.fid = 102 AND
 divided_routes.name = 'Route 75';
--
SELECT Intersects(road_segments.centerline, divided_routes.centerlines)
FROM road_segments, divided_routes
WHERE road_segments.fid = 102 AND
divided_routes.name = 'Route 75';
--
SELECT Contains(forests.boundary, named_places.boundary)
FROM forests, named_places
WHERE forests.name = 'Green Forest'
 AND named_places.name = 'Ashton';
--
SELECT Relate(forests.boundary, named_places.boundary, 'TTTTTTTTT')
FROM forests, named_places
WHERE forests.name = 'Green Forest'
 AND named_places.name = 'Ashton';
--
SELECT Distance(position, boundary)
FROM bridges, named_places
WHERE bridges.name = 'Cam Bridge' AND named_places.name = 'Ashton';
--
SELECT AsText(Intersection(centerline, shore))
FROM streams, lakes
WHERE streams.name = 'Cam Stream' AND
 lakes.name = 'BLUE LAKE';
--
SELECT AsText(Difference(named_places.boundary, forests.boundary))
FROM named_places, forests
WHERE named_places.name = 'Ashton' AND
forests.name = 'Green Forest';
--
SELECT AsText(St_Union(shore, boundary))
FROM lakes, named_places
WHERE lakes.name = 'BLUE LAKE' AND
 named_places.name = 'Goose Island';
```

```
--
SELECT AsText(SymmetricDifference(shore, boundary))
FROM lakes, named_places
WHERE lakes.name = 'BLUE LAKE' OR
named_places.name = 'Goose Island';
--
SELECT count(*)
FROM buildings, bridges
WHERE Contains(Buffer(bridges.position, 15.0), buildings.footprint) = true;
--
SELECT AsText(ConvexHull(shore))
FROM lakes
WHERE lakes.name = 'BLUE LAKE';
```

Si può notare come vengano testate sia le capacità di memorizzazione dei dati geometrici, sia il supporto delle funzioni relative alle Geometrie. Il dump di Blue Lake fornisce anche il risultato atteso per ogni query. Tali risultati sono usati dall'applicativo (leggendoli da file), all'interno degli oggetti che rappresentano le query, al fine di automatizzare la verifica dei risultati. Le query presentate vengono fornite anch'esse in opportuni file XML, assieme ai risultati attesi. L'applicativo provvede ad eseguirle una ad una confrontando i risultati ottenuti con quelli attesi. Contemporaneamente viene popolata una lista di oggetti contenenti le informazioni relative ad ogni Unit Test (uno per ogni query), che viene poi utilizzata per costruire il report, usando un opportuno template.

5.2 Risultati

I risultati di un test sono disponibili una volta eseguiti i passaggi di cui sopra. Nello specifico dei test prefissati in questo progetto, ovvero per PostGIS e SpatiaLite, ci si è assicurati innanzitutto che Hibernate non inficiasse le risposte del RDBMS eseguendoli sia come query SQL che come chiamate nel dialetto di Hibernate, HQL: per entrambe i casi, il risultato è lo stesso.

Eseguendo l'applicativo, si è verificato, controllando sul report le risposte alle varie query, e sul logging eventuali errori, che PostGIS 1.5.2 è compatibile con la versione 1.1 dello standard OGC Simple Feature, e parzialmente anche con la 1.2. Tale parzialità è dovuta al mancato supporto alle Geometrie di tipo PolyhedralSurface, Triangle e TIN, peraltro previsto per la futura versione 2.0 di postGIS. SpatiaLite 2.3.1 è invece limitato alla versione 1.1 a causa della mancanza del supporto alla terza e alla quarta dimensione. Di seguito un esempio di report:

5.2 Risultati

Report di conformità alle OGC Simple Feature

query	actual result	expected result	is HQL	successful
SELECT f_geometry_column FROM geometry_columns WHERE f_table_name = 'streams';	centerline	centerline	false	V
SELECT coord_dimension FROM geometry_columns WHERE f_table_name = 'streams';	2	2	false	V
SELECT srid FROM geometry_columns WHERE f_table_name = 'streams';	101	101	false	V
SELECT srtext FROM SPATIAL_REF_SYS WHERE SRID = 101;	PROJCS["UTM_ZONE_14N", GEOGCS["World Geodetic System 72",DATUM["WGS_72", SPHEROID["NWL_10D", 6378135, 298.26]],PRIMEM["Greenwich", 0], UNIT["Meter", 1.0]],PROJECTION["Transverse_Mercator"],PARAMETER["False_Easting", 500000.0],PARAMETER["False_Northing", 0.0],PARAMETER["Central_Meridian", -99.0],PARAMETER["Scale_Factor", 0.9996],PARAMETER["Latitude_of_origin", 0.0],UNIT["Meter", 1.0]]	PROJCS["UTM_ZONE_14N", GEOGCS["World Geodetic System 72",DATUM["WGS_72", SPHEROID["NWL_10D", 6378135, 298.26]], PRIMEM["Greenwich", 0], UNIT["Meter",1.0]], PROJECTION["Transverse_Mercator"],PARAMETER["False_Easting", 500000.0],PARAMETER["False_Northing", 0.0],PARAMETER["Central_Meridian", -99.0],PARAMETER["Scale_Factor", 0.9996],PARAMETER["Latitude_of_origin", 0.0],UNIT["Meter",1.0]]	false	V
SELECT Dimension(shore) FROM lakes WHERE name = 'BLUE LAKE';	2	THIS_TEST_WILL_FAIL	false	X
SELECT GeometryType(centerlines) FROM divided_routes WHERE name = 'Route 75';	MULTILINESTRING	multilinestring	false	V
SELECT AsText(boundary) FROM named_places WHERE name = 'Goose Island';	POLYGON((67 13,67 18,59 18,59 13,67 13))	POLYGON((67 13,67 18,59 18,59 13,67 13))	false	V
SELECT AsText(PolygonFromWKB(AsBinary(boundary))) FROM named_places WHERE name = 'Goose Island';	POLYGON((67 13,67 18,59 18,59 13,67 13))	POLYGON((67 13,67 18,59 18,59 13,67 13))	false	V
SELECT SRID(boundary) FROM named_places WHERE name = 'Goose Island';	101	101	false	V
SELECT IsEmpty(centerline) FROM road_segments WHERE name = 'Route 5' AND aliases = 'Main Street';	false	false	false	V
SELECT IsSimple(shore) FROM lakes WHERE name = 'BLUE LAKE';	true	true	false	V

Figura 5.1: Test report

Capitolo 6

Conclusioni e Sviluppi Futuri

Come si è dimostrato in questa presentazione, l'applicativo è stato sviluppato per essere riutilizzato in test futuri, essendo questo uno dei punti chiave del progetto. Ad esempio in Spatialite 2.3.1 la gestione di ulteriori dimensioni è previsto per le prossime versioni, che potranno essere nuovamente testate, non appena possibile, con l'applicazione qui presentata. La stessa cosa vale per il supporto di PostGIS 2.0 alle Geometrie PolyhedralSurface, Triangle e TIN. Contribuisce alla riusabilità, nonchè alla solidità dell'applicativo, l'utilizzo di Hibernate come ORM.

In futuro si può pensare di aggiungere nuove funzionalità all'applicativo, come per esempio una verifica di prestazioni e tempi nell' esecuzione, oppure la gestione di risultati più complessi delle query. Per come è stato progettato, l'applicativo potrà certamente essere riutilizzato in futuro, eventualmente per scopi analoghi ma diversi, purchè sia supportato coi relativi driver. In generale l'applicativo si presta a qualunque tipo di test che coinvolga l'esecuzione di query e il confronto di risultati, partendo da semplici file di configurazione.

Bibliografia

[1] Open Geospatial Consortium inc.,

OpenGIS® Implementation Specification for Geographic information - Simple feature access - Part 1:Common architecture,

2005

[2] Open Geospatial Consortium inc.,

OpenGIS Simple Features Specification For SQL Revision 1.1,

1999.

[3] Open Geospatial Consortium inc.,

OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture,

2006.

[4] Open Geospatial Consortium inc.,

OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 2: SQL option,

2006.

[5] java.net

online available,

url = http://jaxp.java.net/

[6] Sosnoski Software Associates Ltd

online available,

url = http://jibx.sourceforge.net/

[7] XStream

online available,

url = http://xstream.codehaus.org/index.html

[8] JUnit
online available,
url = http://www.junit.org/home

[9] TestNG
online available,
url = http://testng.org/doc/documentation-main.html

[10] SLF4J
online available,
url = http://www.slf4j.org/manual.html

[11] Apache Software Foundation
online available,
url = http://logging.apache.org/log4j/1.2/manual.html

[12] Java platform
online available,
url = http://docs.oracle.com/javase/6/docs/api/java/util/logging/package-summary.html

[13] Apache Software Foundation
online available,
url = http://velocity.apache.org/

[14] FreeMarker
online available,
url = http://freemarker.sourceforge.net/docs/index.html

[15] Gavin King, Christian Bauer, Max Rydahl Andersen, Emmanuel Bernard, Steve Ebersole, and Hardy Ferentschik
online available,
url = http://docs.jboss.org/hibernate/core/3.6/reference/en-US/html/

[16] Karel Maesen
online available,
url = http://www.hibernatespatial.org/tutorial.html

Elenco delle figure

1.1 Geometrie SFS 1.1 . . . .
1.2 Geometrie SFS 1.2 . . . .

2.1 Mappa Blue Lake . . . .

3.1 Caso d'uso . . . .

5.1 Test report . . . .