UNIVERSITÀ DEGLI STUDI DI PADOVA

FACOLTÀ DI INGEGNERIA

Corso di laurea Magistrale in Ingegneria Informatica

Servizio di logging, report e backtracking per un processo di generalizzazione

Relatore: **Prof. Rumor Massimo**

Laureando: **Canton Fabio**

Anno Accademico 2011/2012

Sommario

Il logging è un processo di registrazione degli eventi che avvengono all'interno di un sistema, un'applicazione, un servizio, una base di dati o in qualsiasi altra entità programmabile. La creazione di un logger ben strutturato può portare grandi benefici, non solo durante lo sviluppo di un software, ma anche durante la sua fase esecutiva e soprattutto nell'analisi dell'intero processo.

Scopo di questo lavoro di tesi è quello di realizzare un servizio di logging che sia di supporto agli sviluppatori del progetto CARGEN per la realizzazione e l'analisi di un processo di generalizzazione cartografica automatica.

La generalizzazione cartografica è quell'operazione tramite cui le informazioni presenti su di una carta topografica, e la loro simbolizzazione, sono opportunamente selezionate e modificate per adattarsi sia alla scala di rappresentazione che allo scopo per cui la mappa viene redatta. Grazie alla generalizzazione è quindi possibile ricavare una carta geografica partendo da una cartografia a scala maggiore.

L'automazione del processo di generalizzazione comporta quindi continue modifiche agli oggetti geometrici, forniti da un database cartografico, allo scopo di ottenere una rappresentazione che sia adatta alla nuova scala della carta.

Il logger sviluppato per tale progetto si occupa della segnalazione degli eventi generati durante la fase esecutiva del processo, inoltre, mette a disposizione dei programmatori una struttura di supporto per lo sviluppo del codice in grado di dare un riscontro non solo testuale ma anche grafico.

I dati raccolti dal logger durante la fase di esecuzione permettono poi di usufruire di un servizio di backtracking che, partendo dai risultati ottenuti della generalizzazione, consente di analizzare la storia delle modifiche apportate al database cartografico.

INDICE

Capitolo 1

Il logging nella realizzazione ed esecuzione del software

1.1 Introduzione

Nella lingua inglese, il termine log, indica un tronco di legno; nel corso degli anni, però, tale vocabolo ha assunto svariate sfumature. Nel gergo nautico del 1700, il log rappresentava un pezzo di legno utilizzato per indicare, in maniera approssimativa, la velocità di navigazione della nave. Attorno al 1800 il termine log o logbook era usato per indicare il registro di navigazione presente su ogni nave, su cui venivano segnate, ad intervalli regolari, la velocità, il tempo e la forza del vento, oltre che ad eventi significativi accaduti durante la navigazione. Proprio con le caratteristiche di un diario di bordo il termine log è stato importato nel campo informatico per indicare la fase di registrazione cronologica delle operazioni che vengono eseguite ed il file su cui tali registrazioni vengono memorizzate.

Il logging è quindi un processo di registrazione degli eventi che avvengono all'interno di un sistema, un'applicazione, un servizio, una base di dati o in qualsiasi altra entità programmabile. Nella sua semplicità, quella di logging, è un'operazione che va pianificata in maniera approfondita: sono molteplici gli elementi che determinano l'importanza, l'utilità, il peso e l'efficacia di un logger.

In questo lavoro di tesi, con il termine di logger, verrà indicata quella che è l'infrastruttura hardware o software, creata dallo sviluppatore di un progetto, affinché possa essere svolta un'adeguata azione di logging e si arrivi quindi ad ottenere un log che risponda alle esigenze di programmazione o analisi.

Spesso l'espressione log viene utilizzata per indicare il file dove gli eventi vengono registrati, ma questo è vero solo in parte. Il log rappresenta qualunque sistema possa essere utilizzato per registrare e/o notificare un evento; ecco allora che, oltre ad un file, il log può apparire come una notifica su di un output standard, una tabella di un database, un messaggio su un computer remoto oppure come un'entità grafica su di un'opportuna interfaccia.

Ciò che fino ad ora è stato indicato come evento può essere identificato da un'informazione di debug utilizzata durante lo sviluppo di un software, un'eccezione generata da un'applicazione, una richiesta di accesso ad un servizio oppure una o più azioni compiute da un utente durante l'utilizzo di un software.

Molte sono le funzioni per cui viene implementato un logger all'interno di un progetto:

- ✓ Seguire e correggere lo sviluppo di un processo(debug)
- ✓ Segnalare errori
- ✓ Evitare la perdita di dati importanti
- ✓ Analizzare le prestazioni di un programma
- ✓ Garantire la sicurezza di un sistema
- ✓ Realizzare servizi di report e di supporto
- ✓ Ripristinare situazioni precedenti
- ✓ Analizzare le operazioni fatte e determinarne i responsabili

Come si può vedere il processo di logging si rende utile in diversi modi, è dunque importante che chi progetta il logger di un processo identifichi innanzitutto quali sono i motivi per cui questo viene implementato. Il processo di logging, nella quasi totalità dei casi, non viene creato per soddisfare soltanto una delle precedenti funzioni; all'interno di un sistema di elaborazione vengono solitamente generati diversi tipi di log, questi sono riconosciuti come:

- *Log di sistema*: memorizza gli eventi significativi che intercorrono tra il sistema, come fornitore di servizi e le applicazioni, con i clienti dei servizi stessi.
- *Log di applicazione*: molte applicazioni prevedono i propri log su cui sono registrati eventi caratteristici dell'applicazione e che fungono in certi casi da vero e proprio protocollo di entrata e di uscita.
- *Log di base dati*: in questo caso è il sistema gestore di base dati *(DBMS)* che registra le operazioni fatte sulla base dati: *inserimento, aggiornamento, cancellazione di record*. In DBMS evoluti, che forniscono servizi di tipo transazionale, il log è anche la base di riferimento per eseguire le funzioni di transazione completa (commit) o transazione annullata (rollback).
- *Log di sicurezza*: tipico di sistemi informatici complessi o destinati ad ospitare dati particolarmente sensibili, memorizza tutte le operazioni che sono considerate critiche per l'integrità di dati e sistema nonché il controllo dei tentativi di accesso al sistema (autorizzati e non).

Altri tipi di log possono essere generati da un sistema, in tabella abbiamo una dimostrazione di quali siano i log utilizzati da tre diversi processi; per ogni tipo compare anche la descrizione degli eventi che vengono segnalati.

Windows System Logs	Security logs	Tentativi di accesso validi e non validi
		Uso di risorse come creazione, apertura o eliminazione di file
	Application logs	Eventi registrati da applicazioni o programmi
		Informazioni utili allo sviluppatore
	System logs	Eventi registrati dai componenti di sistema
Network devices Logs	Firewall logs	I pacchetti in entrata e in uscita
		Informazioni sui server
		I pacchetti che sono stati eliminati
		Avvisi alla Security Association
	IDS logs	Segnalazioni relative a pacchetti sospetti
		Statistiche sugli attacchi
Application Server Logs	Web Server logs	Errori
		Accessi
	Mail Server logs	Stato delle connessioni
		Code SMTP
		Protocollo di stato (IMAP, POP3, SMTP)
	FTP Server Logs	Login correnti
		Comandi eseguiti
		File caricati e scaricati
	Database Server Logs	Attività degli utenti
		Oggetti accessibili
		Creazione e modifica di tabelle

Altro aspetto da affrontare durante l'implementazione di un logger è il formato con cui i record relativi agli eventi vengono visualizzati o registrati, se ciò che viene generato dal log è un messaggio destinato ad un utilizzatore del sistema allora il record deve essere intelligibile; nel caso in cui l'avviso generato sia di supporto ad uno sviluppatore sarà fondamentale che il record sia interpretabile da quest'ultimo. Diverse considerazioni vanno invece fatte per i record destinati ad essere registrati su file, in tal caso la quantità di dati potrebbe essere elevata, si rende quindi necessario comprimere il più possibile le informazioni così da non creare file estremamente pesanti e non rischiare di gravare sulle prestazioni del sistema. Sarà poi necessario utilizzare un applicativo per la traduzione dei file compressi e per gestire opportunamente le informazioni immagazzinate.

1.2 Le strategie di Logging

Per analizzare appieno quali siano le riflessioni da fare nell'ideazione di un logger efficiente, è giusto prendere in considerazione separatamente quelle che sono le due fasi principali nella realizzazione di un progetto software: la fase di sviluppo e quella di esecuzione del processo.

1.2.1 Il logging in fase di sviluppo

Per realizzare al meglio un logger, capace di andare incontro alle esigenze e alle aspettative di chi lo deve utilizzare durante lo sviluppo di un progetto software, è giusto porsi molte domande su quello che dev'essere il suo impiego.

In questa fase qualunque programmatore, dal più esperto all'ultimo dei novizi, si ritrova ad utilizzare dei messaggi di log per seguire, testare e correggere l'evoluzione del proprio software; stiamo naturalmente parlando dei messaggi di debug. Per tali segnalazioni viene spesso utilizzato lo standard output tramite una chiamata diretta; in ambiente di sviluppo Java, ad esempio, viene impiegato più e più volte il metodo *System.out.print* per visualizzare il valore di una variabile, seguire l'indice di un ciclo o per tantissime altre segnalazioni che permettano di controllare il progresso del programma.

Anche se in modo piuttosto grezzo, le note generate dal metodo *print*, rappresentano dei messaggi di log. Molti di coloro che utilizzano questo metodo lo fanno perché non conoscono il logger e le sue potenzialità, altri invece perché convinti che l'uso di *print* sia il miglior modo per realizzare il debug delle classi.

Ponendo il caso di essere un programmatore che, dopo un'adeguata opera di debug, giunge alla conclusione della scrittura di un programma, si vorranno eliminare tutti quei messaggi che sono stati utili in fase di correzione ma che ora non hanno nessuna utilità e, soprattutto, non devono comparire durante l'esecuzione del codice.

Se i messaggi di debug sono stati generati tramite il metodo *print*, si dovranno ricercare in tutte le classi del progetto le chiamate alla console ed eliminarle una ad una. Il vantaggio principale nell'utilizzo di un logger sapientemente implementato sta nel fatto che questo consente allo sviluppatore di disattivare determinate chiamate e determinati canali di output riducendo abbondantemente la mole di lavoro in fase di pulizia del codice. Questa funzionalità si rende altresì importante per il recupero ed il ripristino dei messaggi di debug nel caso in cui il software necessiti di una revisione futura.

Vista la fondamentale importanza dell'operazione di debug nella fase di sviluppo di un progetto software, è quindi utile che il logger metta a disposizione una struttura per poterlo realizzare in modo semplice e funzionale.

In questa stessa fase è giusto anche prendere in considerazione come debbano essere organizzate le chiamate al logger in funzione della complessità del progetto da sviluppare.

Nel caso in cui si tratti di un progetto composto da una sola classe, il logger, può essere definito indipendentemente come una variabile dinamica o statica. Se invece il progetto è composto da molti moduli e sviluppato da un gruppo di programmatori, bisognerà fare maggior attenzione alla sua definizione ed al suo utilizzo.

Se non c'è la necessità di fare riferimento ad un unico logger per tutto il progetto, allora ogni classe ha la possibilità di definire un proprio logger, ognuno con un nome diverso. Se, invece, tutte le classi devono utilizzare lo stesso logger si può procedere definendo un logger statico per ogni classe che faccia riferimento al logger della classe principale, oppure il logger può essere passato come parametro nelle chiamate tra le classi.

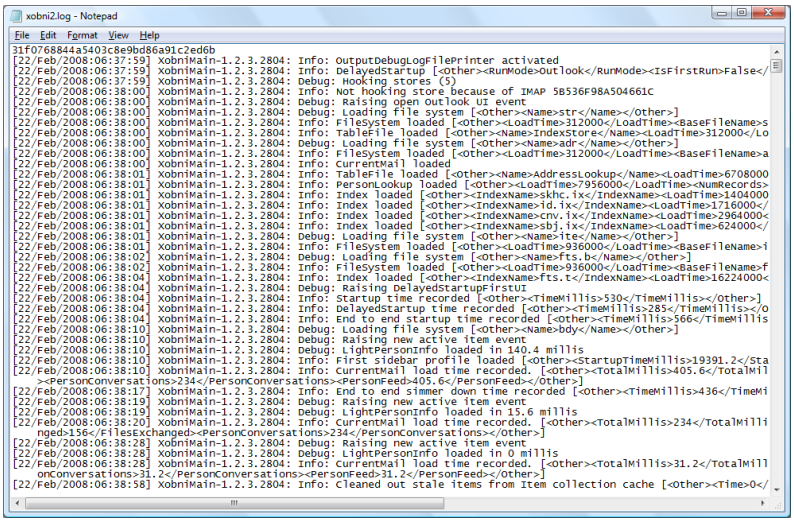
Altro aspetto da considerare per quanto riguarda l'utilizzo del logger in fase di sviluppo è che quest'ultimo deve rappresentare un supporto alla programmazione e non un peso, le chiamate ed i metodi di log non devono confondere chi programma, ma ,al contrario, devono essere esplicative e semplici da utilizzare.

1.2.2 Il logging in fase di esecuzione

La fase di esecuzione di un programma è solitamente quella che produce la parte più pesante ed interessante di log. Gli eventi che si registrano possono essere eccezioni generate dal programma, passi eseguiti dal software oppure azioni compiute da un utilizzatore.

Nel momento in cui si deve creare la parte di logger relativa all'esecuzione del software è innanzitutto necessario distinguere quelli che saranno gli eventi da riportare sul log e definire l'output sul quale dovranno comparire o essere registrati.

Ad alcune segnalazioni non si desidera dare grande importanza e spesso vengono indirizzate verso uno output standard come la consolle; ma non per tutti gli eventi si può agire in tal modo. Ci sono alcuni comportamenti del sistema dei quali si vuole mantenere traccia a lungo o per i quali lo standard output non è ritenuto un supporto adeguato, bisogna quindi prendere in considerazione l'utilizzo di altri strumenti come la scrittura su file oppure su database, facendo però attenzione alla quantità di informazioni immagazzinate. Per grandi progetti, la fase di sviluppo, può avere una durata dell'ordine dei minuti se non addirittura delle



ore, portando con se una sequenza infinita di operazioni ; bisogna quindi fare attenzione all'uso che si fa del logger, non vi si possono riportare tutti i passi, ma solo quelli considerati importanti.

Considerando il fatto che non è sempre possibile ridurre al minimo le chiamate al logger durante l'esecuzione, si rende indispensabile, prevedere un modo più conciso possibile per immagazzinare i dati comprimendo il più possibile le informazioni. In tal caso sarà poi necessario l'uso di un applicativo per rendere le righe di log intelligibili ed analizzabili.

Gran parte dei dati registrati dal logger durante l'esecuzione di un programma viene poi utilizzata per l'analisi del processo e dei suoi errori, è questa la fase riconosciuta con il nome di Report il cui scopo è quello di trasformare le righe di log in informazioni preziose. Per agevolare il lavoro compiuto dal report è quindi importante immagazzinare le informazioni in modo tale che queste possano essere reperite ed utilizzate facilmente.

1.2.3 Il logging per l'uso di interfacce

Non tutte le informazioni utili al programmatore in fase di sviluppo o di esecuzione di un processo devono forzatamente essere testuali, ci sono processi che manipolano dati per cui le righe di testo non sono sufficientemente informative.

In tal caso il logger può rivelarsi ancor più importante vista la sua possibilità di utilizzare diversi canali per la gestione dell'output, uno di questi può essere un'interfaccia grafica generata dal logger stesso oppure da un applicativo con il quale il logger comunica.

Si rende quindi opportuno considerare tutte le possibili modalità per comunicare dati ed eventi sia durante la fase di sviluppo che quella di esecuzione, soprattutto per quel che riguarda il report dei dati.

1.3 Supporti per la realizzazione del logging

Per ogni linguaggio di sviluppo esistono diverse strutture in grado di permettere la realizzazione di un logger efficace; in questo paragrafo andrò ad analizzare le due più famose API per il raggiungimento di un ottimo processo di logging nella programmazione Java.

1.3.1 Il package java.util.logging

Java.util.logging è lo standard Java per l'implementazione di un logger, al suo interno si trovano le seguenti classi:

- ConsoleHandler
- FileHander
- Filter
- Formatter
- Handler
- Level
- Logger
- LoggingPermission
- LogManager
- LogRecord
- MemoryHandler
- SimpleFormatter
- SocketHandler
- StreamHandler
- XMLFormatter

Tra queste ricoprono un'importanza maggiore le seguenti classi:

Logger

Un oggetto di tipo Logger si ottiene chiamando uno dei metodi factory `getLogger` della classe LogManager che creano una nuova istanza di Logger se questa non esiste, oppure restituisce l'istanza appropriata, se questa esiste già. La sua esistenza viene valutata in basa al nome che viene dato al logger, questo viene solitamente attribuito in maniera gerarchica con nomi separati da punti.

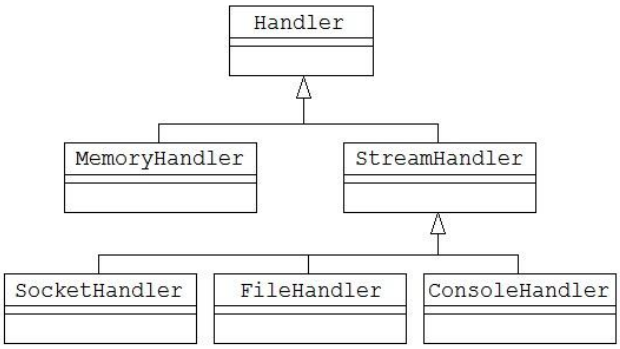
Ogni Logger tiene traccia di un Logger "genitore", che è il suo più vicino antenato esistente nello spazio dei nomi dell'oggetto e dal quale eredita le proprietà.
Ad ogni Logger è associato un "livello" ed un "handler"; il primo definisce il livello minimo dei messaggi che il logger può inoltrare mentre l'handler definisce su quali output deve agire l'oggetto stesso. Il livello di log può essere configurato sia utilizzando un file di configurazione, che ne definisce le proprietà, oppure tramite il metodo `setLevel`.

LogRecord

Un oggetto di tipo LogRecord rappresenta l'essenza dell'attività di logging e cioè la segnalazione dell'evento, un'istanza di tale classe viene usata per passare le richieste di logging tra il framework e i vari handlers.

Handler

Indirizza gli oggetti istanza di LogRecord alle destinazioni designate, queste possono essere lo standard error, lo standard output, uno o più file, oppure un socket. A tale scopo il package mette a disposizione diversi Handler standard ma è possibile svilupparne altri. Anche agli handler possono essere associati diversi livelli.



Level

Definisce un insieme di livelli di log standard che permettono di controllare l'output. Configurando il livello di log vengono loggati alcuni tipi di messaggi mentre altri vengono ignorati. I livelli predefiniti, in ordine decrescente, sono i seguenti:

SEVERE	*Il valore più alto*, usato per messaggi di una certa importanza (es. errori fatali). Questo livello è utile principalmente agli utenti finali o ad un system administrator.
WARNING	Si usa per i messaggi di avvertimento, per segnalare potenziali problemi. Di interesse per utenti finali e system manager
INFO	Usato per messaggi di informazione a runtime. Questo livello è utile principalmente agli utenti finali o ad un system administrator.
CONFIG	Usato per informazioni relative alla configurazione di un'applicazione. Utile agli sviluppatori in fase di debug di una particolare configurazione di un software.
FINE	Usato per informazioni di dettaglio durante il debug o il test di un'applicazione. Questo livello, insieme ai successivi FINER e FINEST, sono di interesse per lo sviluppatore che vuole tenere traccia, più o meno dettagliatamente ,dell'output del software che sta sviluppando.
FINER	Come FINE ma con ulteriore dettaglio
FINEST	*Il valore più basso,* altissimo dettaglio.

Settando un determinato livello di logging si abilitano anche i livelli di log più alti. In aggiunta ai livelli descritti in tabella, c'è il livello ALL che abilita il log di tutti i messaggi e il livello OFF che disabilita il log. E' inoltre possibile creare livelli di log personalizzati.
Ad ogni chiamata di un metodo di log del Logger avviene un check del livello di log: se il livello di priorità associato al metodo è minore del livello di log settato, non viene loggato nulla.

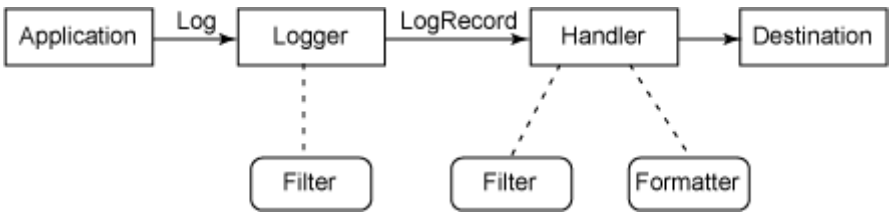
Filter

Un oggetto di tipo Filter permette di aggiungere un controllo dell'output più fine rispetto a quello già svolto dai livelli di log. Un'istanza di tipo Filter può essere associata sia ad un oggetto di tipo Logger che ad uno di tipo Handler.

Formatter

Serve per definire il formato di output del log. Il package include due formatter standard SimpleFormatter e XMLFormatter da cui si ottiene, rispettivamente, come output un testo piano e un XML. Altri formatter possono essere sviluppati per ottenere output personalizzati.

Il funzionamento del package di Java può essere rapidamente riassunto in una breve sequenza di passi: per ogni richiesta fatta ad un *Logger* viene creato un oggetto LogRecord che contiene le informazioni per ogni singolo evento su cui eventualmente eseguire il logging. Questi oggetti sono passati a diversi *Handler* che si occuperanno di pubblicarli su vari canali (console, file, ecc.).



Sia i *Logger* che gli *Handler* possono sfruttare vari Levels (che corrispondono alla gravità dell'evento di cui viene tenuta traccia) e, opzionalmente, dei Filters per decidere se un *LogRecord* meriti di essere pubblicato o no. In caso affermativo l'*Handler* può usare, opzionalmente, un *Formatter* per localizzare e formattare il messaggio prima di pubblicarlo.

1.3.2 Log4j di Apache

La più famosa API per realizzare il logging in Java è Log4j, un progetto Open Source dell'Apache Software Foundation e diffuso. Al suo interno, Log4j, contiene diverse classi; le più importanti sono sono:

- Logger
- Appender
- Layouts

Logger

Un'istanza della classe Logger specifica cosa si vuole tracciare. Molto spesso viene utilizzato il Root Logger, ovvero un Logger che si trova al top della gerarchia di package. Utilizzandolo si decide che per l'applicazione è sufficiente uno solo Logger che traccerà tutto. E' tuttavia possibile definire diversi Logger utilizzando una gerarchia di nomi separati da dei punti.

Al Logger sono associabili 5 diversi livelli di logging:

FATAL	Segnalazione di errori molto gravi che potrebbero portare l'applicazione ad abortire
ERROR	Segnalazione di errori che potrebbero consentire all'applicazione di continuare a funzionare
WARN	Situazioni potenzialmente dannose
INFO	Messaggi informativi a grana grossa che evidenziano i progressi dell'applicazione
DEBUG	Messaggi informativi utili per il debug dell'applicazione

Ci sono poi due ulteriori livelli speciali, ALL, possiede il rank più basso ed è inteso per abilitare tutto il logging, mentre OFF possiede il rank più alto e serve per disabilitare tutto il logging.

Appender

Specifica dove si vuole loggare (console, file di testo etc.). Log4j mette già a disposizione diversi Appender. I più utilizzati sono i seguenti:

- ConsoleAppender permette di scrivere sulla console dell'applicazione
- FileAppender permette di scrivere su file
- RollingFileAppender permette di scrivere su un file di testo definendone la lunghezza massima. Quando la lunghezza massima è raggiunta, il file è rinominato aggiungendo un numero progressivo al nome del file (rotazione del file di log)
- DailyRollingFileAppender permette di scrivere su un file di testo definendone un intervallo di tempo massimo. Quando il tempo scade, viene creato un altro file
- SocketAppender permette di scrivere su un socket utilizzando il protocollo TCP/IP
- JMSAppender permette di scrivere su una coda JMS
- SMTPAppender permette di inviare mail utilizzando il protocollo SMTP e JavaMail
- JDBCAppender consente di inviare i log ad un database, configurando le tabelle riceventi

A ciascun Appender è possibile associare un Layout.

Layout

Specifica con quale formato devono comparire le informazioni loggate. Log4J mette a disposizione diverse tipologie di Layout predefinite. Le principali sono le seguenti:

- SimpleLayout che produce stringhe di testo semplice
- PatternLayout che produce stringhe di testo formattate secondo un pattern definito nel file di configurazione
- HTMLLayout che produce un layout in formato HTML
- XMLLayout che produce un layout in formato XML

Se il layout non viene specificato, log4j utilizza il SimpleLayout.

Il modo migliore per configurare la libreria di Apache, ed utilizzarla in un'applicazione, è quello di scrivere un file di properties in cui impostare gli appender da usare, con quale formato e secondo quali livelli. Il file può anche essere scritto in formato xml.

La struttura tipica di un file di configurazione è la seguente:

```
#LOGGER
log4j.rootCategory=DEBUG, APPENDER_OUT, APPENDER_FILE

#APPENDER_OUT
log4j.appender.APPENDER_OUT=org.apache.log4j.ConsoleAppender
log4j.appender.APPENDER_OUT.layout=org.apache.log4j.PatternLayout
log4j.appender.APPENDER_OUT.layout.ConversionPattern=%5p [%t] (%F:%L) -
%m%n

#APPENDER_FILE
log4j.appender.APPENDER_FILE=org.apache.log4j.RollingFileAppender
log4j.appender.APPENDER_FILE.File=mioLog.log
log4j.appender.APPENDER_FILE.MaxFileSize=100KB
log4j.appender.APPENDER_FILE.MaxBackupIndex=1
log4j.appender.APPENDER_FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.APPENDER_FILE.layout.ConversionPattern=%p %t %c - %m%n
```

Nell’esempio, il Logger, viene impostato con livello DEBUG e gli vengono associati due Appender: APPENDER_OUT e APPENDER_FILE. Ciascun Appender definisce un indirizzamento del flusso. Nell’esempio l’APPENDER_OUT è di tipo Console mentre l’APPENDER_FILE è di tipo RollingFile.

Nel layout si specifica di scrivere:

- %p : priorità dell'evento di LOG
- %t : nome del thread che genera l'evento
- %c : categoria
- %m : messaggio passato dall'applicazione
- %n : ritorno a capo

Capitolo 2

La generalizzazione ed il progetto Cargen

2.1 La cartografia

Ogni giorno sono molteplici i modi che abbiamo di comunicare con le altre persone; possiamo utilizzare parole, gesti, simboli, numeri e disegni. Quando usiamo un metodo grafico per comunicare un'informazione spaziale allora parliamo di cartografia.

La cartografia è quel ramo della geografia che ha per oggetto la rappresentazione grafica, in proporzioni ridotte, della superficie terrestre o di una parte di essa, fornendo una possibile descrizione dei suoi particolari naturali e artificiali.

Le funzioni base della cartografia sono:

- dare una conoscenza del territorio sia puntuale (basata cioè sull'osservazione di ogni singolo oggetto) che generale (visione d'insieme);
- consentire di sviluppare processi logici di tipo deduttivo e induttivo in funzione di relazioni di concomitanza, vicinanza, frequenza,...;
- fungere da supporto di base per classificazioni, pianificazione, progettazione e gestione del territorio.

2.1.1 La carta geografica

Una carta geografica è una rappresentazione grafica, riprodotta in scala e secondo simboli convenzionali, di una parte o di tutta la superficie terrestre, o di alcuni suoi aspetti particolari (diz. Garzanti)

Le carte geografiche vengono comunemente suddivise in due distinte tipologie:

- *Carte topografiche* che forniscono informazioni metriche e descrittive di interesse generale della superficie fisica della Terra o di una porzione di questa. Oltre a informazioni orografiche e idrografiche, sono presenti anche le opere umane. Ad esempio possono essere indicate le principali vie di trasporto, o i principali insediamenti umani, sullo sfondo dello spazio naturale (fiumi, coste, rilievi, etc.)

- *Carte tematiche* che forniscono informazioni specifiche su circoscritti fenomeni di particolare interesse.

La produzione di una carta geografica è un processo molto complesso durante il quale è necessario affrontare molteplici problemi.

Il primo è dovuto al fatto che la superficie terrestre è ricurva, mentre il supporto di una carta geografica è una superficie piana; il secondo è legato alle dimensioni: gli elementi che compongono la realtà non possono essere rappresentati in dimensioni reali, ma devono passare attraverso un processo di riduzione di scala; il terzo è legato alla complessità intrinseca nella realtà stessa: l'infinita eterogeneità degli elementi che compongono la realtà la rendono impossibile da rappresentare se non previa una necessaria sua semplificazione. Durante la produzione di una carta geografica viene quindi creata un'astrazione della realtà, in cui gli elementi subiscono sia trasformazioni spaziali, sia trasformazioni semantiche: il risultato di questo processo è quindi una approssimazione della realtà.

Un aspetto di basilare importanza nella cartografia, e più precisamente nella definizione di una mappa, è il concetto di scala di rappresentazione (o semplicemente scala). La scala di una carta geografica indica quante volte una porzione della superficie terrestre è stata ridotta per poter essere rappresentata su un foglio di carta. Questa è solitamente espressa come il rapporto tra una distanza sulla carta e la corrispondente distanza sul terreno: ad esempio, una misura di 1 mm di lunghezza in una mappa in scala 1:25000 corrisponde ad una lunghezza di 25 metri nella realtà.

Pensando alla scala come al valore di una frazione, che diminuisce all'aumentare del denominatore della frazione, si può dire che una mappa in 1:5000 ha una scala più grande di una mappa in 1:25000; il valore della scala è quindi inversamente proporzionale al fattore di riduzione adottato: all'aumentare del secondo diminuisce il primo.

Oltre alla dimensione dell'area che può essere rappresentata sulla carta, il rapporto di scala, determina anche il grado di dettaglio delle informazioni contenute sulla stessa. Questo segue dal fatto che la scala di rappresentazione governa due importanti parametri: il grado di risoluzione e l'errore massimo di posizionamento.

- Il grado di risoluzione, e cioè la dimensione lineare del particolare più piccolo rappresentabile, è dato dal minimo spessore del tratto grafico con cui la carta viene disegnata, e viene assunto, per convenzione, uguale a 0,2 mm; l'errore massimo di posizionamento di un punto rappresenta il diametro del cerchio al cui interno il punto è sicuramente contenuto e corrisponde all'incertezza con cui è rappresentata la posizione di un generico punto sulla carta. Questo valore è assunto a 0,5 mm.
- Il livello di dettaglio di una mappa ha un impatto sull'accuratezza dei dati presentati, sul suo contenuto e in generale sulla rappresentazione grafica, ma anche sul costo di produzione e di manutenzione della mappa stessa.

2.1.2 L’evoluzione cartografica, i GIS

Negli ultimi anni del secolo scorso la cartografia ha conosciuto un momento di grande sviluppo, che potremmo considerare una sorta di rivoluzione, con l’introduzione dell’informatica in questo campo, e la nascita dei primi Sistemi Informativi Geografici. I dati cartografici si svincolano dalla staticità del tradizionale supporto cartaceo, e possono essere visualizzati in modo dinamico su un monitor, creando così la possibilità di nuovi utilizzi e, al contempo, introducendo nuove problematiche e necessità.

I GIS, infatti, permettono di manipolare oggetti spaziali e di estrarre, in vari modi, una grande quantità di informazioni dalle mappe: è così che la mappa si trasforma in una sorta di oggetto dinamico, mentre sino a poco prima era vista come un elemento statico. Il singolo dato geografico diventa un'entità dotata di attributi e inserita in una gerarchia relazionale; a questa entità corrisponde, oltre che una rappresentazione grafica, un contenuto logico e semantico.

Un GIS rappresenta una potente serie di strumenti per acquisire, memorizzare, estrarre a volontà, trasformare e visualizzare dati spaziali dal mondo reale. In altre parole, un GIS permette di integrare operazioni che tipicamente offre un database con i vantaggi offerti dalla visualizzazione della mappa. Le entità geografiche, elementi base del GIS, vengono memorizzate all'interno di un database geografico (GeoDB), detto anche database spaziale, la cui funzione è quella di memorizzare, interrogare e manipolare informazioni geografiche e dati spaziali.

Più precisamente, un sistema GIS è costituito da risorse e procedure. Le risorse coinvolte sono risorse umane, infrastrutturali e di dati; le procedure, invece, si suddividono in acquisizione, archiviazione, elaborazione, presentazione e trasmissione.

I dati sono in assoluto l'elemento più prezioso del sistema, perché hanno un costo di produzione elevato. Essi, inoltre, per mantenere il loro valore, devono costantemente essere aggiornati.

Ogni dato, chiamato feature, è costituito da due componenti:

- la componente spaziale, che contiene informazioni relative alla posizione (geografica), alla geometria (forma e dimensione) e alla topologia (relazioni spaziali con altri dati) del dato;
- e la componente non spaziale, formata da dati descrittivi, o attributi, e metadati associati alla componente spaziale che velocizzano le operazioni di query spaziale.

L'acquisizione dei dati avviene normalmente scannerizzando le mappe cartacee tradizionali; un altro metodo, invece, si basa sull'utilizzo di file contenenti le coordinate degli elementi geografici, espresse secondo un determinato sistema di riferimento. In entrambi i casi, l'acquisizione è seguita da un processo di astrazione e generalizzazione.

Una volta che i dati sono stati acquisiti e opportunamente memorizzati è possibile effettuare su di essi operazioni di elaborazione e trasformazione degli elementi geografici tramite degli strumenti di analisi forniti dal GIS.

Alcune di queste operazioni sono:

- L'overlay topologico, in cui si effettua una sovrapposizione tra gli elementi di due temi per crearne uno nuovo, ad esempio per sovrapporre il tema dei confini di un parco con i confini dei comuni in modo da determinare le superfici di competenza di ogni amministrazione o la percentuale di area comunale protetta;
- Le query spaziali, ovvero interrogazioni dei dati a partire da criteri spaziali (vicinanza, inclusione, sovrapposizione etc.) ;
- Il buffering, che permette di creare un poligono che circonda la geometria originaria;
- La segmentazione, che applicato su un elemento lineare, determina il punto alla distanza specificata dall'inizio dell'elemento;
- La network analysis, che applicata su una rete di elementi lineari, ad esempio la rete stradale, determina i percorsi minimi tra due punti;
- La spatial analysis, che effettua un'analisi spaziale di varia tipologia utilizzando un modello raster, come ad esempio l'analisi di visibilità;

2.1.3 Il processo cartografico

Con processo cartografico si definisce tutto l'insieme di procedimenti e operazioni necessari alla creazione di una carta geografica.

È rimasto sostanzialmente invariato nel tempo, e può essere schematizzato nei passi seguenti:

1. Definizione
2. Analisi
3. Acquisizione dei dati
4. Costruzione della mappa
5. Collaudo

<u>Definizione e analisi</u>

Nel processo di definizione e analisi si decidono le caratteristiche che la mappa dovrà possedere. I parametri da definire sono molteplici: tra questi si possono citare la scala, la proiezione e il supporto fisico su cui verranno rappresentati i dati.

In queste fasi viene stabilito cosa la mappa deve rappresentare e in quale modo deve farlo. Le scelte fatte influenzano sia caratteristiche tecniche che semantiche, ovvero relative ai contenuti. Per le prime vengono scelti parametri quali superficie di riferimento, proiezione adottata, la scala, etc.

Le decisioni prese in questa fase definiscono una prima astrazione della realtà e delineano a livello semantico un primo modello della realtà. Questa modellazione, inoltre, nella

cartografia tradizionale su supporto cartaceo porta alla creazione della legenda, mentre in quella digitale alla definizione del geodatabase.

Acquisizione dei dati

L'acquisizione dei dati differisce a seconda della provenienza dei dati di origine. Se questi sono ottenuti direttamente dalla realtà, tramite acquisizione diretta sul territorio, si parla di mappe rilevate. Nel caso in cui le informazioni siano estratte da cartografia preesistente le mappe si dicono derivate. È importante specificare che queste ultime si possono ottenere solo utilizzando carte a scala maggiore rispetto a quella della mappa che si vuole ottenere.

La tecnica più usata per l'acquisizione diretta delle informazioni è la fotogrammetria. Questa prevede l'utilizzo di immagini fotografiche stereoscopiche del terreno. Solitamente queste immagini sono riprese da un aereo, in sequenze denominate strisce (strip). Ogni striscia si sovrappone con quelle adiacenti di circa il 15% su entrambi i lati. Appositi strumenti, detti stereorestitutori, permettono di associare le coordinate degli oggetti contenuti nelle strisciate e restituirne latitudine, longitudine ed elevazione.

Durante questa fase le informazioni vengono interpretate e classificate: ad ogni oggetto acquisito viene assegnato un codice, che stabilisce a quale specifico elemento del modello precedentemente creato questo appartiene (es. casa, ferrovia, autostrada, fiume, etc.).

Costruzione della mappa

La fase successiva all'acquisizione dei dati è la costruzione della mappa. Il cartografo deve produrre una mappa che non solo soddisfi le specifiche delineate nella fase di definizione e analisi, ma che risulti usabile e leggibile. Per svolgere il lavoro non esistono però schemi precisi, ma si deve affidare alla sua conoscenza ed esperienza. Un altro suo compito consiste nell'estrarre il contenuto dei dati di partenza creando un'astrazione della realtà efficace e rappresentativa che faciliti la comprensione dell'informazione. Questo processo è definito generalizzazione cartografica, e verrà descritto in maniera più completa nel prossimo paragrafo.

La fase di costruzione viene finalizzata con un raffinamento estetico. Possono essere citati vari esempi quali la creazione di ombreggiature per il delineamento dell'orografia, il posizionamento di riferimenti, l'assegnazioni di vestizioni ai vari particolari.

Collaudo

Il collaudo consiste in una serie di test mirati alla verifica di correttezza e consistenza della carta prodotta. Questa fase può produrre anche un raffinamento estetico del prodotto. Alcuni esempi di test sono il controllo dei punti d'appoggio, della restituzione, del disegno e il controllo finale sul terreno mediante operazioni di misura e verifica della rappresentazione cartografica, che si svolge confrontando i dati rilevati sul terreno tramite strumenti ad alta precisione (per esempio GPS differenziale) con i dati riportati sulla carta.

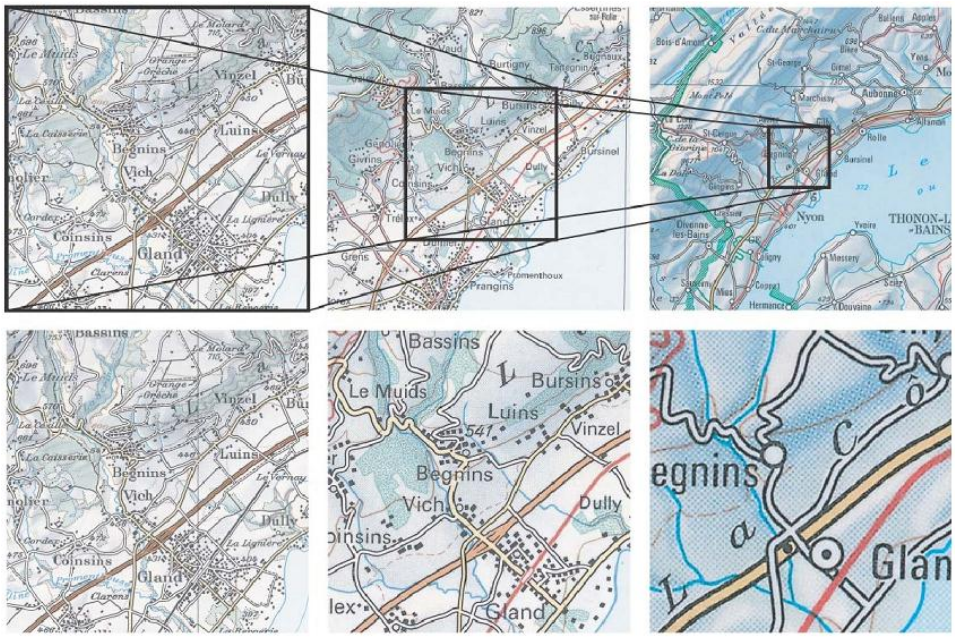
2.2 La Generalizzazione

La generalizzazione è, senza alcun dubbio, uno dei passi chiave nella realizzazione di una mappa.

Osservando una carta geografica, si può notare come gli elementi della realtà vengono classificati, ridefiniti e posizionati, sempre secondo un ordine logico e grafico. Se non fossero compiute queste azioni di classificazione e ridefinizione, l'informazione veicolata tramite la carta sarebbe sensibilmente inferiore. Ad esempio, se tentassimo di riportare in una mappa topologica di una città in scala 1:100 000 l'intero edificato, otterremmo nient'altro che una nuvola di punti, creando solamente disordine e confusione; se, invece, volessimo riportare nella stessa carta tutte le strade, non riusciremmo più ad individuare con facilità le strade principali, come quelle che portano da un lato della città all'altro.

Ne consegue che non ha senso tentare di rappresentare nella mappa tutta la realtà; al contrario, è necessario selezionare le informazioni rilevanti. Il risultato sarà dunque una astratta e più o meno limitata parte della realtà.

Lo scopo della generalizzazione, così, è proprio quello di astrarre la realtà per rappresentarla in modo che essa risulti chiara, comprensibile e leggibile, tenendo presente la funzione che deve svolgere la mappa, e in accordo con la scala di rappresentazione scelta.



Tenendo a mente le osservazioni precedenti, la generalizzazione può essere allora definita come il processo che punta alla semplificazione delle informazioni geografiche per soddisfare i vincoli di rappresentazione e per raggiungere la finalità della carta, mettendo in evidenza gli elementi importanti e al tempo stesso rimuovendo quelli irrilevanti.

I fattori che devono essere tenuti in considerazione dal processo di generalizzazione sono molteplici; McMaster e Shea (1992) ne individuano sei:

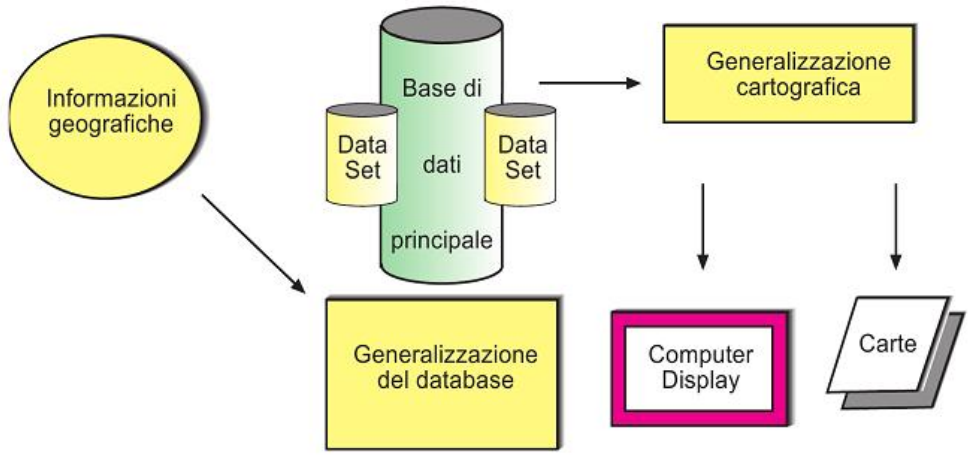
1. Riduzione della complessità: il passaggio ad una scala inferiore porta naturalmente diversi oggetti ad entrare in conflitto per lo stesso spazio. Ridurre il numero di questi oggetti consente di far risaltare gli elementi più importanti e di mantenere la mappa leggibile.
2. Mantenimento dell'accuratezza spaziale: tanto più la scala è alta e tanto più l'accuratezza spaziale deve essere rispettata, limitando al minimo l'errore dovuto alla diversa posizione degli oggetti nella mappa rispetto alla realtà.
3. Mantenimento dell'accuratezza degli attributi: in particolare nelle mappe tematiche, l'obiettivo è quello di minimizzare le alterazioni non intenzionali degli attributi delle feature.
4. Mantenimento della qualità estetica: dai colori utilizzati alla simbologia, dal bilanciamento allo stile tipografico, vari fattori influenzano l'estetica complessiva di una mappa; l'arte del cartografo sta nel mixare opportunamente questi fattori in modo da ottenere una mappa esteticamente bella.
5. Mantenimento di una logica gerarchica: elementi di una stessa categoria, ma di importanza o dimensioni differenti, devono essere differenziati, in accordo con lo scopo della mappa; ad esempio, una città grande deve risultare molto più prominente di una città piccola, mostrando una maggiore densità del grafo stradale e dell'edificato.
6. Coerente applicazione delle regole di generalizzazione: allo scopo di ottenere una generalizzazione imparziale e coerente, il cartografo deve determinare esattamente quali algoritmi applicare e in quale ordine e i parametri di input necessari per ottenere il risultato voluto quando si opera ad una data scala.

A seconda dei dati di partenza per la costruzione della mappa, si possono distinguere due processi di generalizzazione: se i dati disponibili sono il frutto di una rilevazione, cioè di un'acquisizione diretta, allora si parla di “compilazione cartografica”; se, invece, i dati sono quelli provenienti da una cartografia preesistente, si parla di “generalizzazione cartografica”. In entrambi i casi, comunque, il risultato sarà un nuovo modello della realtà.

2.2.1 Il processo di generalizzazione

Il processo di generalizzazione può essere definito come “la selezione e la rappresentazione semplificata dei dettagli che meglio si adatta alla scala e o allo scopo di una mappa” (Meyen, 1973)

Come possiamo osservare in figura, due sono le componenti che caratterizzano questo processo: la generalizzazione del modello e la generalizzazione cartografica, che verranno esaminate in dettaglio nelle sezioni seguenti.



Generalizzazione del modello

Per mezzo della generalizzazione del modello, il risultato che si ottiene è la creazione di un modello astratto, rappresentativo della realtà; la scelta relativa a quali aspetti della realtà riportare e quali invece eliminare avviene, come già detto, in funzione degli scopi per cui viene realizzata la mappa. Ad esempio, in una carta politica verrà dato risalto ai confini amministrativi mentre elementi fisici come i rilievi o le depressioni non verranno riportati.

Questo processo porta alla definizione di una tassonomia degli elementi della realtà, cioè ad una loro suddivisione in classi. Anche i parametri che portano a distribuire gli oggetti in più classi variano a seconda delle funzioni della mappa che si vuole realizzare: così, una carta civile può classificare le strade in urbane ed extraurbane, mentre una carta militare può enfatizzare maggiormente una distinzione delle vie di comunicazione basata sulla loro larghezza e tipo di fondo.

Nella cartografia digitale, la generalizzazione del modello corrisponde alla definizione di un GeoDatabase. In fase di definizione del database, è fondamentale ricordare che una modellazione e strutturazione efficiente della base di dati rappresenta un presupposto necessario per la buona riuscita della fase di generalizzazione cartografica, nonché per un proficuo utilizzo della cartografia digitale in ambiente GIS.

Generalizzazione cartografica

La seconda componente della generalizzazione è quella cartografica, che agisce sulle informazioni geografiche delle feature.

Una volta definito il modello astratto della realtà, si passa alla ridefinizione delle geometrie, allo scopo di fornire una coerente rappresentazione grafica al modello dei dati. Questa fase è di fondamentale importanza per garantire una mappa accurata e comprensibile. Il cartografo, per ridefinire le geometrie, si deve affidare alla sua interpretazione personale: non esiste infatti nessuno schema rigido che si possa applicare sistematicamente ad ogni singolo caso. A

titolo d'esempio, un agglomerato di case potrebbe venire rappresentato come un unico blocco da un primo operatore e da due o più blocchi da un secondo operatore.

Tra i fattori che possono influenzare le scelte compiute dal cartografo troviamo in primis la scala, e successivamente il contesto. In relazione al valore di scala, un centro urbano potrebbe venire rappresentato con un punto affiancato dal nome della città oppure da un suo sottoinsieme di strade ed edifici. Il contesto invece determina il livello di importanza attribuito ad un oggetto, che può determinare la sua eliminazione o la precisione con cui lo si rappresenta. Ad esempio, un edificio posizionato in una zona densamente edificata potrebbe essere semplificato, o addirittura omesso, mentre una fattoria, posizionata in aperta campagna, potrebbe venire riportata accuratamente.

2.2.2 La generalizzazione automatica

La generalizzazione manuale, che un tempo era l'unica opzione praticabile, è un'operazione lunga ed onerosa; fortunatamente, le tecnologie informatiche hanno permesso di automatizzare questa procedura, riducendo notevolmente fatica e tempo che l'operatore umano deve dedicarvi.

Per capire l'importanza che riveste l'automazione del processo di produzione delle mappe geografiche, può esser utile ricorrere ad un esempio pratico. In Francia, per la produzione di un foglio 91 x 121 cm della serie Topo1001 con i metodi tradizionali, l'IGN2, impiega mediamente 2000 ore di lavoro, di cui 1200 per la generalizzazione e 800 per il posizionamento della toponomastica. Se supponiamo che i fogli vengano realizzati in sequenza, ci vogliono circa 20 anni per realizzare le mappe relative all'intero territorio francese. Diminuendo il tempo di produzione, invece, si possono aggiornare le mappe con sempre maggior frequenza raggiungendo, naturalmente, anche un obiettivo di risparmio economico. Così, nel 2003, l'IGN ha utilizzato per la prima volta un software per la generalizzazione automatica (LAMPS2 della Laser-Scan), riducendo così i tempi medi per la produzione di un foglio a 150 ore per la generalizzazione e a 160 per il posizionamento dei nomi.

È evidente, dunque, il motivo per cui il campo della generalizzazione automatica desta sempre più interesse presso tutti gli enti cartografici del mondo, stimolando la produzione scientifica in questo innovativo settore.

La ricerca relativa all'automazione di questi processi si è focalizzata soprattutto sulla generalizzazione cartografica, in quanto sembra l'unica delle due fasi della generalizzazione per cui è possibile immaginare una forma di automazione; la generalizzazione del modello, infatti, presuppone una capacità di astrazione ed un livello di conoscenza difficilmente trasferibile ad un calcolatore.

La realizzazione di un processo di generalizzazione automatico si deve comunque scontrare con alcuni ostacoli (Mackaness, 2007). Un primo, comprensibile ostacolo è sicuramente la

complessità del processo di progettazione, che non può prescindere dalla valutazione di molteplici fattori e vincoli: la mappa è un complesso mix di pattern metrici e topologici che solitamente sono molto interdipendenti. Interpretare correttamente queste forme e individuare le caratteristiche notevoli della mappa richiede una conoscenza sia cartografica che geografica. Un secondo ostacolo, inoltre, ha a che fare con la trasformazione delle informazioni dovuta ad un cambiamento di scala: le mappe infatti, al variare della scala, mettono in luce caratteristiche geografiche diverse.

Il processo di generalizzazione deve, perciò, essere in grado di estrapolare queste differenti caratteristiche in funzione della scala, a partire dalla stessa base di dati.

2.2.3 Gli operatori della generalizzazione

Quando si deve generalizzare una carta ci sono numerosi vincoli da soddisfare; il progetto, infatti, ha delle specifiche che devono essere rispettate nella soluzione del problema di generalizzazione. Un vincolo è una specifica di progetto che deve essere rispettata nella soluzione ad un problema di generalizzazione. Tra i più importanti vincoli si possono citare i seguenti:

- ✓ vincoli di tipo grafico, legati a parametri come la dimensione, la larghezza e la distanza;
- ✓ vincoli topologici, legati a connessione, adiacenza e inclusione tra elementi;
- ✓ vincoli spaziali, legati alla conservazione delle forme e degli allineamenti;
- ✓ vincoli semantici, legati alle relazioni logiche che intercorrono tra gli oggetti e i loro insiemi;
- ✓ vincoli di disegno, legati all'aspetto grafico della carta.

Allo scopo di soddisfare questi vincoli, una grande varietà di soluzioni sono stati studiate nel corso degli anni; queste soluzioni si chiamano operatori di generalizzazione.

Un operatore di generalizzazione rappresenta un tipo di trasformazione spaziale che si vuole ottenere (Weibel and Dutton, 1999). Gli operatori sono stati sviluppati sia emulando le pratiche manuali dei cartografi, sia tramite studi puramente matematici.

Ogni operatore, comunque, prende vita mediante un algoritmo di generalizzazione che lo implementa.

Diverse classificazioni sono state proposte per dare un ordine logico a questi operatori, in modo da facilitare i cartografi nella scelta dell'operatore che più si presta a soddisfare un particolare vincolo. Alcuni ricercatori suddividono gli operatori in base al tipo di geometria a cui fanno riferimento (ad esempio l'operatore di semplificazione è progettato per elementi lineari, mentre l'operatore di fusione opera su oggetti areali).

Un'importante classificazione è quella proposta da McMaster e Shea (1992), che distinguono gli operatori in operatori che trasformano le geometrie e operatori che trasformano gli attributi. Queste due classi saranno descritte in dettaglio nei seguenti due sottoparagrafi.

Operatori geometrici

Gli operatori geometrici sono operatori che agiscono sull'aspetto grafico e topologico di uno o più oggetti geografici.

Una possibile classificazione degli operatori geometrici è stata prodotta all'interno del progetto AGENT; questi sono stati classificati in base al numero di oggetti su cui agiscono. Nella prima classe troviamo quelli che operano su un singolo elemento, e sono:

- simplification operator: riduce la granularità dei contorni di linee e aree, in pratica producendo una versione semplificata dell'oggetto grazie all'eliminazione della ridondanza di punti;
- collapse operator: trasforma oggetti areali in punti o linee risolvendo la progressiva mancanza di spazio.
- Enhancement operator: valorizza un oggetto in vari modi, ovvero ingrandendo la sua forma completa (enlargement) o una sua parte (exaggeration), addolcendo il suo contorno per migliorare l'estetica (smoothing), squadrando la sua geometria se si avvicina ad una forma rettangolare (squaring);

Nella seconda classe vi sono gli operatori che possono essere applicati indistintamente ad uno o più oggetti:

- selection operator: seleziona gli elementi importanti considerando lo scopo della mappa;
- elimination operator: elimina dalla mappa gli oggetti ritenuti non importanti o ridondanti;
- displacement operator: sposta di posizione un oggetto o un gruppo mantenendone inalterata la forma.

Infine, nella terza categoria ricadono gli operatori che modificano un insieme di oggetti, e che vengono definiti operatori di aggregazione:

- combine operator: unisce in un unico oggetto punti che precedentemente erano separati e distinti; è identificato come un operatore 0-dimensionale, in quanto agisce su geometrie zero-dimensionali;
- merging operator: fonde 2 o più linee in un'unica linea, che normalmente viene posizionata a mezzeria; è un operatore 1-dimensionale, in quanto agisce su geometrie lineari;
- Amalgamation operator: fonde in un'unica geometria areale un gruppo di poligoni; è un operatore 2-dimensionale, in quanto agisce su geometrie areali;

- typification operator: riduce la complessità di un gruppo di oggetti attraverso la loro eliminazione, riposizionamento, allargamento o aggregazione mantenendo la disposizione tipica di quell'insieme di oggetti.

Operatori per la trasformazione degli attributi

Gli operatori di trasformazione degli attributi si differenziano da quelli geometrici in quanto non vanno a manipolare l'informazione geometrica dell'oggetto, ma agiscono sulla sua componente statistica. Ad esempio, una piantagione di latifoglie e conifere potrebbe essere trasformata in una foresta, a causa di una riduzione di scala che mette in secondo piano le caratteristiche degli alberi.

Gli operatori di questo tipo, identificati sempre da McMaster e Shea (1992), sono:

- classification operator: riduce i dati grezzi in un insieme di classi, operazione spesso necessaria in quanto non è sempre praticabile associare ad ogni singolo fenomeno un simbolo diverso per successivamente mapparlo;
- symbolization operator: codifica graficamente i dati e può essere applicato sia alla componente statistica sia a quella geografica.

Operatori e algoritmi

Gli operatori appena esaminati descrivono, di fatto, le singole operazioni che il cartografo compie per generalizzare la mappa. Ad ogni operatore, poi, corrisponde un algoritmo che implementa una trasformazione del dato geografico, anche se è facile imbattersi in più implementazioni dello stesso operatore, a seconda dell'oggetto su cui agisce. Ad esempio, l'operatore di semplificazione viene realizzato in modo diverso quando si applica ad una strada rispetto al caso in cui si applichi ad un edifico.

Ogni algoritmo è caratterizzato da una serie di parametri, che vanno impostati in funzione della scala, dell'oggetto a cui fanno riferimento, e in generale, del contesto. Lo stesso algoritmo di semplificazione, così, potrebbe essere applicato con parametri differenti a seconda che l'oggetto si trovi all'interno di un centro abitato, dove troppi dettagli tendono a creare confusione, oppure in una zona scarsamente abitata, dove non ci sono questo tipo di complicazioni. Il primo problema da affrontare è, quindi, la scelta dei parametri degli algoritmi e l'opportunità di variarli in funzione del contesto.

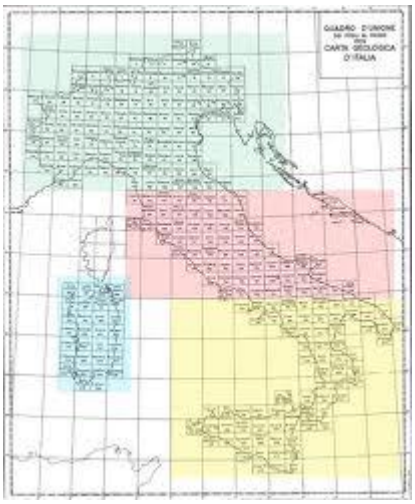
Il secondo problema, non meno importante, è l'ordine con cui gli algoritmi vengono eseguiti: il cartografo, per decidere la sequenza delle azioni da compiere e la loro coerenza con lo scopo della mappa o con il contesto, utilizza la sua conoscenza cartografica e la sua capacità di vedere sotto diverse astrazioni la realtà rappresentata.

Gli algoritmi che implementano gli operatori di generalizzazione sono, infatti, potenti strumenti per risolvere i vincoli spaziali, semantici e grafici; essi, però, non vanno utilizzati indiscriminatamente sull'intera area da generalizzare o sempre secondo la stessa sequenza

statica, ma vanno scelti, settati e applicati ogni volta diversamente, in funzione del contesto, della scala e dello scopo della mappa.

2.3 La situazione cartografica in Italia

In Italia, l'ente che si occupa di mantenere una cartografia aggiornata a livello nazionale è l'Istituto Geografico militare (IGM). La prima produzione cartografica dell'istituto è stata la Nuova Carta Topografica d'Italia, realizzata alla scala 1:100.000 (277 "fogli"). L'IGM ha prodotto diverse carte che coprono il territorio a livello nazionale e solo una parte di queste vengono mantenute aggiornate:



- per la scala 1:25.000 esistono le serie 25V, 25 e 25DB
- per la scala 1:50.000 le serie 50 e 50/L
- per la scala 1:100.000 le serie 100V e 100L.

Tra le varie serie sussiste un preciso rapporto matematico: ogni foglio di una mappa in scala 1:100.000 è divisa in quattro settori, rappresentati in altrettante mappe della serie 1:50.000; queste mappe vengono dette "quadranti", e ogni quadrante è a sua volta diviso in quattro parti, le "tavolette", che sono in scala 1:25.000.

Delle serie cartografiche alla scala 1:25.000, solo la più antica, la 25V (vecchio taglio), è stata completata. I dati di questa serie sono aggiornati mediamente al 1960, ad eccezione di alcuni elementi, che sono stati prodotti durante una campagna di aggiornamento parziale del 1984.

La serie 25 copre, dunque, solo il 36% del territorio nazionale; la sua produzione è stata interrotta per fare spazio alla nuova serie 25DB. Questa serie introduce la cartografia digitale, e, su un totale di 2298 sezioni, ne sono state realizzate 68, ottenute tramite stereorestituzione numerica o come derivazione dalla cartografia tecnica regionale numerica.

Oltre all'IGM, che lavora su cartografie con scala inferiore, da 1:25.000 in giù, vi sono altri enti che si occupano di cartografia e ai quali vengono affidate le realizzazioni di carte a media e grande scala (1:10.000 e superiore). Inizialmente era il Catasto a dover adempiere a questo compito; successivamente tale funzione è stata trasferita alle Regioni che hanno potuto così gestire in modo autonomo la creazione delle carte regionali, definite tecniche in quanto create specificamente per i tecnici delle amministrazioni.

Le carte prodotte dalle Regioni sono carte ricche di particolari e vengono aggiornate frequentemente, operazione facilitata dalla ristretta porzione del territorio nazionale che viene rappresentata. La CTR costituisce la base di riferimento per la redazione degli strumenti urbanistici comunali, per i Piani di Coordinamento Provinciali, per i Piani d'Area e per i vari piani di settore della pianificazione e della programmazione regionale.

In Veneto vengono prodotte due serie di Carte Tecniche Regionali (dette CTR): una in scala 1:10.000, composta da sezioni, e una in scala 1:5.000, composta da elementi. Il taglio e l'inquadramento le rendono sovrapponibili alle carte IGM di nuova produzione: in particolare, ogni foglio IGM in scala 1:50.000 è diviso in 4 quadranti della serie 25, ognuno dei quali ripartito in 4 sezioni in scala 1:10.000 della CTR, divisi a loro volta in 4 elementi in scala 1:5.000.

Recentemente anche gli uffici cartografici regionali hanno intrapreso la strada verso la digitalizzazione cartografica, iniziando la produzione di carte tecniche regionali numeriche (CTRN). Questo fatto, in concomitanza con il passaggio, da parte dell'IGM, dalla serie 25 alla 25DB, ha aperto un nuovo scenario nella cartografia italiana: la possibilità di usare la derivazione come mezzo per produrre la nuova serie 25DB.

Questa situazione permetterebbe quindi di limitare le rilevazioni dei dati sul territorio, e accelerare i tempi di produzione delle sezioni della serie 25DB. La realizzazione di una procedura informatica per la generalizzazione cartografica permetterebbe di accelerare ulteriormente i tempi, e darebbe quindi la possibilità di automatizzare il processo di derivazione.

Il progetto CARGEN nasce proprio per questo scopo, ovvero sviluppare un processo di generalizzazione delle carte tecniche regionali per produrre una base di dati coerente con il modello 25DB dell'IGM.

2.4 Il progetto CARGEN

Il progetto CARGEN, CARtographic GENeralization, è un progetto di ricerca, partito nel 2006, che vede coinvolti il Dipartimento di Ingegneria Informatica dell'Università di Padova, la Regione Veneto, e l'Istituto Geografico Militare. L'obiettivo di questo progetto è arrivare ad un processo di generalizzazione cartografica automatizzato applicabile alla produzione di mappe derivate. I dati di ingresso del processo sono quelli della Cartografia Tecnica Regionale (CTR), alla scala 1:5000, mentre la cartografia che si intende produrre è quella coerente con i modelli DB25 e DB50 dell'IGM.

La cartografia di partenza, da cui derivare i dati per il DB25, è quella della CTRN, fornita dalla Regione Veneto. Questa cartografia, viene utilizzata per popolare il GeoDBR, dal quale si derivano successivamente i dati per popolare il DB25.

Successivamente saranno descritti i tre modelli dati coinvolti nel progetto CARGEN, ovvero il modello CTRN, GeoDBR e DB25, e saranno presentati i risultati raggiunti all'interno del progetto.

CTRN

La Carta Tecnica Regionale Numerica è una cartografia generale e metrica, in formato vettoriale, prodotta dalla Regione Veneto. La carta, la cui produzione trova la sua principale fonte di dati nel rilievo fotogrammetrico, gode di campagne d'aggiornamento piuttosto frequenti ed offre quindi un dato geografico piuttosto recente e di buona qualità.

Le scale di rappresentazione adottate sono la scala 1:5.000 per la quasi totalità del territorio regionale e la scala 1:10.000 per le zone montane scarsamente urbanizzate. Gli oggetti e le informazioni territoriali contenute nella Carta Tecnica Regionale, acquisiti in forma vettoriale, sono organizzati in Livelli e Codici: i Livelli costituiscono una primaria classe di aggregazione degli oggetti, che a loro volta sono suddivisi nei Codici, relativi alle caratteristiche particolari di ciascun oggetto.

I dati della CTRN, però, non si prestano bene all'analisi spaziale e ad un diretto utilizzo, in quanto sono realizzati prevalentemente tramite tecniche CAD, e perciò non offrono alcuna forma di controllo di coerenza topologica. Questo fatto si ripercuote nella necessità di attuare una lunga fase di controllo e pulizia dei dati.

GeoDBR

Il GeoDBR è un modello dati sviluppato dalla Regione Veneto nell'ambito di un progetto per l'aggiornamento del proprio sistema informativo territoriale, realizzato secondo le specifiche definite all'interno del progetto IntesaGis. Aderendo alle specifiche IntesaGis, il GeoDBR si presta come perfetto tramite tra i dati della CTRN e quelli del DB25. Le specifiche di IntesaGis sono infatti mirate proprio alla definizione di un modello di dati che permetta una facile derivazione del DB25.

Il GeoDBR è un modello dati di moderna concezione, caratterizzato da una rappresentazione che prevede una divisione dell'informazione geografica in informazione di base e strati tematici, e la gestione di geometrie tridimensionali. Gli oggetti topografici sono rappresentati nel GeoDBR da feature inserite in un'organizzazione di livelli informativi a strati. Risulta così presente una forte componente gerarchica nella strutturazione del modello dati: le feature, prima che negli strati informativi, sono raggruppate in feature class e distinte tra loro tramite attributi.

Questo tipo di organizzazione è in contrapposizione con il modello dati del DB25 e della CTRN, dove gli attributi vengono usati più per fini descrittivi che per fini tassonomici.

DB25

Il DB25 è il modello dati creato dall'IGM per la compilazione della cartografia della serie topografica 25DB. Il DB25 contiene un modello di rappresentazione del mondo reale basato sulle feature, suddivise in oggetti semplici identificati dall'attributo LAB (Label).

Ogni feature è caratterizzata da una tipologia di primitiva grafica, evidenziata dalla prima lettera del codice: C per la tipologia areale, L per quella lineare, P per quella puntuale e T per le feature testuali.

A differenza del GeoDBR, l'organizzazione gerarchica delle feature del DB25 è minima: ogni oggetto topografico è di norma descritto direttamente da un codice LAB univoco. Questa scelta si spiega nella finalità tipografica di questo modello dati: ad ogni codice LAB è infatti associata anche una vestizione grafica, che viene usata nella stampa delle carte e riportata nella legenda delle cartografie in serie 25DB. Conseguenza di questa scarsa organizzazione gerarchica è la presenza, nel DB25, di più feature che afferiscono allo stesso oggetto reale; la scelta di quale tra queste feature utilizzare per l'oggetto si basa generalmente sui limiti di acquisizione: un esempio è l'oggetto edificio industriale, che, nonostante sia descritto dal medesimo codice FACC AC000 Processing Plant/Treatment Plant, viene rappresentato dalla feature Opificio Generico, con codice LAB C406 se la sua superficie è inferiore a 1500 metri quadri, oppure con la feature Stabilimento Industriale, LAB C405A, se la sua area risulta superiore a tale dimensione.

2.4.1 La derivazione del DB25

La parte del progetto CARGEN dedicata alla derivazione del DB25, condotta nei primi anni del progetto, è stata caratterizzata dalle seguenti tre fasi:

- creazione del GeoDBR e suo popolamento con i dati provenienti dalla CTRN in scala 1:5.000;
- analisi delle differenze tra i modelli di dati proprietari del GeoDBR e del DB25;
- la creazione del prototipo DB25.

La prima fase consisteva nel completamento delle specifiche del GeoDBR, nella creazione delle tabelle all'interno del DBMS Oracle Spatial 10g e nel loro popolamento.

La migrazione dei dati dal modello della CTRN a quello del GeoDBR si è scontrata in primo luogo con la diversa rappresentazione delle geometrie usate nei due modelli, e in secondo luogo con gli errori topologici dei dati originali. All'interno del progetto sono stati perciò sviluppati algoritmi per la trasformazione delle geometrie e per il controllo topologico, in modo da migliorare la qualità dei dati facilitando la successiva fase di derivazione del DB25.

La seconda fase, che aveva come obiettivo quello di trovare una corrispondenza tra le feature del GeoDBR e quelle del DB25, ha permesso di evidenziare alcune discrepanze e di conseguenza suggerire alcune migliorie ai due modelli.

La terza fase, consistente nella creazione del prototipo DB25, ha visto l'implementazione di un processo di generalizzazione cartografica, allo scopo di soddisfare i vincoli dettati dal passaggio di scala. Il processo di derivazione (Savino, 2007) è stato implementato in otto passi, utilizzando il DBMS Oracle per il mantenimento dei dati, Geomedia Professional 6 per l'elaborazione e l'export dei dati, e Dynamo/Dynagen per la fase di ricostruzione, acquisizione e generalizzazione dei dati.

L'analisi e la verifica dei risultati di quest'ultima fase hanno mostrato una carenza negli strumenti messi a disposizione dai software Geomedia, ed in particolare, Dynamo/Dynagen, che non è risultato sufficiente per attuare una generalizzazione completa e soddisfacente degli elementi geografici.

Questa analisi ha messo in luce una naturale distinzione degli elementi geografici in due gruppi: il primo, costituito da tutte le classi per cui la generalizzazione può essere ottenuta tramite le sole operazioni di selezione geometrica e/o spaziale descritte nel modello dati; il secondo, costituito da classi complesse, come la viabilità stradale, ferroviaria, l'idrografia e l'edificato, che richiedono una generalizzazione più specifica e accurata.

Tramite la procedura di popolamento si possono quindi generalizzare gran parte delle classi del modello dati, mentre, per le restanti, è richiesto lo sviluppo di algoritmi appositi per rendere queste classi complesse adatte alla rappresentazione in scala 1: 25.000 e compatibili con l'applicazione delle specifiche del modello dati. Questo fatto ha reso necessario lo sviluppo di un certo numero di algoritmi parametrici, scritti in Java, per risolvere alcuni importanti problemi di generalizzazione. In particolare, la viabilità stradale è stata semplificata mediante riconoscimento e generalizzazione degli incroci, identificazione e accorpamento delle carreggiate autostradali, e sfoltimento della viabilità secondaria; la rete fluviale tramite sfoltimento e semplificazione; l'edificato tramite amalgamento, selezione e tipificazione; infine, i binari all'interno delle stazioni ferroviarie sono stati generalizzati riducendo il loro numero.

Questi algoritmi sono stati inseriti in un processo di tipo batch, ovvero un processo che gli esegue in ordine sequenziale e predefinito. Il risultato finale del processo dipende perciò solo esclusivamente dai parametri scelti per i singoli algoritmi.

I risultati della generalizzazione del prototipo DB25 sono stati presentati in un convegno nazionale tenutosi presso l'Università di Padova nel 2009 (De Gennaro, 2009). A partire da questa data, gli obiettivi del progetto sono stati estesi alla derivazione del DB50.

2.4.2 La derivazione del DB50

La seconda fase del progetto CARGEN, quella relativa alla derivazione del DB50, ha avuto come primo obiettivo quello di realizzare un prototipo di modello dati per il DB50; per quest'ultimo, infatti, non è mai stato definito un modello dall'IGM. Nel febbraio 2010 è stato redatto un documento contenente l'analisi della derivabilità della Carta d'Italia IGM in scala

1:50.000, a partire dal database topografico IGM DB25 in scala 1:25.000, fornendo una proposta di modello dati per un database topografico alla scala 1:50.000.

2.4.3 Ambiente di lavoro

Il modello inizialmente adottato all'interno del progetto CARGEN per la gestione dei dati è stato quello client-server.

Il server è costituito da una macchina con installato un DBMS Oracle Spatial 10g, la cui funzione è quella di memorizzare e mantenere i dati spaziali, accessibili tramite query.

Nel lato client, invece, sono stati installati i software Geomedia Professional 6 e Dynamo/Dynagen, entrambi di proprietà della Intergraph. Geomedia viene utilizzato principalmente come strumento d'accesso ai dati spaziali; Dynamo/Dynagen sono invece gli strumenti usati durante il processo di generalizzazione cartografica.

Questa architettura, tuttavia, è stata in parte abbandonata recentemente: infatti, sia allo scopo di semplificare lo sviluppo di nuovi algoritmi, sia allo scopo di migliorare le prestazioni temporali, è stato cambiato metodo d'accesso ai dati ed il software per visualizzarli. I dati vengono caricati in RAM e gestiti proprio come se fossero delle tabelle, grazie ad una libreria sviluppata all'interno del Progetto. Inoltre, una libreria potentissima sviluppata in Java, la JTS (JTS Topology Suite), fornisce una vasta serie di operatori spaziali, evitando così di ricorrere al DBMS di Oracle per effettuare le interrogazioni spaziali.

Nei successivi paragrafi saranno brevemente descritti i software più importanti usati durante l'attività di tesi, e i dati geografici su cui sono stati effettuati i relativi test.

JTS Topology Suite

La JTS Topology Suite è una libreria open source, scritta interamente in Java, che fornisce una modellazione ad oggetti per le geometrie lineari in uno spazio euclideo. In questa libreria sono definite tre geometrie fondamentali, Point, LineString e Polygon, che rappresentano rispettivamente la geometria puntuale, lineare e areale.

La JTS mette a disposizione numerose funzioni geometriche, tra le quali possiamo citare:

- gli operatori topologici, che realizzano le funzioni di intersezione, differenza, unione;
- la funzione per la creazione del buffer intorno alla geometria;
- la funzione per la costruzione dell'inviluppo convesso;
- alcune funzioni per la semplificazione delle geometrie, come l'algoritmo di Douglas-Peucker;
- la funzione per la costruzione del Minimum Bounding Box.

Oltre a queste funzioni, la JTS fornisce l'implementazione di indici spaziali, come il quadtree, che offrono un modo veloce per la risoluzione di query spaziali. La versione utilizzata nell'ambito di questa tesi è la JTS 1.11.

OpenJump

OpenJump10 è un Desktop GIS open source che permette di visualizzare, modificare e interrogare dati spaziali.

OpenJump è scritto in Java, si basa sulla JTS ed è in grado di gestire file raster, vettoriali e database (PostGis, Oracle, ArcSDE); una caratteristica degna di nota è la sua architettura modulare, che permette di estendere di molto le funzionalità di base, potendo integrare, per esempio, il proprio codice mediante la realizzazione di un plugin.

In OpenJump, la creazione di un plugin, relativo al proprio codice, offre al programmatore il grosso vantaggio di poter visionare tramite l'interfaccia grafica gli effetti della propria applicazione. Il plugin diventa così uno strumento essenziale nello sviluppo di nuovi algoritmi che manipolano geometrie: avere una risposta grafica e istantanea è un grande aiuto per semplificare e velocizzare la fase di testing e debug.

La modalità con cui OpenJump gestisce le feature si basa sull'utilizzo dei layer, o livelli, che svolgono il ruolo di contenitori di feature. Ogni layer è in grado di contenere le feature relative ad uno specifico schema dati, chiamato FeatureSchema; quest'ultimo specifica il nome e la tipologia degli attributi che costituiscono la feature, simile a quanto accade nelle tabelle dei database. Un layer rappresenta, quindi, una vera e propria tabella, il cui schema dati è specificato dal FeatureSchema. Il layer è interrogabile per mezzo di query, che possono essere sia spaziali che non spaziali. Per migliorare le query spaziali, è possibile associare al layer uno degli indici spaziali forniti dalla JTS.

La versione di OpenJump utilizzata nello svolgimento di questa tesi è la 1.3.1, rilasciata nell'aprile 2009.

Capitolo 3

Il logging all'interno del progetto Cargen

Nei suoi primi anni di sviluppo, il progetto Cargen, era soggetto ad alcuni problemi strutturali: l'utilizzo del database Oracle per la gestione delle geometrie rallentava ampiamente le prestazioni degli algoritmi sviluppati, mentre l'uso di un GIS come Geomedia non permetteva una buona interattività con il programmatore.

Il cambio architetturale all'interno del progetto era quindi il primo passo da compiere per dare una svolta al lavoro di programmazione. Determinante è stata la decisione di gestire i dati direttamente in RAM abbattendo così i tempi di esecuzione, grazie anche all'introduzione della JTS Topology Suite; ma grande rilevanza va data anche all'utilizzo di un GIS open source come OpenJump.

L'impiego di un software proprietario come Geomedia non permetteva un'approfondita interazione con il programmatore, al quale bastava un semplice editor di testo per poter sviluppare i propri algoritmi in Java; ma la scelta di passare ad un GIS come OpenJump per la gestione grafica dei dati ha portato all'adozione quasi obbligata di un ambiente di sviluppo come Eclipse all'interno del progetto di ricerca Cargen.

Eclipse è un IDE open source sviluppato interamente in Java che permette di gestire progetti complessi in maniera facile e veloce grazie ai suoi potenti strumenti; il vantaggio principale che scaturisce dall'uso combinato di Eclipse ed OpenJump è la possibilità che ha lo sviluppatore di interagire direttamente con il sistema geografico tramite poche righe di codice. Tale interattività permette quindi la concezione e lo sviluppo di un'ampia gamma di funzionalità, sia per aiutare il programmatore a rendere il suo codice il più robusto possibile, sia per permettere all'utente finale di avere un'esperienza più completa del software di generalizzazione.

3.1 Logger: Considerazioni e scelte

Vista la possibilità di creare un "dialogo" diretto tra lo sviluppatore del processo ed il sistema geografico si è quindi deciso che il logger non deve limitarsi a compiere il suo compito di raccolta degli eventi, ma dev'essere uno strumento in grado di generare molto più che una serie di righe di testo. Inoltre, la possibilità di avere lo stesso logger all'interno di tutte le

classi del progetto, permette di realizzare un unico canale di comunicazione tra il processo di generalizzazione e l'esterno che porta ad una visione globale del sistema.

Partendo da queste considerazioni, molte sono state le idee per sviluppare nuove funzionalità all'interno del progetto, ma prima di andarle ad esaminare è giusto capire com'è stato pianificato il processo di logging.

Il logger è stato strutturato sulla base di 3 diversi canali di output: Console, File e Interfaccia Grafica; ma le annotazioni vengono loggate su 5 differenti destinazioni, in quanto della console vengono utilizzati sia lo standard output che lo stream degli errori, vengono inoltre utilizzati due file per registrare gli eventi.

Il primo file, denominato SystemLog, è destinato a contenere le annotazioni ritenute importanti dal programmatore e gli errori più gravi rilevati dal sistema.

L'altro file, chiamato dbLog, è incaricato di mantenere tutte le informazioni relative alle modifiche subite dalle geometrie e tutti gli errori riscontrati durante il processo di generalizzazione. Visto la mole di informazioni che il dlLog è destinato a contenere, le stringhe relative ai vari eventi vengono compresse prima di essere inserite nel file. Il dbLog è di fondamentale importanza all'interno di questo processo di logging in quanto permette di recuperare i dati necessari allo sviluppo di importanti funzionalità.

Non essendo la scrittura su console un metodo di conservazione permanente, questa viene utilizzata per assistere lo sviluppatore durante la fase di debug, per comunicare informazioni di poca rilevanza, oppure per segnalare gravi errori riscontrati durante il processo.

Ultimo output che rimane da presentare è la GUI (Graphical User Interface) , questo canale può essere utilizzato per comunicare direttamente con il sistema geografico OpenJump; ne deriva una grande funzionalità sia per operare una sorta di debug delle geometrie, che per comunicare graficamente con l'utilizzatore del software durante la fase di elaborazione.

La struttura del logger creato per il progetto Cargen può essere riassunta nella seguente tabella:

Logger's output	SystemLog [file]	- Annotazioni importanti - Errori più gravi
	dbLog [file]	- Modifiche apportate alle geometrie - Tutti gli errori riscontrati durante il processo
	StandardConsole	- Informazioni di debug - Informazioni di esecuzione di minore importanza
	GUI [OpenJump]	- Debug grafico delle geometrie - Comunicazioni grafiche
	ErrorConsole	- Errori più gravi

3.2 Funzionalità per lo sviluppo

Lo sviluppo di un software è la fase in cui viene dato vita al programma che si intende realizzare. Molte sono le difficoltà che si possono incontrare in questo processo, sia a causa di funzionalità complesse da creare sia per colpa di piccoli errori di stesura del codice che si possono ripercuotere sulla corretta realizzazione del programma.

Allo scopo di supportare il programmatore nello sviluppo di algoritmi efficienti all'interno del progetto Cargen sono state pensate alcune funzionalità che il logger dovrebbe essere in grado di garantire.

Debug del codice

Quella di debug è un'operazione che consiste nel localizzare, isolare e quindi risolvere gli errori presenti in un software. Mentre gli errori sintattici presenti all'interno del codice vengono facilmente individuati e corretti tramite l'utilizzo del compilatore, gli errori semantici sono invece più complessi da gestire. La presenza di questo tipo di bug può portare all'arresto improvviso dell'esecuzione del programma oppure ad un output non previsto dal programmatore. Per correggere questi errori si rende necessaria un'adeguata azione di debug che prevede la stampa su console di variabili, risultati, strutture dati o qualsiasi informazione che si ritenga utile a determinare la causa del malfunzionamento.

Compito essenziale del logger è quindi quello di mettere a disposizione del programmatore una funzione per realizzare l'output su console evitando di richiamare il metodo *System.out.print* per ogni segnalazione; in questo modo, una volta corretto il bug, le stampe sullo standard output possono essere evitate semplicemente disattivando la funzione di debug.

Debug di un ciclo

Per alcuni costrutti del linguaggio di programmazione si è considerata la possibilità di realizzare un'opera di debugging intelligente. Più precisamente, si è pensato di realizzare un metodo per loggare le informazioni presenti all'interno di un ciclo solo al realizzarsi di una determinata condizione di errore; in tal modo è possibile evitare di stampare in output troppe informazioni, che, invece di aiutare il programmatore, rischiano di metterlo in difficoltà.

Controllo di subroutine

La correttezza non sempre è l'unico parametro su cui si basa la riuscita di un algoritmo, molto spesso viene attribuito un grande peso all'efficienza temporale di elaborazione. All'interno di un processo come quello di generalizzazione dove si ha una quantità enorme di dati da

elaborare, quella dell'efficienza è una prerogativa alla quale il programmatore non può sottrarsi.

Il logger potrebbe quindi mettere a disposizione dello sviluppatore dei metodi che permettano di controllare il tempo di esecuzione di una subroutine ed il numero di elementi elaborati, consentendo così di monitorare i tempi di esecuzione delle singole funzioni, o di osservare in quale misura alcune modifiche apportate al codice possano incidere sulla velocità di calcolo.

Statistiche

Durante la realizzazione di un algoritmo, tra i programmatori del progetto Cargen, è prassi creare strutture di tipo Array per gestire piccole statistiche numeriche in modo da controllare quali effetti comporti l'esecuzione dell'algoritmo stesso. Esempi di queste statistiche possono essere il numero di elementi che sono stati classificati in un determinato modo oppure quelli che sono stati generalizzati secondo particolari specifiche o anche quanti oggetti siano stati eliminati dall'algoritmo.

La realizzazione di una struttura dinamica per immagazzinare, gestire e stampare i dati relativi ad ogni tipo di statistica può agevolare il lavoro del programmatore in fase di sviluppo del software.

Draw

Realizzare un processo automatico di generalizzazione cartografica è un obiettivo che porta ogni programmatore inserito all'interno del progetto Cargen a sviluppare degli algoritmi che modificano oggetti geometrici. L'utilizzo della console per il debug del codice realizzato non è quindi sufficiente a garantire la sua correttezza, si rendono necessarie continue verifiche dei risultati ottenuti direttamente su OpenJump.

Quello che è visibile sul sistema geografico è però soltanto il risultato finale ottenuto dall'algoritmo creato. L'alto grado di interazione tra il logger e OpenJump può permettere la realizzazione di un metodo per "disegnare" qualsiasi geometria intermedia generata dall'algoritmo durante la sua elaborazione; è quindi possibile ottenere una funzione in grado di rendere possibile il debug grafico delle geometrie.

3.3 Funzionalità per l'esecuzione

Un valido processo di logging si può ottenere combinando assieme un logger ben strutturato e, soprattutto, un suo intelligente utilizzo. Le chiamate al logger vengono infatti eseguite da chi programma il processo, spetta quindi allo sviluppatore capire quali e quante informazioni devono essere riportate, sia in funzione dell'output che si vuole ottenere, sia considerando i servizi che il log dovrà garantire.

Diverse sono le funzionalità che si sono pensate il logger possa mettere a disposizione dell'utente finale, alcune per tenere sotto controllo il processo di generalizzazione, altre, per permettere di capire come si sia giunti al risultato finale. A seguire vediamo quali idee hanno guidato l'implementazione del logger per la fase di esecuzione del programma.

Segnalazione errori e annotazioni importanti

Il ruolo principale del logger è quello di tenere traccia degli eventi che vengono generati durante la fase attiva del processo, è quindi scontato l'utilizzo di metodi che permettano di segnalare gli eventuali errori che si possono riscontrare. Oltre a questo, al programmatore, deve essere concesso di riportare sul log annotazioni che si ritengono importanti per capire come si sviluppi il processo di generalizzazione, sarà quindi necessario implementare un metodo che permetta di loggare tali informazioni.

Progress bar

Nonostante le novità architetturali apportate all'interno del progetto Cargen, quello di generalizzazione cartografica, rimane comunque un processo che impiega parecchio tempo per essere eseguito. Durante questa lunga fase, l'utente finale, non ha la possibilità di rendersi conto a quale punto sia arrivato il processo. Per questo motivo si è pensato all'introduzione di un metodo che, vista la possibilità di interagire con OpenJump, permetta al logger di generare una progress bar in grado di tenere aggiornato l'utente sullo stato di avanzamento del processo.

Zoom to

Con l'utilizzo di OpenJump, il processo di generalizzazione, è comandato direttamente all'interno del software tramite l'utilizzo di plug-in. Nel momento in cui viene dato via al processo, sul fondo del software GIS continua ad essere visualizzata la parte di mappa dove si è posizionati, ma, l'apertura di una finestra utilizzata da OpenJump per far comunicare il plug-in con l'utente, impedisce a questo di poter interagire con la mappa.

All’utente non è quindi permesso di spostarsi sulle varie geometrie per osservare da vicino quelle che sono se le evoluzioni portate dalla generalizzazione mentre questa è in corso. La capacità del logger di comunicare con il software OpenJump può consentire di spostare la visualizzazione in background sul software mentre il processo è in atto. Si può pensare di sviluppare una funzione in grado di zoomare su di un’area precisa della mappa ogni qual volta il metodo venga richiamato su di una geometria.

Tracking delle modifiche

Come detto nel capitolo precedente, il processo di generalizzazione realizzato all’interno del progetto Cargen, si sviluppa in otto passi principali che non andremo ora ad approfondire; quel che ci interessa realizzare è come avviene a grandi linee tale processo.

Il software sviluppato parte da un database contenente le geometrie riferite alla scala 1:5000, tali geometrie vengono riclassificate e modificate secondo quelle che sono le specifiche dettate dall’IGM in merito al cambio di scala, ed infine viene salvato il modello risultante su di un nuovo database. Al termine del processo l’utente potrà quindi visualizzare sia le geometrie di partenza che quelle risultanti dall’elaborazione, mentre non verranno riportate le geometrie intermedie in quanto non viene tenuta traccia di quelle che sono le modifiche indotte alla mappa nel pervenire al modello finale.

Non è proibitivo pensare di poter utilizzare il logger per effettuare il tracking di quelle che sono le modifiche apportate alle geometrie; in tal modo sarebbe possibile loggare ogni variazione subita dal modello. Nell’implementare una funzionalità così interessante bisogna però comprendere che il numero di modifiche che avvengono durante il processo può essere estremamente elevato, è quindi indispensabile prevedere un metodo efficace per registrare questi eventi. Si rende inoltre opportuno che ogni sviluppatore del software di generalizzazione sia accorto nell’utilizzare tale potenzialità così da costruire un tracking corretto e veritiero delle modifiche.

3.4 Funzionalità post-esecuzione

La funzione principale di un logger è quella di registrare, su diversi output, gli eventi e le informazioni più importanti relative ad un processo. Ciò che è stato omesso in questa definizione è l’importanza che acquisiscono i dati raccolti durante il processo di logging per la realizzazione di servizi aggiuntivi, capaci di dare all’utente una visione più approfondita dei risultati generati dal processo ed i problemi eventualmente riscontrati.

In funzione delle informazioni raccolte durante il processo di generalizzazione si è ritenuto opportuno creare due diversi servizi per l’analisi dei risultati ottenuti.

Servizio di Backtracking

Abbiamo visto al paragrafo precedente che riuscire ad implementare una struttura in grado di realizzare il tracking delle modifiche è un'operazione complessa. Questo lavoro sarebbe però sprecato se non venisse data all'utente finale la possibilità di osservare le trasformazioni subite dalle geometrie.

L'idea concepita per sfruttare al meglio le informazioni registrate durante il tracking delle modifiche è quella di creare un applicazione, da usare al termine dell'esecuzione del processo, in grado di rivisitare tutte le trasformazioni subite da una determinata geometria così da poterne osservare l'evoluzione compiuta durante la fase di generalizzazione.

Visione delle geometrie eliminate

In un processo di generalizzazione alcune geometrie vengono eliminate per ottenere un risultato che soddisfi le specifiche richieste. Un'applicazione in grado di mostrare quali feature sono state scartate dal database finale può mettere chiarezza all'utente su come si sviluppi la fase di selezione.

Raccolta degli errori

Non essendoci un canale di output deputato a registrare tutti e soli gli errori riscontrati durante il processo, sarebbe utile avere a disposizione un servizio in grado di estrarre queste informazioni dal file destinato al database. In questo caso sarebbe possibile effettuare un'analisi completa dei problemi riscontrati durante il processo di generalizzazione.

Capitolo 4

Implementazione del logger e dei servizi post-processo

Dopo aver introdotto quali siano state le idee che hanno guidato la realizzazione di un logger da utilizzare all'interno del progetto Cargen, in questo capitolo, vengono descritte le classi ed i metodi principali che ne hanno permesso l' implementazione.

Le classi costruite sono state organizzate in due diversi package così da distinguerne il ruolo principale. Nel primo package si trovano le classi che permettono la realizzazione di un corretto processo di logging, nel secondo sono invece presenti le classi che sfruttano le informazioni di log, prodotte in fase di esecuzione, per mettere a disposizione dell'utente adeguati servizi di analisi dei risultati.

4.1 Il package Logging

Il logger da utilizzare all'interno del progetto Cargen è stato costruito estendendo il package standard messo a disposizione dal framework java.util.logging, la scelta tra questa architettura e quella distribuita da Apache con Log4J non è stata cruciale in quanto entrambe presentano una struttura simile e mettono a disposizione gli stessi strumenti di base.

La struttura del logger, creato per supportare le funzionalità richieste dal progetto Cargen, si fonda su 6 diverse classi racchiuse all'interno del package Logging, queste sono:

- CargenLogger
- LogConsoleOutHandler
- LogFileOutHandler
- LogFormatterFactory
- LogOpenJumpOutHandler
- LogUtility

La classe principale è naturalmente CargenLogger che rappresenta l'API tramite cui il programmatore può comunicare con il logger. Dei sette livelli di base messi a disposizione dal framework standard di java si è ritenuto sufficiente utilizzarne soltanto cinque. Molti di questi

sono stati rinominati per renderne più esplicito il ruolo ricoperto; di seguito si riporta la nuova gerarchia dei livelli utilizzati nel CargenLogger:

java.util.logging	*CargenLogger*	*Utilizzo*
SEVERE	**ERROR**	Livello usato per segnalare la presenza di errori pericolosi che possono compromettere la corretta riuscita del processo
WARNING	**WARNING**	Usato per avvertire della presenza di potenziali situazioni di pericolo
INFO	**TXT**	Livello per i messaggi generati in fase di sviluppo, utile ad eseguire la fase di debug
CONFIG	**DBSTORE**	Usato per i messaggi destinati ad essere riportati su di una tabella del database
FINE	**SYSTEM**	Livello utile a riportare le informazioni più interessanti del processo di generalizzazione
FINER		
FINEST		

Un oggetto di tipo CargenLogger può essere istanziato tramite l'utilizzo della funzione `getCargenLogger()` della classe LogUtility, questo è un metodo statico, il che significa che il logger può essere inserito in qualsiasi classe senza doverne passare un riferimento da classe a classe. Nell'invocare il metodo `getCargenLogger()` c'è la possibilità di specificare il nome da assegnare al nuovo logger, i nomi assegnati sono poi disposti su di una struttura gerarchica proprio come per la classe util.logging.Logger.

Le informazioni generate durante il processo di logging vengono distribuite su 5 differenti output, sono infatti 5 gli Handler utilizzati all'interno della classe CargenLogger ed appartengono a tre diverse classi:

- LogConsoleOutHandler
- LogFileOutHandler
- LogOpenJumpOutHandler

Due degli Handler utilizzati sono di tipo LogConsoleOutHandler, classe che consente di dirigere i messaggi di log sulla console; il primo Handler scrive direttamente sullo standard output mentre l'altro utilizza lo stream di output dedicato agli errori.

Altri due degli Handler utilizzati da CargenLogger sono invece di tipo LogFileOutHandler ed utilizzano due diversi file per riportare le informazioni di log. Il primo file prende il nome di dbLog ed è destinato a contenere le informazioni che dovranno poi essere riportate su un database, l'altro, systemLog, è invece il file di sistema sul quale vengono riportati i dati più importanti del processo. Entrambi i file presentano nell'intestazione del nome la data e l'ora in cui vengono creati così da poterli conservare senza il rischio di sovrascriverli.

Un ultimo Handler è invece del tipo LogOpenJumpOutHandler, classe che permette di utilizzare un'interfaccia di OpenJump per poter comunicare con il programma stesso.

Ad ogni Handler viene poi associata una particolare formattazione da applicare al messaggio di log sul rispettivo canale di output, per far questo viene utilizzata la classe LogFormatterFactory che permette di ottenere 4 diverse impaginazioni per l'informazione di log.

Come detto nel paragrafo 1.3.1 ad ogni logger e ad ogni handler può essere associato un filtro per definire quali livelli possano essere loggati sui canali di destinazione. La classe LogUtility mette a disposizione il metodo `createFilter( Level livelloSoglia, int mode)` che permette di incanalare verso l'output il solo livello di soglia oppure i livelli maggiori, minori o diversi da questo in base alla modalità selezionata.

Le caratteristiche appena descritte, riguardanti un oggetto del tipo CargenLogger, sono state raccolte nella seguente tabella dove per ogni livello vengono indicati gli handler associati e gli output a cui sono destinati i messaggi di log.

CargenLogger			
	ERROR	FileHandler ConsoleHandler OpenJumpHandler	ErrorConsole dbFile systemFile OpenJump
	WARNING	FileHandler	dbFile
	TXT	ConsoleHandler	StandardConsole
	DBSTORE	FileHandler	dbFile
	WARNING	FileHandler	systemFile

4.1.1 Funzioni base del logger

Come descritto nel precedente capitolo molte sono state le idee che hanno guidato lo sviluppo di un logger adeguato al suo utilizzo all'interno del progetto di ricerca Cargen. Le funzioni di seguito descritte sono quelle basilari per assistere lo sviluppo e l'esecuzione del processo di generalizzazione.

Debug

L'importanza di un intelligente processo di debug è già stata sottolineata più volte, utilizzare il metodo `system.out.print()` per stampare su console le informazioni utili a monitorare la correttezza del codice non è la soluzione più adatta. Nel momento in cui il programma viene stabilito essere corretto si rende necessario scandire tutto il codice per pulirlo da queste stampe divenute superflue.

Il CargenLogger permette di utilizzare il metodo `txt(String msg)` che invia un messaggio di log al livello TXT il quale, tramite il ConsoleHandler, scrive il messaggio msg direttamente sullo standard output della console.

Ciò che un oggetto di tipo CargenLogger permette di fare è di rimuovere o di assegnare il ConsoleHandler al livello TXT; in questo modo, usando i metodi `removeConsoleHandler()` e `enableConsoleHandler()`, è possibile disabilitare e riabilitare la scrittura su console a piacimento.

Controllo di subroutine

Il controllo del tempo di esecuzione di una parte di codice è una prassi ricorrente per capire il peso temporale di un determinato algoritmo all'interno di un programma.

La classe CargenLogger mette a disposizione il metodo `subroutineStart(nome)` per avviare un "cronometro virtuale" che registra il tempo di inizio della subroutine. Di subroutine ne possono essere istanziate senza limiti di numero, con l'unica accortezza di utilizzare nomi diversi se non se ne vuole resettare una precedentemente avviata.

Durante la fase di esecuzione dell'algoritmo al quale una subroutine fa riferimento è possibile registrare il numero di elementi elaborati utilizzando il metodo `subroutineItera(nome)`; i risultati sul tempo di esecuzione e numero di iterazioni possono essere ottenuti grazie al metodo `subroutineStop(nome)` che da la possibilità di loggare tali informazioni su console o su file di sistema in base all'importanza assegnata alla subroutine.

Statistiche

L'uso adeguato di semplici strutture dati permette ai programmatori di mantenere delle statistiche in grado di generare piccoli report riguardanti il flusso di un particolare algoritmo. In particolare, per tali dati, vengono utilizzati degli array di interi le cui celle rappresentano ognuna una statistica diversa, ad esempio, una locazione può contenere il numero di elementi elaborati dall'algoritmo, oppure il numero delle geometrie che lo stesso algoritmo ha classificato in un modo anziché classificarle in un altro. Ma l'utilizzo di un array, comporta innanzitutto una certa staticità nel numero di statistiche mantenibili, e, soprattutto, costringe il programmatore a sfruttare la sua memoria per associare ogni cella ad una determinata statistica.

La classe CargenLogger mette a disposizione dello sviluppatore una struttura dinamica per la gestione delle statistiche, che permette di rifersi ad ogni statistica tramite il suo nome.

Il metodo `stat( name, val )` permette di incrementare il valore associato ad una statistica oppure, nel caso questa non esistesse, di crearla assegnandole un valore iniziale; la funzione `statGet( name )` restituisce invece il valore associato alla statistica.

Il metodo `statPrint( name )` permette di stampare il valore di una statistica, ma, se il nome inserito come parametro dovesse avere la forma `*prefix`, vengono stampate tutte le statistiche con prefisso `prefix`. In questo modo il programmatore può associare lo stesso prefisso alle statistiche relative allo stesso algoritmo semplificando l'operazione di report dei valori.

Annotazioni importanti

Durante la fase di esecuzione può essere necessario dover annotare alcuni comportamenti e risultati intermedi generati dal processo, queste informazioni possono riguardare il numero delle trasformazioni eseguite da un particolare algoritmo di generalizzazione, la descrizione del comportamento del processo o qualsiasi altro dato che possa essere ritenuto interessante per capire come la generalizzazione è stata realizzata.

Il CargenLogger mette a disposizione del programmatore il metodo `system(String msg)` con il quale è possibile scrivere un'annotazione sul file di sistema systemLog. Anche in questo caso il flusso di scrittura può essere disabilitato e riabilitato in qualsiasi punto del codice utilizzando le funzioni `removeFileHandler()` e `enableFileHandler()` che agiscono sul fileHandler associato al livello SYSTEM.

Segnalazione errori

Qualsiasi programma può sollevare un'eccezione durante la sua esecuzione ed è giusto che il programmatore provveda a gestirla nel migliore dei modi. Per capire il comportamento di un processo, oltre alle annotazioni, è giusto che anche gli errori riscontrati vengano loggati

adeguatamente. A questo proposito sono stati creati due diversi livelli per la segnalazione degli errori: WARNING e ERROR.

Un errore dannoso per il processo in corso che rischia di bloccarne l'esecuzione dev'essere segnalato tramite il metodo `error()` della classe CargenLogger. Vista la sua gravità, questa informazione è destinata ad essere riportata su svariati output: oltre al file di sistema vengono coinvolti anche il file dbLog, la console degli errori e la finestra di output testuale di OpenJump. Errori meno gravi devono invece essere segnalati tramite il metodo `warning()` e sono destinati a comparire solamente nel file che verrà poi riportato su database, in questo modo all'interno di tale file compaiono tutti gli errori riscontrati durante il processo.

4.1.2 Interazioni con OpenJump

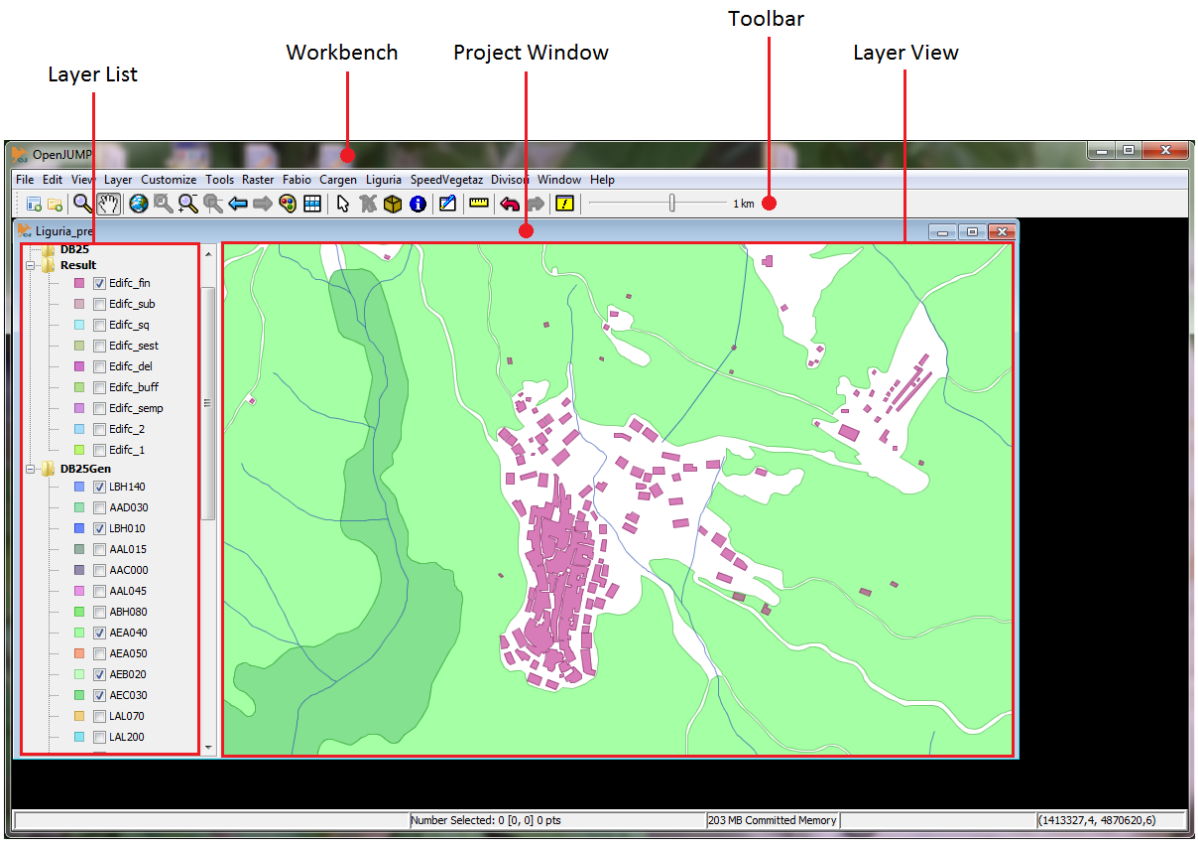
OpenJump è un GIS open source che, grazie alla sua architettura modulare, permette a qualsiasi programmatore di estenderne le funzionalità di base integrandole con il proprio codice tramite l'utilizzo di semplici plug-in.

Nel momento in cui viene inizializzato un plug-in all'interno di OpenJump, questo si impossessa di una variabile di tipo PlugInContext che gli viene passata direttamente dal software GIS. La variabile `context` permette al programmatore di agire direttamente sul workbench di OpenJump e cioè sulla finestra di lavoro aperta, in tal modo, viene data al programmatore la possibilità di realizzare le stesse azioni che si possono compiere direttamente sull'interfaccia del software utilizzando opportune righe di codice su Eclipse.

Per sfruttare al meglio i vantaggi derivanti dall'alto grado di interazione, il CargenLogger è stato costruito in modo tale da poter instaurare un dialogo diretto con il software OpenJump.

La classe LogUtility permette infatti di creare un oggetto di tipo CargenLogger usando il seguente costruttore `LogUtility.getCargenLogger( String name, PlugInContext context )` in questo modo viene data la possibilità di assegnare al logger sia un nome che, soprattutto, un `context` che gli permette di interagire con OpenJump.

In alternativa, per assegnare un context ad un oggetto CargenLogger, è possibile utilizzare il suo metodo dinamico `setUseOpenJumpHandler()`.



Una volta che il logger è in possesso del `context` del plug-in è possibile utilizzare i metodi `drawGeom()` e `zoomTo()` per realizzare un processo di debug grafico.

drawGeom()

Invocando il metodo `drawGeom( Geometry geom )` della classe CargenLogger il programmatore ha la possibilità di visualizzare una geometria su OpenJump anche se questa non risulta dal processo di generalizzazione. In questo modo si possono creare dei Layer che permettano di giudicare la bontà dei cambiamenti apportati da un algoritmo alle geometrie della mappa. Quel che si arriva ad ottenere è quindi un processo di debug grafico che affianca lo sviluppatore nella stesura del codice.

zoomTo()

Con il metodo `zoomTo( Geometry geom )` viene data la possibilità al programmatore di spostare la vista dei layer di OpenJump andando a fare uno zoom sulla zona dove compare la geometria indicata.

Questo metodo si rende utile durante l'esecuzione di un plug-in, in questa fase, infatti, l'interfaccia non è utilizzabile perché bloccata dalla comparsa di una finestra, chiamata monitor, il cui scopo è quello di riportare informazioni di esecuzione all'utente.

La funzione `zoomTo( Geometry geom )` crea un buffer attorno alla geometria inserita come parametro e va ad impostare la visualizzazione sul buffer stesso, al programmatore viene quindi data la possibilità di spostare la vista in background per seguire l'evolversi del processo in determinati punti di interesse.

4.1.3 Il tracking delle modifiche

La struttura del processo di generalizzazione automatica all'interno del progetto Cargen non da la possibilità di mantenere in memoria quelle che sono le modifiche intermedie subite dagli oggetti geometrici. Al termine della generalizzazione si possono visualizzare solamente le tabelle contenenti le geometrie finali.

Tener traccia delle modifiche subite dalle geometrie può permettere di valutare attraverso quali passi si sia arrivati ad ottenere il risultato finale. Quello che è stato realizzato è quindi un importante strumento di analisi del processo di generalizzazione.

Di seguito vengono presentati i problemi incontrati durante la realizzazione del servizio di tracking e le decisioni prese per risolverli.

Operazioni su geometrie

Primo aspetto sul quale si è reso necessario fare chiarezza è stato capire quali modifiche subisce una geometria durante il processo di generalizzazione e di quali valga la pena tenere traccia. Per aiutarci nel ragionamento è stato preso in considerazione come punto di partenza il modello degli operatori geometrici definito all'interno del progetto AGENT e già presentato nel capitolo 2.2.3. Ne riportiamo uno schema riassuntivo:

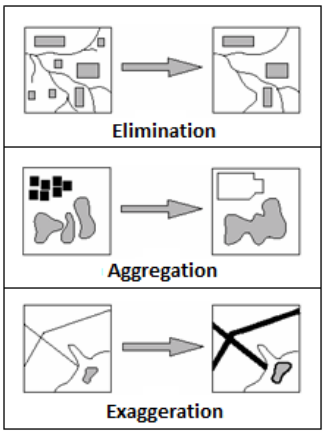
Operatori geometrici della generalizzazione (Progetto AGENT)	**Individuali**	Simplification	Riduce la granularità dei contorni di linee e aree, mantenendo un numero minore di punti per rappresentare l'oggetto.	
		Collapse	Trasforma un oggetto N-dimensionale in uno a N-1 o N-2 dimensioni. Es: oggetti areali in punti o linee.	
		Enhancement	Enlargement	Aumento delle dimensioni di un oggetto mantenendo le proporzioni
			Exaggeration	Ingrandimento di una parte dell'oggetto
			Smoothing	Addolcimento del contorno di una geometria
			Squaring	Utilizzo di una forma rettangolare per rappresentare la geometria
	Individuali o su di un insieme	Selection / Elimination	Selection	Selezione degli elementi importanti della mappa
			Elimination	Eliminazione degli oggetti ritenuti non importanti o ridondanti
		Displacement	Spostamento di posizione di un oggetto o di un gruppo mantenendone inalterata la forma	
	Su di un insieme (Aggregation)	Amalgamation	Fusion	Fondere in un'unica geometria areale un gruppo di poligoni
			Merge	Fondere 2 o più linee in un'unica linea, che normalmente viene posizionata a mezzeria
		Combine	Unire in un unico oggetto, elementi che precedentemente erano separati e distinti	
		Typification	Ridurre la complessità di un gruppo di oggetti attraverso la loro eliminazione, riposizionamento, allargamento o aggregazione mantenendo la disposizione tipica di quell'insieme di oggetti.	

Il processo di tracking di una geometria nasce allo scopo di poter ricostruire l'evoluzione delle trasformazioni subite da un oggetto geometrico al termine del processo di generalizzazione. Partendo da questo presupposto si è quindi considerato inappropriato tener conto di tutte le trasformazioni geometriche suggerite dal modello AGENT.

Per le operazioni di modifica che coinvolgono un insieme di oggetti sono stati mantenuti i concetti espressi dalla tabella, ma i quattro operatori proposti sono stati ridotti e rivisti per avvicinarsi alle esigenze dei programmatori:

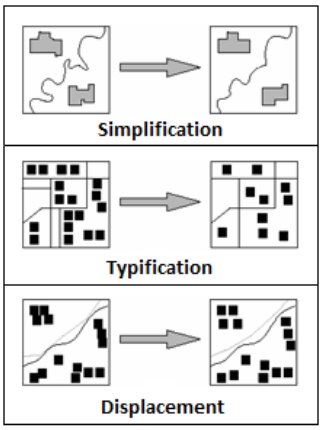
- L'operatore Typification è stato mantenuto con il suo significato originale

- Gli operatori Fusion e Merge sono stati sostituiti dall'operatore Union che rappresenta l'unione tra due o più geometrie connesse
- L'operatore Combine è stato invece sostituto dall'operatore Aggregation che rappresenta l'aggregazione di un insieme di oggetti non necessariamente connessi



Degli operatori geometrici applicabili sia ad un insieme di geometrie che singolarmente, il Displacement e l'Elimination sono stati mantenuti con il loro significato originale, mentre l'operazione di Selection è stata scartata in quanto risulta scontato che gli oggetti geometrici che permangono al termine del processo di generalizzazione sono stati selezionati. Si è invece deciso di tenere traccia della creazione di un nuovo oggetto così da poterne seguire le evoluzioni future.

Gli operatori di modifica applicabili a singole geometrie presentati nel modello sono invece stati ritenuti essenziali e quindi totalmente mantenuti.



Dopo aver definito quelli che sono gli operatori della generalizzazione di cui tenere traccia si è ritenuto importante capire come riconoscere una geometria in modo univoco. L'oggetto geometrico che viene modificato durante il processo di generalizzazione è soltanto un attributo di una feature, altri attributi appartenenti alla feature sono di certo un ID ed un codice LAB, ma possono essercene anche altri. Ogni feature appartiene ad una tabella che altro non è che una collezione di feature; su OpenJump le tabelle sono definite come Layer. Per riconoscere in modo univoco una geometria è quindi sufficiente identificarla con l'ID della feature e la tabella a cui questa appartiene.

Nel rispettare quelle che sono le specifiche IGM per il processo di generalizzazione, le feature, possono essere spostate da una tabella ad un'altra; si è reso quindi necessario l'introduzione di una nuova funzione di tracking in grado di tracciare il cambio di tabella, questa operazione è stata chiamata TableChange.

Gli operatori presentati riescono a descrivere le operazioni che vengono eseguite durante il processo di generalizzazione, ma in alcuni casi il programmatore potrebbe voler dare un nome diverso ad una serie di interventi operati su di una geometria. Nasce quindi il bisogno di lasciare al programmatore la possibilità di coniare nuovi nomi per definire alcune operazioni utilizzate.

Nome dei metodi di tracking

Un buon processo di logging può essere ottenuto solamente se il programmatore utilizza in modo corretto il logger che gli viene messo a disposizione, è quindi importante agevolare il

lavoro dello sviluppatore cercando di creare delle chiamate al logger con nomi intuitivi e semplici da inserire all'interno del codice.

In una prima versione del logger si era pensato di utilizzare un unico metodo editGeom() per racchiudere tutte le chiamate ad operazioni su geometrie delle quali bisognava tenere traccia, ad ogni operazione era stato associato un codice numerico che doveva essere inserito come parametro del metodo, proprio come nell'esempio:

```
editGeom( String table, int id, int codiceOperazione, ……)
```

In questo modo non era innanzitutto permesso coniare delle nuove operazioni e, soprattutto, il lavoro del programmatore veniva complicato in quanto gli era richiesto di imparare il codice associato ad ognuna delle operazioni.

Si è quindi passati ad utilizzare un metodo diverso per ognuna delle operazioni che devono essere tracciate, per distinguere questo tipo di chiamate al logger dalle altre, è stato utilizzato un prefisso comune "track" ad indicare che la funzione serve appunto a tracciare le trasformazioni della geometria. Tra questi metodi ne è stato creato uno generico grazie al quale è possibile assegnare il nome desiderato all'operazione usata.

Nel creare i metodi di tracking è stata però riposta anche attenzione al non definire un numero troppo elevato di funzioni che potrebbero creare confusione nel programmatore, le operazioni di Enhancement sono quindi state raccolte in un unico metodo dove l'utilizzo di un numero intero permette di definire quale delle quattro operazioni è stata utilizzata.

I metodi di tracking utilizzati sono riassunti nella seguente tabella:

Operazioni di tracking		
trackSimplification	Riduzione del numero di punti della geometria	
trackCollapse	Trasformazione da geometria areale a lineare o puntuale	
trackShapeChange	type 1: Enlargement	Ingrandimento in scala della geometria
	type 2: Exaggeration	Ingrandimento di una parte dell'oggetto
	type 3: Smoothing	Addolcimento dei contorni
	type 4: Squaring	Rappresentazione tramite rettangolo
trackDisplacement	Spostamento della geometria sulla mappa	
trackElimination	Eliminazione di una feature	
trackUnion	Unione di due o più oggetti connessi	
trackAggregation	Aggregazione di un insieme di oggetti non necessariamente connessi	
trackTypification	Riduzione della complessità di un gruppo di oggetti	
trackTableChange	Cambio di tabella della feature contenente la geometria	
trackCreateGeom	Creazione di una nuova geometria	
trackGenericOperation	Operazione generica alla quale è possibile associare un nome	

Ognuno dei metodi presentati, fatta eccezione per trackShapeChange, trackGenericOperation e trackCreateGeom, accetta in ingresso gli stessi parametri che sono:

- **table** (String)
- **id** (int)

  Chiave per identificare la feature la cui geometria è stata modificata.

- **newTable** (String)
- **newId** (int)

  Chiave da specificare se l'operazione dovesse comportare anche un cambio di tabella

- **note** (String) — Nota riguardante l'operazione
- **geom** (Geometry) — Geometria risultante dall'operazione

Per il metodo trackShapeChange è richiesto di inserire anche un numero intero per identificare il tipo di operazione eseguita, per la funzione trackGenericOperation è invece necessario aggiungere il nome della nuova operazione, mentre per trackCreateTable viene richiesta soltanto una coppia di identificazione per delineare a quale tabella la geometria geom appartiene e con quale ID appare.

Memorizzazione dei dati nel dbLog

Le annotazioni di tracking generate durante l'esecuzione del processo di generalizzazione devono essere registrate e mantenute adeguatamente per permetterne poi un opportuno utilizzo. Queste informazioni sono destinate a comparire all'interno di un database così da poter essere elaborate con maggior facilità tramite l'utilizzo di semplici query; ma la scelta di creare un stram diretto tra il logger e la base di dati non è stata ritenuta vantaggiosa.

L'uso di una tabella, pur sembrando la scelta più ovvia, non si avvicina alla necessità di mantenere nel tempo le informazioni di log che hanno portato alla creazione di una mappa; sulla base di questo ragionamento si è quindi deciso di registrare le informazioni di tracking all'interno di in un file di testo, un supporto molto più semplice da gestire e da mantenere. Il file destinato a contenere queste annotazioni all'interno CargenLogger è il dbFile, dove sono presenti anche i dati di log relativi agli errori sollevati durante il processo.

Il tracking delle modifiche subite da un oggetto geometrico viene eseguito allo scopo di poter ripercorrere l'evoluzione della feature durante il processo, è dunque importante che le informazioni raccolte da un'operarazione di tracking permettano di mantenere un legame con le altre modifiche apportate ad una stessa geometria. Le variabili richieste come parametro di ingresso dai metodi di tracking sono state appositamente studiate per consentire di mantenere tale legame.

Una riga di log del dbFile relativa alla modifica di una geometria compariva inizialmente nella seguente forma:

tabella; id; nuovaTabella; nuovoId; nomeOperazione; nota; geometria.toString()

Naturalmente un oggetto del tipo Geometry non può essere riportato in un file di testo, si è però riusciti a superare questo ostacolo utilizzando il metodo `toString()` messo a disposizione dalla classe Geometry che permette di riportare una geometria nel formato testuale WKT (Well Known Text).

Con un log così strutturato si rende quindi possibile conoscere quale modifica è stata apportata alla geometria e capire se questa è stata spostata da una tabella ad una nuova.

Ma il numero di operazioni compiute sugli oggetti geometrici durante il processo di generalizzazione è molto elevato, si rendeva quindi necessario comprimere il più possibile il carico di dati immagazzinati nel file, sia per scongiurare il pericolo di generare un log troppo pesante, che per evitare di rallentare il processo di generalizzazione visto l'overhead che la scrittura su file comporta.

Compressione dei dati

Per cercare di ridurre la quantità di dati da immagazzinare si è pensato di rappresentare sia i nomi delle tabelle che i nomi delle operazioni con un codice numerico, ma questo codice doveva però essere dinamico vista la possibilità di usare nuovi nomi per indicare le operazioni e in considerazione del fatto che non si può richiedere al programmatore di assegnare un codice numerico ad ognuna delle tabelle che saranno poi utilizzate durante il processo di generalizzazione.

Nel risolvere questo problema ci sono venute incontro due strutture dati molto potenti il cui uso combinato permette di realizzare un codice dinamico, tali strutture sono l'ArrayList e le HashMap.

Un ArrayList è un array rappresentato mediante una lista, un oggetto di questo tipo ha una capacità che può variare in base al numero di oggetti inseriti al suo interno e, proprio come un array, permette l'accesso ad un oggetto in una certa posizione della lista.

Un oggetto di tipo HashMap gestisce invece un insieme di coppie <chiave,valore>, il suo scopo principale è quello di rendere veloci ed efficienti operazioni quali inserimento e ricerca di elementi a partire dalla loro chiave.

Nel nostro caso sono stati utilizzati:

- ArrayList contenenti oggetti di tipo String, in grado quindi di consentire l'accesso ad una stringa a partire dal suo indice
- HashMap definita da oggetti di tipo <String, int> così da permettere l'accesso al numero intero a partire dalla sua chiave di tipo String.

Quel che viene riportato di seguito è una parte di codice in cui viene utilizzata la struttura appena descritta per legare il nome di un'operazione geometrica ad un numero che la rappresenti.

```
listaOperations = new ArrayList<String>();
mappaOperations = new HashMap<String,Integer>();

private int getOperationCode(String operation)
      {
            int codeOp;
            if (mappaOperations.containsKey(operation))
                  codeOp = mappaOperations.get(operation);
            else
            {
                  codeOp = listaOperations.size();
                  listaOperations.add(operation);
                  mappaOperations.put(operation, codeOp);
            }
            return codeOp;
      }
```

In queste poche righe di codice possiamo vedere come sia possibile ottenere un codice numerico `codeOp` da associare all'operazione `operation`. Il processo contrario potrà essere eseguito accedendo alla cella di indice `codeOp` della lista `listaOperations` così da ottenere il nome dell'operazione.

Una volta realizzata la compressione dei nomi delle operazioni e delle tabelle si è pensato di identificare le coppie [table, id] e [newTable, newId] tramite l'utilizzo di un unico numero di tipo int.

Il tipo di dato primitivo int è un numero compreso tra -2^{31} e $2^{31}-1$ che occupa 4 byte di memoria e cioè 32 bit, ciò che si è deciso di fare è stato utilizzare i primi 10bit più significativi di un int per rappresentare il numero associato alla tabella, mentre gli ultimi 22 bit servono ad indicare l'id. Con questa configurazione, tramite un unico numero, è quindi possibile riconoscere 1024 tabelle diverse con associate 2^{22} feature ciascuna; numeri in grado di evitare qualunque tipo di collisione tra dati.

Tale operazione viene eseguita all'interno del metodo `getHashCode()` della classe CargenLogger, se ne riporta la parte più significativa:

```
int getHashCode(String tab, int id){

      ……

      int codeTab; // codeTab contiene il numero intero associato alla tabella

      // Shifto a sinistra i 10bit meno significativi di codeTab
      int temp = codeTab << 22;

      // Creo la maschera per selezionare i 22 bit meno significativi di id
      String mask = "00000000001111111111111111111111";
      int maskNumber = Integer.valueOf(mask, 2).intValue();
      int selectedId = id & maskNumber;

      // Concateno codeTab[10bit] ad id[22bit] con un'operazione OR
      int finalCode = temp | selectedId;
      return finalCode;
}
```

Ultima informazione da comprimere rimaneva a questo punto l'espressione testuale della geometria. Come già detto, il metodo `toString()` di una geomety restituisce una stringa in formato WKT, cioè, un linguaggio di markup per rappresentare geometrie che però occupa molti caratteri. Per ovviare a questo problema è stato utilizzato un linguaggio equivalente in grado però di comprimere l'informazione, il Well-Known Binary (WKB).

In seguito alle modifiche descritte, una riga del dbFile riferita ad un'annotazione di tracking, si presenta quindi nel seguente modo:

codiceTableId; codiceNewTableNewID; codiceOperazione; nota; geometria[WKB];

4.1.4 Versione finale del file dbLog

Come descritto nei precedenti paragrafi, all'interno del file dbLog, compaiono le annotazioni di log generate durante l'esecuzione del processo di generalizzazione e riguardanti sia gli errori riscontrati che le operazioni di modica degli oggetti geometrici.

Per quel che riguarda l'inserimento di dati relativi agli errori generati, la gestione delle informazioni è stata simile a quella utilizzata per i metodi di tracking; sia per il nome delle operazioni che per i nomi delle eccezioni si è utilizzata la combinazione di ArrayList ed HashMap così da comprimere l'informazione e creare una struttura dinamica per la gestione degli errori da inserire.

Una linea del dbLog riferita ad un errore riscontrato durante il processo ha quindi questo aspetto:

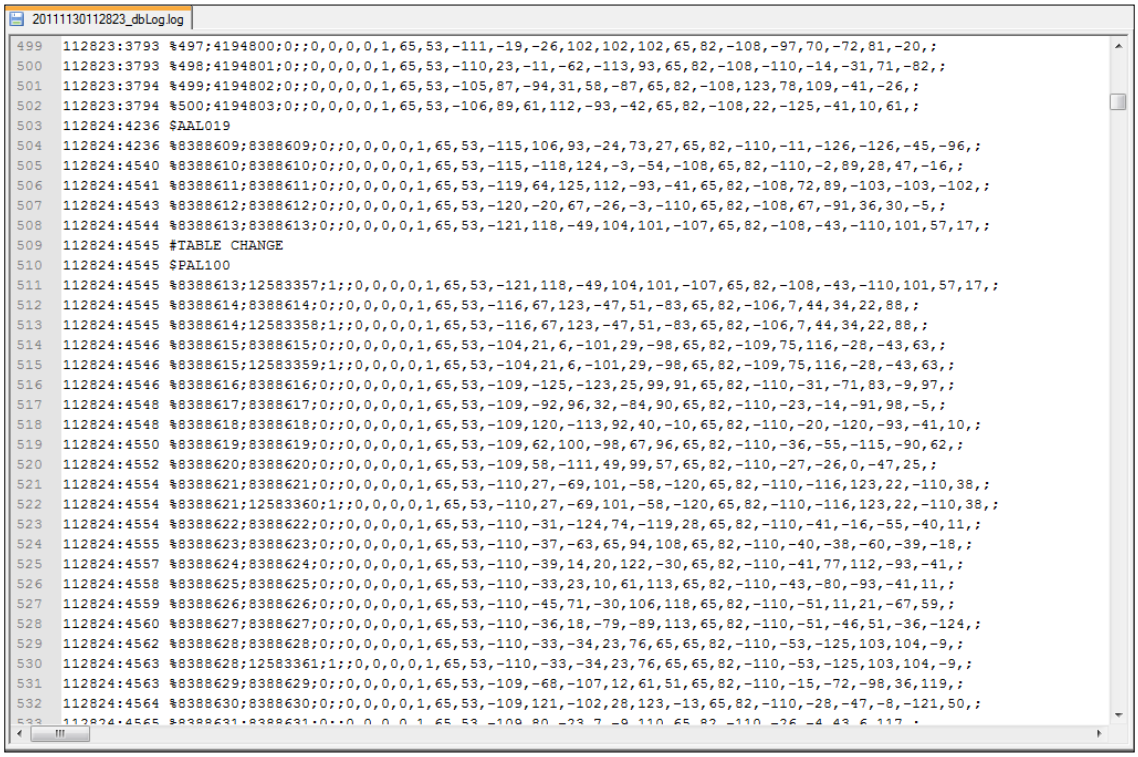
tipo; codiceEccezione; codiceOperazione;

dove il tipo sta ad indicare se si tratta di un ERROR o di un WARNING.

Il file dbLog è destinato ad essere letto, decompresso ed esportato su tabelle del database; per agevolare queste operazioni ogni riga immessa nel file inizia con un carattere speciale che ne determina il contenuto:

Special chars in dbFile	
$	Nome di una nuova tabella utilizzata durante il processo
#	Nome di una nuova operazione
!	Nome di una nuova eccezione
%	Informazioni di tracking(modifica di geometrie)
£	Informazioni su di un errore riscontrato

La formattazione applicata al livello DBSTORE prevede che ogni riga inserita nel file inizi con il timestamp relativo al momento dell'inserimento. Il seguente è un esempio del log che viene riportato sul file dbLog:



4.2 Il package LoggingService

Una fase di logging ben eseguita permette all'utente di analizzare in modo completo i risultati del processo, se questi è però in possesso degli strumenti adatti. Il package LoggingService contiene le classi necessarie a generare un eccellente servizio post-esecuzione per quanto riguarda il processo di generalizzazione.

Il package si compone di 4 classi principali di cui 2 sono plug-in di OpenJump:

- LogTableManager
- LogTableUtil
- BacktrackingPlugIn
- ErrorFilePlugIn
- ShowEliminatedPlugIn

I plug-in contenuti in questo pacchetto vogliono mettere a disposizione dell'utente due differenti servizi: uno per generare un file di testo contenente tutti i soli errori riscontrati durante il processo e quello più importante che permette di visualizzare le modifiche di una geometria al termine del processo di generalizzazione.

LogTableManager

Lo scopo di questa classe è quello di mantenere un legame tra il file dbLog destinato ad essere analizzato e la tabella del database all'interno della quale verranno riposte le informazioni del file.

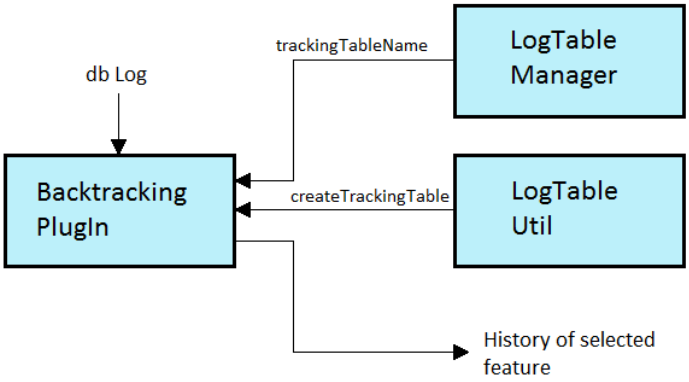
Come detto in precedenza, ogni file originato dal processo di generalizzazione ha come prefisso del nome, il giorno e l'ora in cui è stato creato, in questo modo non esistono conflitti di omonimia tra file; viene quindi data la possibilità di mantenere in memoria un file per ogni processo di generalizzazione eseguito.

Entrambi i plug-in presenti nel package, quando eseguiti, prendono in ingresso un file dbLog, estraggono le righe a cui sono interessati e le trascrivono in una tabella all'interno del database così da poterle utilizzare per il servizio. La classe LogTableManager si assicura di mantenere un riferimento tra il nome del file dbLog e le due tabelle che da questo possono essere generate, in tal modo si evita di eseguire il trasferimento dei dati da file a DB ogni volta che un plug-in viene avviato sullo stesso file.

4.2.1 Backtracking service

Scopo del plug-in di backtracking è quello di visitare tutte le modifiche subite da una geometria durante il processo di generalizzazione così da poter analizzare i risultati ottenuti.

Il plug-in richiede innanzitutto all'utente di selezionare il file dbLog relativo al processo di generalizzazione che si vuole esaminare, dopo esserne in possesso, si rivolge alle altre due classi del package.



Tramite la classe LogTableManager il servizio può ottenere il nome della tabella destinata a contenere le informazione del file proprio come visto nel precedente paragrafo.

Alla classe LogTableUtil spetta invece il compito di trasferire le informazioni del file all'interno dell'apposita tabella del database, per far questo utilizza il metodo `createTrackingTable()` . Questa funzione scorre una ad una tutte le righe del file di log e ne esamina innanzitutto il carattere speciale che le distingue: se la riga è di interesse per la ricostruzione delle informazioni viene elaborata, altrimenti la riga viene ignorata.

Le strutture di dati utilizzate per tradurre le informazioni compresse nel file sono le stesse usate dalla classe CargenLogger e cioè ArrayList ed HashMap, in questo modo ogni volta che il metodo legge il nome di una nuova tabella utilizzata o di una nuova operazione è in grado di associarle lo stesso codice numerico con cui era stata immagazzinata all'interno della stringa di log. Una volta eseguita la decodifica, la classe LogTableUtil, è in grado di decomprimere e trasferire i dati del file nella tabella del DB, questo processo impiega un tempo proporzionale alla lunghezza del file e, richiederne l'esecuzione ogni volta che viene selezionato lo stesso file dbLog, sarebbe una procedura errata; tale situazione viene però evitata grazie alle funzionalità già descritte della classe LogTableManager.

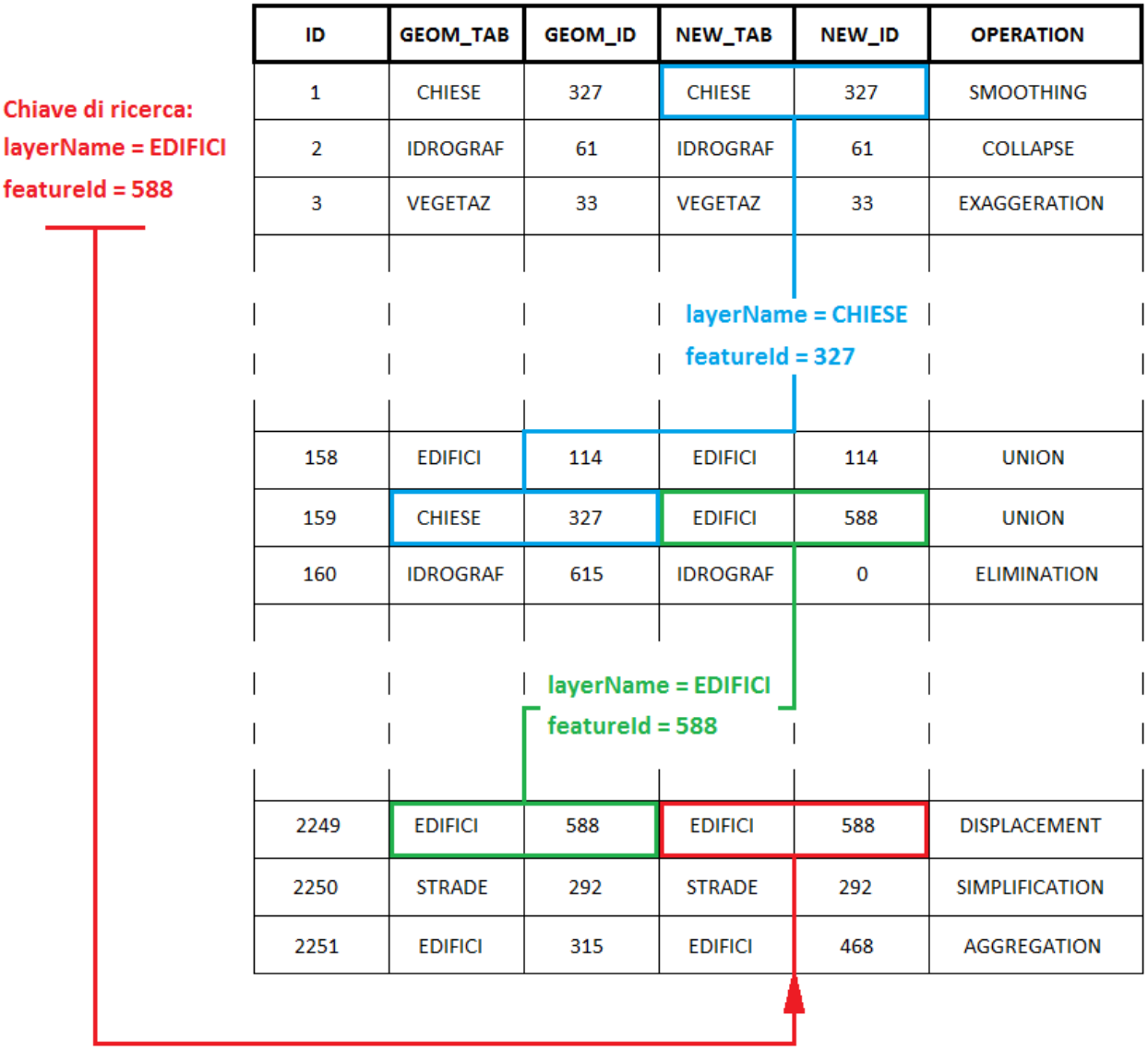
Una riga appartenente alla tabella contenente le informazioni di tracking è caratterizzata da 8 attributi:

ID	GEOM_TAB	GEOM_ID	NEW_TAB	NEW_ID	OPERATION	DESCRZ	GEOMETRY

- ID: Attributo chiave della tabella di tracking
- GEOM_TAB, GEOM_ID: Identificano la feature prima di subire la modifica
- NEW_TAB, NEW_ID: Identificano la feature dopo la modifica, se non c'è stato un cambio di tabella avremo GEOM_TAB, GEOM_ID = NEW_TAB, NEW_ID
- OPERATION: Nome dell'operazione
- DESCRZ: Eventuali note aggiunte dal programmatore
- GEOMETRY: Geometria risultante dalla modifica

Una voltra popolata la tabella di tracking all'interno del database, il plug-in è in grado di generare una history delle modifiche subite da una qualsiasi delle geometrie selezionate dall'utente tramite l'utilizzo del metodo `generateBacktracking()`.

Questa funzione estrae dalla feature selezionata su OpenJump il suo ID ed il nome del Layer di appartenenza(cioè la tabella) e li utilizza per effetture la ricerca delle modifiche inizializzando le variabili *layerName* e *featureId*.



ID	GEOM_TAB	GEOM_ID	NEW_TAB	NEW_ID	OPERATION
1	CHIESE	327	CHIESE	327	SMOOTHING
2	IDROGRAF	61	IDROGRAF	61	COLLAPSE
3	VEGETAZ	33	VEGETAZ	33	EXAGGERATION
158	EDIFICI	114	EDIFICI	114	UNION
159	CHIESE	327	EDIFICI	588	UNION
160	IDROGRAF	615	IDROGRAF	0	ELIMINATION
2249	EDIFICI	588	EDIFICI	588	DISPLACEMENT
2250	STRADE	292	STRADE	292	SIMPLIFICATION
2251	EDIFICI	315	EDIFICI	468	AGGREGATION

L'algoritmo realizzato parte dal fondo della tabella e la scorre a ritroso alla ricerca di una riga che abbia valori NEW_TAB = *layerName* e NEW_ID = *featureId*; nel momento in cui si ha un riscontro positivo, viene generata in output su OpenJump la geometria associata alla riga e viene continuata la ricerca sulla tabella, tenendo però in considerazione un eventuale cambio di tabella, quindi:

- Se i valori NEW_TAB, NEW_ID della riga coincidono con quelli presenti in GEOM_TAB, GEOM_ID la ricerca continua verso l'alto con gli stessi valori *layerName* e *featureId*.
- Se invece GEOM_TAB, GEOM_ID hanno valori differenti da NEW_TAB, NEW_ID, le variabili di ricerca vengono modificate in tal modo *layerName* = GEOM_TAB, *featureId* = GEOM_ID. La ricerca prosegue quindi con i nuovi valori.

L'algoritmo continua l'analisi della tabella fino ad arrivare al suo inizio; l'output delle informazioni su OpenJump è stato strutturato in modo tale che, ad ogni geometria visualizzata sul layerView corrisponde un layer che ne descrive il nome della tabella di appartenenza, l'id, e la modifica che ha subito.

4.2.2 Recupero delle geometrie eliminate

Tra i vari metodi di tracking, l'utilizzo di trackElimination permette di avere una notifica ogni volta che una geometria viene eliminata dal modello finale della generalizzazione. Le informazioni riguardanti tale operazione compaiono, assieme a quelle riguardanti le altre operazioni di tracking, nel file compresso dbLog.

Il plug-in che recupera le geometrie eliminate, proprio come quello di backtracking, si rivolge alle classi LogTableManager e LogTableUtil per ottenere la tabella del DB riferita al file dbLog selezionato. Una volta in possesso del riferimento alla tabella, il plug-in, riporta su OpenJump tutte le geometrie corrispondenti alle righe in cui il valore dell'attributo NEW_ID è pari a zero.

Questo perché il metodo `trackElimination()` genera una riga di log del tipo:

tab; id; new_tab; 0; nota; geometry;

4.2.3 Error file generator

Le scelte in merito alla gestione degli output all'interno del CargenLogger fanno si che sia gli errori gravi che i warning, vengano registrati sul file dbLog. In questo modo è possibile avere una raccolta di tutte le eccezioni sollevate durante l'esecuzione del processo, ma queste informazioni sono compresse. La classe ErrorFilePlugIn permette di estrarre i dati relativi ad ogni errore annotato nel file compresso e di riscrivere tali eccezioni su di un file di testo, nello stesso ordine in cui queste eccezioni sono state catturate; viene così consentita l'analisi dei problemi incontrati dal processo di generalizzazione eseguito.

Capitolo 5

Test Eseguiti

5.1 Cartografia utilizzata

Pur nascendo dalla collaborazione tra il dipartimento di ingegneria dell'informazione dell'Università di Padova e la regione Veneto, il progetto CARGEN, cerca di esportare i risultati di ricerca ottenuti oltre i confini regionali, proponendosi presso altri enti per risolvere il problema della generalizzazione.

La partecipazione al bando per la generalizzazione automatica della cartografia in scala 1:25000 della regione Liguria è stato un pretesto per creare un processo fondato sulla nuova architettura del progetto CARGEN. Ad oggi è questo l'unico processo creato che può essere comandato da un plug-in di OpenJump e che elabora i dati direttamente in RAM senza l'ausilio di un Database topologico; per tal motivo il processo preparato ad hoc per la generalizzazione della Liguria è stato utilizzato per testare le funzionalità del CargenLogger.

I dati cartografici utilizzati per i test si riferiscono ad una zona rurale con ampia presenza di vegetazione(vigneti, frutteti, boschi), si nota poi la presenza di piccoli paesi, una rete stradale non complessa e qualche tratto fluviale di poca portata.

5.2 Struttura del processo di generalizzazione

L'intero processo di generalizzazione risiede all'interno del package Liguria, la classe principale capace di comandare l'intero processo è GeneralizzazionePlugIn, al suo interno è possibile settare ciò che si vuole venga generalizzato.

Le opzioni possibili sono le seguenti:

- `genFiumi( db25v )`
- `genStrade( db25v )`
- `popolaPonti( db5v, db25v )`
- `genPontiCollassa( db25v )`

- `genAreeNaturali( db25v )`
- `genCippi( db25v )`
- `genPali( db25v )`
- `genTerrazzamenti( db25v )`
- `genMuri( db25v )`
- `genVegetazioneSimple( db25v )`
- `genRuderi( db25v )`
- `genBaracche( db25v )`
- `genTettoie( db25v )`
- `genEdifici( db25v )`
- `genAreeToPoint( db25v )`

Ognuno di questi metodi risiede appartiene alla classe GeneralizzazioneClassi, è al suo interno che risiedono le procedure da applicare alle feature perché queste rispettino le specifiche espresse dall'IGM.

Il processo di generalizzazione si sviluppa utilizzando una serie di algoritmi eseguiti secondo una determinata sequenza, nel nostro caso le API utilizzate sono state:

- `genRuderi( db25v )`
- `genBaracche( db25v )`
- `genTettoie( db25v )`
- `genEdifici( db25v )`
- `genAreeToPoint( db25v )`

Possiamo osservare che il processo applica una generalizzazione leggera che coinvolge principalmente gli stabilimenti presenti nella carta geografica.

Le specifiche dettate dall'IGM per la generalizzazione di ruderi, baracche e tettoie sono piuttosto semplici da applicare in quanto si tratta di trasformazioni semantiche o, al più, di operazioni di collapse per tramutare le aree in punti.

L'algoritmo di maggior interesse è senza dubbio quello per la generalizzazione degli edifici. Al suo interno, come prima cosa, vengono richiamati i metodi

- `genEdifCimiteri( )`
- `genChiese( )`
- `genCampanili( )`

in questo modo vengono portati nella tabella degli edifici tutti i cimiteri, le chiese ed i campanili che ne hanno diritto secondo le specifiche.

Una volta ottenuta la tabella totale degli edifici, avviene l'unione tra tutte le geometrie che hanno almeno un vertice in comune; fase, questa, seguita da interventi per rimuovere possibili buchi dalle geometrie risultanti ed altre operazioni di semplificazione per rendere il risultato finale più adeguato alla rappresentazione su carta.

Le altre classi contenute all'interno del package Liguria sono di supporto agli algoritmi descritti in precedenza.

5.3 Utilizzo del CargenLogger

Per creare un processo di logging è stato necessario innanzitutto creare una variabile statica di tipo CargenLogger all'interno di ogni classe coinvolta dal processo di generalizzazione. Affinché le diverse classi utilizzassero lo stesso logger per la notifica degli eventi, è stato fondamentale trasmettere a tutte l'istanza del logger definito all'interno della classe che comandava il processo, cioè, GeneralizzazionePlugIn.

Le chiamate effettuate durante il processo sono state rivolte soprattutto alla standard console ed al file dbLog. Su console è stato possibile testare la correttezza di funzionalità quali subroutine e statistiche e l'utilità della rimozione del consoleHandler, impedendo al logger di riportare annotazioni sullo standard output.

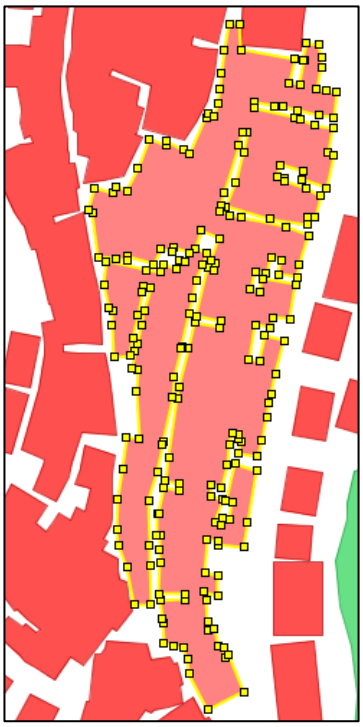
Per quanto riguarda la segnalazione degli errori e dei warning, non tutte le eccezioni all'interno delle classi erano gestite, è stato quindi necessario apportare alcune modifiche al codice per poter utilizzare correttamente i metodi `warning()` ed `error()`.

Gli eventi ai quali è stata riposta maggior importanza sono di certo le operazioni di modifica delle geometrie che dovevano essere tracciate; in una prima fase di studio degli algoritmi si è però notato, all'interno del codice, un forte utilizzo di strutture del tipo ArrayList per creare collezioni di geometrie o di ID; alcuni dei metodi di tracking sono quindi stati arricchiti così da garantirne la compatibilità con tali strutture ed agevolare i programmatori nel loro.

Le chiamate di tracking messe a disposizione dal logger sono state inserite senza difficoltà ed hanno comportato minimi ritocchi al codice originale. Un meccanismo importante osservato all'interno degli algoritmi è stato l'utilizzo di tabelle temporanee all'interno delle quali venivano spostate alcune feature a causa di particolari modifiche. Si è quindi reso indispensabile applicare il metodo `trackTableChange()` a tutte le feature spostate. Senza questa accortezza si rischiava di troncare la tracciabilità di una geometria, impedendo così al servizio di backtracking di compiere un'analisi completa delle modifiche.

All’interno del codice sono state inserite anche alcune chiamate a metodi per testare il grado di interazione tra il logger ed OpenJump; il metodo drawGeom() si è dimostrato molto importante per garantire un debug grafico delle geometrie. La funzione zoomTo() dev’essere invece utilizzata con cautela, la velocità di elaborazione rischia di spostare continuamente la visualizzazione in background di OpenJump, rendendo vano il tentativo di osservare l’evolvere del processo.

5.4 Risultati ottenuti

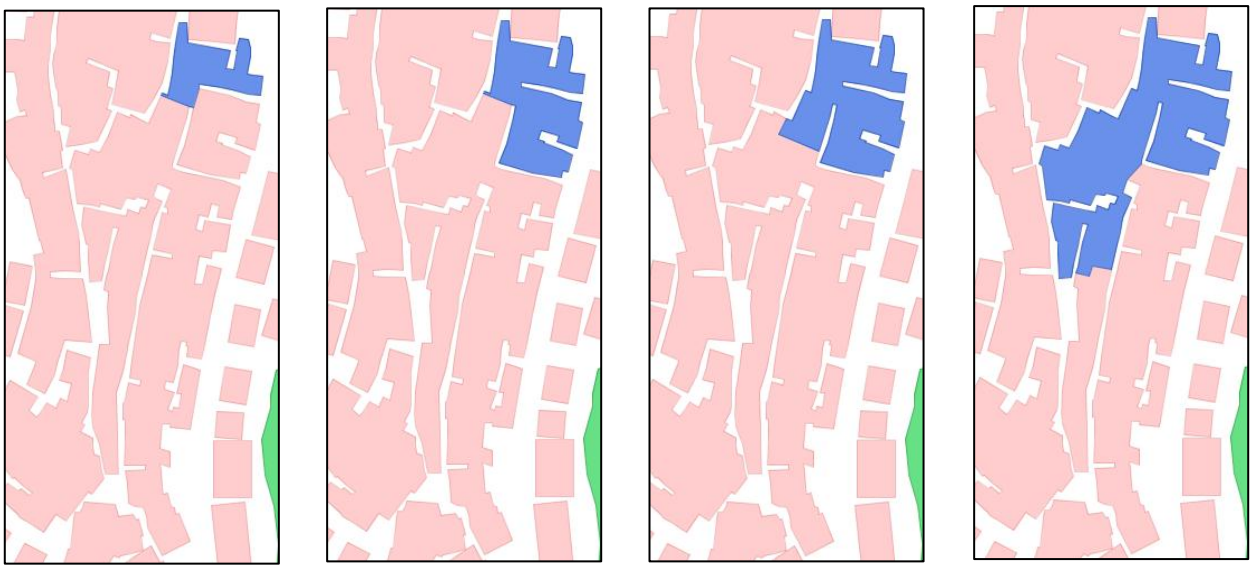


In questa parte si riportano i risultati ottenuti con l’uso del plug-in di backtracking.

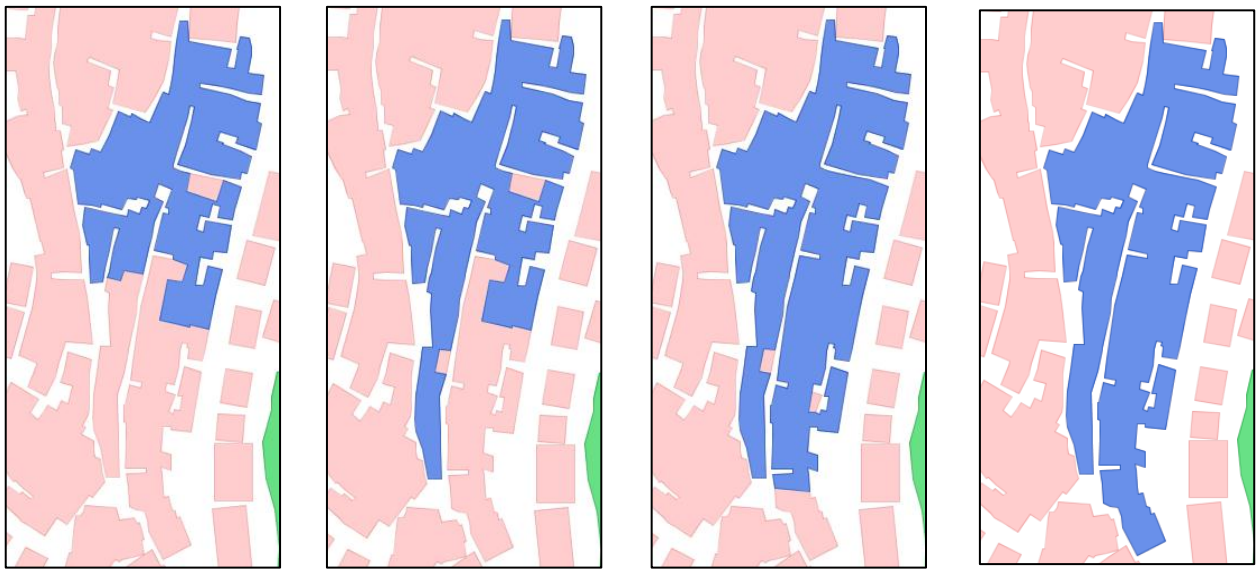
Quel che appare nell’immagine a sinistra è una parte della mappa al termine del processo di generalizzazione, dove le geometrie in rosso rappresentano gli edifici.

L’analisi dei risultati tramite il servizio di backtracking permette di capire quali operazioni l’oggetto selezionato abbia subito per giungere alla sua rappresentazione finale.

Come detto nel paragrafo precedente, l’algoritmo di generalizzazione interviene sugli edifici unendo quelli che si toccano in almeno un punto; nelle immagini a seguire possiamo osservare come si evolva l’opera di unione delle geometrie:



(a) (b) (c) (d)



(e) (f) (g) (h)

I risultati ottenuti con questo strumento di analisi sono quelli che ci si aspettava, è ora possibile valutare l'opera di generalizzazione nelle sue fasi di sviluppo e non più dal confronto tra lo stato iniziale e quello finale del processo.

Conclusioni

Questo lavoro di tesi si poneva come obiettivo quello di realizzare un servizio di logging da mettere a disposizione dei programmatori del progetto CARGEN per gestire lo sviluppo, l'esecuzione e soprattutto l'analisi del processo di generalizzazione automatico.

Le API per l'utilizzo del logger sono state sviluppate tenendo in forte considerazione la praticità d'uso delle invocazioni, un logger deve infatti essere un supporto alla programmazione e non un peso aggiuntivo; sono inoltre state utilizzate strutture dinamiche per la realizzazione dei metodi, in questo modo ogni programmatore può avere un modo personale di utilizzare le funzionalità messe a loro disposizione.

Il logger è stato testato su di un processo di generalizzazione completo, in tal modo è stato possibile verificarne l'utilità in fase di esecuzione e soprattutto l'importanza degli strumenti di analisi. Il servizio di backtracking riesce infatti a permettere lo studio del processo nelle sue fasi evolutive, la sua riuscita è però legata ad un uso adeguato delle operazioni di tracking.

Nello sviluppo delle funzionalità del logger è stata fatta particolare attenzione a non incidere negativamente sulle prestazioni del processo, a questo scopo è stata fondamentale l'opera di compressione dei dati all'interno del file dbLog, visto l'alto numero di informazioni che vengono riportate al suo interno. Nel file non si è però riusciti a comprimere ulteriormente le informazioni relative alle geometrie, nonostante questo i file di log generati da varie esecuzioni del processo sono dell'ordine di qualche MB.

Varie misurazioni effettuate durante i test di generalizzazione automatica hanno appurato che l'uso delle operazioni di tracking applicate hanno comportato un overhead trascurabile rispetto al tempo di esecuzione dell'intero processo.

Bibliografia

CERT-In (2008) Security Guidelines for Auditing and Logging. Department of Information Technology of India.

Sandro Savino (2007) Il processo di generalizzazione cartografica: dalla Carta Tecnica Regionale al DB25 IGM". Tesi di ricerca. Padova – Dipartimento DEI: Università degli Studi di Padova - Dipartimento di Ingegneria dell'Informazione.

De Gennaro M., Rumor M., Savino S. (2009) Le procedure per la derivazione del DB25 dal DBT della Regione del Veneto: risultati del progetto CARGEN Bollettino della Associazione Italiana di Cartografia, 135, Aprile 2009.

Deruda G, Falchi E, Falchi U. e Vacca G. (2005) La generalizzazione cartografica automatica in ambiente GIS Bollettino SIFET 2/2005.

Enciclopedia Italiana Grolier (1987) Vol. 4, Cartografia, pp 293 – 296

Robinson A. H., Sale, R. e Morrison J. L. (1978) Elements of Cartography. New York: Wiley & Sons.

IGM (2004) Norme e Segni convenzionali per la realizzazione dei fogli della Carta d'Italia alla scala 1:50 000 Istituto Geografico Militare, Firenze.

Vivid Solutions (2003) JTS Topology Developer'S Guide, Version 1.4 URL: http://www.vividsolutions.com/jts/jtshome.htm