ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA
SCUOLA DI SCIENZE

Corso di Laurea in Ingegneria e Scienze Informatiche

INTEGRAZIONE DEI FORMATI DI NAVIGAZIONE GPS STANDARD IN ALCHEMIST

Relazione finale in
Programmazione ad Oggetti

Relatore:
Prof. MIRKO VIROLI

Co-relatore:
Ing. DANILO PIANINI

Tesi di Laurea di:
ANDREA PLACUZZI

ANNO ACCADEMICO 2016–2017
SESSIONE II

PAROLE CHIAVE

Alchemist

GPS

Geographic Information System

GPX

Simulazione

Indice

Introduzione

Lo scopo di questo progetto di tesi è il rinnovamento del modulo di geographic information system (GIS) del simulatore di Alchemist, rinnovamento reso necessario dalla necessità di poter supportare molteplici formati standard di tracce di Global Positioning System (GPS), come ad esempio i formati GPX e KML. Il supporto a formati standard e l'abbandono dell'attuale formato proprietario ha lo scopo di semplificare l'utilizzo di questo modulo da parte di tutti gli utenti, nonché consentire di utilizzare diverse fonti per recuperare tracce GPS. La tesi è stata suddivisa in capitoli secondo le fasi in cui è stato suddiviso il lavoro:

- Il primo capitolo racchiude una descrizione del simulatore Alchemist. In particolare si tratta il modello del dominio del simulatore, la realizzazione delle simulazioni e quali sono le funzionalità di GIS che attualmente espone.
- Nel secondo capitolo si presenta cosa siano i tracciati GPS, come vengano creati, quali sono i formati attualmente più utilizzati per lo scambio di queste informazioni, e da dove possono essere reperite.
- Nel terzo capitolo sarà esposto tutto il lavoro svolto per l'integrazione dei formati di navigazione GPS standard in Alchemist, partendo dalla fase di analisi dei requisiti, passando per la fase di progettazione, arrivando all'implementazione, e terminando con una guida che descriva l'utilizzo della nuova parte aggiunta.
- L'ultimo capitolo riassume il lavoro svolto, esponendo un giudizio e le possibili evoluzioni future del modulo di GIS.

Capitolo 1

Alchemist

Alchemist [17] è un meta-simulatore estensibile nato come progetto dell'università di Bologna ispirato dagli algoritmi di simulazione stocastica per la (bio) chimica, adattato ed evoluto al fine di supportare la simulazione di applicazioni pervasive in cui esseri umani e dispositivi interagiscono opportunamente per fornire e sfruttare servizi d'informazione.
Attualmente Alchemist è utilizzato per l'esecuzione di simulazioni in diversi ambiti: simulazioni di evacuazione e guida di folle [2, 14], simulazioni di autocomposizione di servizi [13], valutazioni di pattern di auto organizzazione [10, 19, 22], simulazioni di aggregate computing [3, 16, 20, 21], simulazione di fenomeni biochimici come la divisione cellulare [11]. Questo elevato grado di genericità è fornito da Alchemist grazie all'utilizzo di un meta-modello flessibile, su cui gli sviluppatori devono mappare le proprie astrazioni, realizzando una cosiddetta "incarnazione".
È possibile simulare qualunque modello traducibile nel meta-modello di Alchemist, il tutto mantenendo le elevate prestazioni derivanti dal tipo di algoritmi utilizzati.

1.1 Modello del dominio

La parte fondamentale di Alchemist, che gli consente di simulare diversi sistemi pur mantenendo prestazioni elevate è la struttura del suo meta-modello. Il meta-modello, rappresentato in Figura 1.1, è composto dalle seguenti entità:

- **Ambiente:** in Alchemist modella l'astrazione dello spazio. Il suo compito è quello di contenere i nodi e quindi sapere la posizione dei vari nodi e la distanza tra questi.

- **Nodo:** situato all'interno di un ambiente ha il compito di contenere le molecole e le reazioni.

- **Linking-rule:** funzione che associa ogni nodo ad un suo vicinato.

- **Vicinato:** è un'entità che mi permette di conoscere i nodi vicini a un dato nodo.

- **Molecola:** nome simbolico di un dato, corrisponde al nome di una variabile in un linguaggio di programmazione imperativo.

- **Concentrazione:** valore associato ad una particolare molecola.

- **Reazione:** rappresenta un qualsiasi evento che comporta un cambiamento di stato dell'ambiente, attraverso le azioni associate alla reazione ed eseguite in risposta all'evento che le ha generate. La frequenza con cui gli eventi vengono generati dipendono dalla combinazione di una lista di condizioni e da distribuzioni temporali.

- **Condizione:** è una funzione che dato in input lo stato attuale dell'ambiente consente o meno l'esecuzione della reazione a cui è associata e ne può influenzare la frequenza con cui si presenta.

- **Azione:** invocata in seguito al verificarsi di una reazione ha il compito di apportare modifiche allo stato dell'ambiente.

- **Posizione:** coordinate che rappresentano il punto in cui è situato il nodo all'interno dell'ambiente.

- **Route:** insieme di posizioni che rappresentano il percorso che deve seguire un nodo.

Le implementazioni del meta-modello sono definite come incarnazioni ed hanno l'obbligo di includere la concretizzazione del concetto di concentrazione. Al momento, le astrazioni presenti sono:

- **SAPERE:** [23, 24, 25] incarnazione progettata per modellare un ecosistema di calcolo distribuito che trae ispirazione da ecosistemi naturali.
- **Protelis:** incarnazione per simulare reti di dispositivi che eseguono programmi Protelis [18]. Protelis è un linguaggio funzionale basato sui campi computazionali ed offre una piattaforma per la programmazione aggregata.
- **Biochemistry:** [12] incarnazione che permette di simulare un ambiente multi-cellulare, dove le molecole e le concentrazioni hanno lo stesso significato loro normalmente attribuito nelle scienze biologiche.

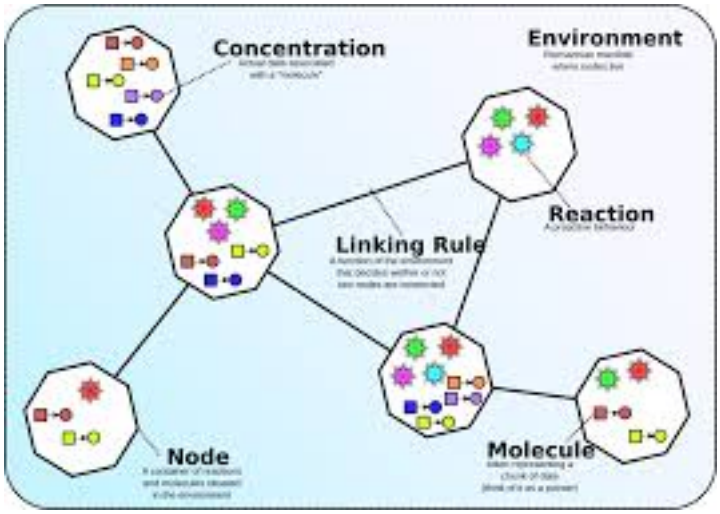

Figura 1.1: Rappresentazione del meta-modello di Alchemist

1.2 Architettura

Alchemist presenta un'architettura di tipo Boundary Control Entity (BCE), il boundary collega al controller l'input dell'utente e ne gestisce l'output, il controller contiene il motore del simulatore, attraverso il quale modifica lo stato del modello. Il motore di Alchemist [6, 9] utilizza un algoritmo modificato di Gibson-Bruck [8] al fine di ottenere un'efficiente simulazione stocastica. Il

boundary utilizza un *OutputMonitor* al fine di ottenere lo stato del modello per esporlo all'utente tramite GUI o per esportare su file i parametri d'interesse. Le iterazioni tra queste macro-entità sono rappresentate dallo schema in Figura 1.2.

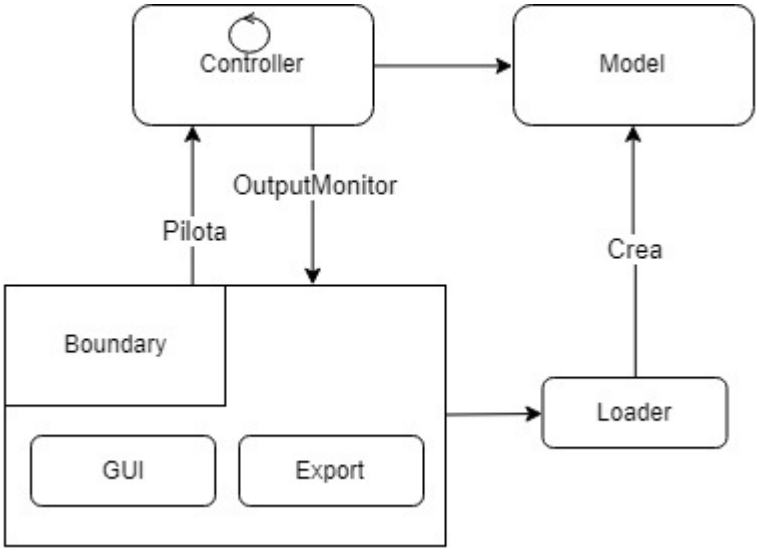

Figura 1.2: Rappresentazione dell'architettura di Alchemist

1.3 Utilizzo

Al fine di eseguire una simulazione utilizzando Alchemist, è necessario fornirgli un file di configurazione che definisca come devono essere implementate tutte le varie entità del modello.
Alchemist richiede un file in formato YAML [4] in cui occorre specificare quali classi (presenti nel classpath) utilizzare come entità concrete del modello e i relativi parametri. In particolare è possibile definire:

- **Incarnation:** è l'unica chiave obbligatoria da specificare ed accetta una stringa che rappresenta il nome dell'incarnazione che si vuole utilizzare. Le incarnazioni attualmente disponibili sono: *proteils*, *sapere*, *scafi* e *biochemistry*.

- **Variables:** permette di specificare una lista di variabili da utilizzare per batch di simulazioni.
- **Seed:** rappresenta la definizione del seme per la generazione dei numeri pseudo-random usati durante la creazione della simulazione.
- **Environment:** questa chiave è opzionale e indica quale ambiente vogliamo utilizzare nella nostra simulazione, ad esempio nel caso vogliamo utilizzare una mappa dovremmo indicare l'ambiente *OSMEnvironment* e passare come parametro il file contenente la mappa.
- **Position:** consente di specificare quale classe utilizzare per rappresentare le coordinate che indicano le posizioni dei nodi nell'ambiente.
- **Network model:** permette di definire come devono essere collegati tra loro i nodi a seconda di quale classe viene richiesta.
- **Displacements:** questa chiave è necessaria per specificare come vengono decise le posizioni iniziali da attribuire ai nodi e quali reazioni, quindi azioni, associare a quest'ultimi.
- **Export:** consente di definire quali informazioni si vogliono esportare in un file esterno

Il risultato ottenuto da una simulazione Alchemist è l'evoluzione dell'ambiente allo scorrere del tempo a causa delle azioni configurate. Ad esempio, possiamo simulare come le persone si siano spostate durante un particolare periodo di tempo, assegnando ad ogni nodo un'azione che lo faccia spostare seguendo un preciso tracciato GPS, come la simulazione dello spostamento degli spettatori durante la maratona di Vienna [1], oppure, presupponendo di sapere la qualità dell'aria in vari punti della città, possiamo configurare la nostra simulazione per mostrarci quale dovrebbe essere il percorso migliore data una partenza e una destinazione, in termini di tempo impiegato e attraversamento di zone con la miglior qualità dell'aria.

1.4 Funzionalità di geographic information system

Il GIS incorporato in Alchemist consente di caricare mappe geografiche provenienti da OpenStreetMap al fine di costruire un grafo in grado di rappresentare

tutte le strade della zona caricata, in particolare i nodi rappresenteranno i vari punti di interconnessione tra più strade, le curve e comunque tutti i punti in cui non siamo in presenza di strade diritte; mentre gli archi, che rappresentano le strade, ci consentono semplicemente di sapere quali nodi sono collegati tra loro. Queste informazioni ci consento di poter interrogare il navigatore chiedendo di restituirci un insieme di archi e nodi da dover attraversare per poter raggiungere una particolare destinazione dal punto di partenza indicato. Inoltre, è anche possibile ottenere itinerari specifici per diversi mezzi di trasporto, ad esempio, supponendo di essere dei pedoni, può consentirci di attraversare parchi e zone pedonali, ma evita i percorsi che includono autostrade, e viceversa, nel caso in cui si intenda utilizzare una vettura per il nostro spostamento.
La presenza di questo sistema da solo, non è però sempre sufficiente in quanto, ad esempio, il navigatore che è presente, ci consente di creare dei percorsi partendo da un certo punto, consentendoci quindi di simulare quali potrebbero essere i possibili spostamenti intrapresi, ma non è possibile definire, in modo preciso, le posizioni iniziali da assegnare ai nodi, ne tantomeno poter ripercorrere particolari percorsi svolti nel mondo reale. Al fine di risolvere questi problemi ed avere quindi simulazioni più accurate e tendenti alla realtà, si è integrato un sistema per il supporto ai tracciati GPS. I tracciati GPS sono quindi utilizzati per definire le posizioni iniziali dei nodi, e percorrere particolari itinerari d'interesse. Utilizzati in congiunzione con il navigatore, è possibile interpolare le tracce GPS, per utilizzare i soli percorsi consentiti al veicolo selezionato.

Capitolo 2

I tracciati GPS

Con il termine traccia GPS si intende una sequenza di punti geo localizzati ordinati in base all'istante di tempo in cui sono stati rilevati. Un punto GPS è rappresentato mediante le due coordinate, latitudine e longitudine, e un terzo valore che rappresenta l'elevazione dal livello del mare. In realtà un tracciato non è formato direttamente da un insieme di punti, ma bensì da un insieme di segmenti che a loro volta contengono i punti. Un segmento rappresenta in particolare un insieme di punti rilevati uno successivamente all'altro senza nessuna interruzione, il verificarsi di una interruzione nei rilevamenti, generalmente causata dalla mancanza del segnale GPS, comporta la terminazione di quel segmento, mentre la ripresa della registrazione dei rilevamenti comporta l'inizio di un nuovo segmento.

2.1 Il formato GPX

Il formato GPX [7] (GPS eXchange Format) consiste in uno schema XML[1] progettato per consentire il trasferimento di informazioni GPS tra diversi software, è un formato aperto e, grazie anche alla sua semplicità, si è imposto come standard de facto. Il formato GPX consente di rappresentare:

- **Waypoint:** punti di interesse anche scollegati tra di loro
- **Route:** insieme sequenziale di waypoints che rappresentano informazioni geografiche utilizzate per fornire percorsi utili

[1] http://www.topografix.com/GPX/1/1/gpx.xsd

- **Track:** lista ordinata di punti che rappresenta un percorso. A differenza della route, ogni punto mantiene l'istante in cui è stato registrato. Lo schema XML di un Track è riportato in Figura 2.1.

2.2 Il formato KML

KML [5] (Keyhole Markup Language) è un linguaggio di markup basato su XML con lo scopo di rappresentare dati geo spaziali in due o tre dimensioni. Nato per rappresentare le informazioni su Google Earth, la versione 2.2 è stata riconosciuta come standard internazionale dall' **Open Geospatial Consortium** nel 2008, al fine di garantire il suo stato di standard aperto.
In KML ogni punto è rappresentato dalla latitudine e longitudine, più, opzionalmente, da inclinazione, inquadratura, ed altitudine. In presenza di tutte le suddette informazioni, si parla di *camera view*. KML specifica anche un set di elementi rappresentabili dai software geo spaziali come: segnalibri geografici, immagini, poligoni, modelli 3D, descrizioni, etichette testuali, eccetera. Lo schema XML di un tracciato KML è riportato in Figura 2.2.

2.3 Il formato TCX

TCX[2] (Training Center XML) è un formato per lo scambio dati introdotto nel 2007 da Garmin come parte dei sui prodotti indirizzati all'attività sportiva. Lo schema XML è simile a quello GPX, con la differenza che le tracce GPS sono rappresentate come attività, piuttosto che una serie di punti GPS. TCX inoltre, in quanto pensato per rappresentare sessioni di allenamento, fornisce elementi per la rappresentazione di parametri fondamentali come la frequenza cardiaca, il ritmo della pedalata e le calorie consumate.

2.4 Fonti di tracce GPS esistenti

Esistono varie fonti che consentono di ottenere tracce GPS, tra cui OpenStreetMap (OSM), un progetto collaborativo nato con lo scopo di creare mappe e cartografie composte da dati disponibili con licenza libera. OSM rende

[2] http://www8.garmin.com/xmlschemas/TrainingCenterDatabasev2.xsd

```
<xsd:complexType name="trkType">
    <xsd:annotation>...</xsd:annotation>
    <xsd:sequence>
        <xsd:element name="name" type="xsd:string"
        minOccurs="0">...</xsd:element>
        <xsd:element name="cmt" type="xsd:string"
        minOccurs="0">...</xsd:element>
        <xsd:element name="desc" type="xsd:string"
        minOccurs="0">...</xsd:element>
        <xsd:element name="src" type="xsd:string"
        minOccurs="0">...</xsd:element>
        <xsd:element name="link" type="linkType"
        minOccurs="0" maxOccurs="unbounded">
        ...</xsd:element>
        <xsd:element name="number"
        type="xsd:nonNegativeInteger" minOccurs="0">
        ...</xsd:element>
        <xsd:element name="type" type="xsd:string"
        minOccurs="0">...</xsd:element>
        <xsd:element name="extensions"
        type="extensionsType" minOccurs="0">
        ...</xsd:element>
        <xsd:element name="trkseg" type="trksegType"
        minOccurs="0" maxOccurs="unbounded">
        ...</xsd:element>
    </xsd:sequence>
</xsd:complexType>
```

Figura 2.1: schema XML che rappresenta un tracciato. Come si può vedere, oltre che da una lista di segmenti, è composto anche da informazioni aggiuntive che consentono di sapere quale tracciato è rappresentato e chi lo ha costruito oltre ad un apposito elemento extension che consente di aggiungere elementi personalizzati definiti in altri schemi.

```
<kml:Track
 id="ID [0..1]"
 targetId="NCName [0..1]"
 anyAttribute="anySimpleType [0..1]">
    <kml:ObjectSimpleExtensionGroup>...
    </kml:ObjectSimpleExtensionGroup> [0..*]
    <kml:AbstractGeometrySimpleExtensionGroup>...
    </kml:AbstractGeometrySimpleExtensionGroup> [0..*]
   <kml:AbstractGeometryObjectExtensionGroup>...
    </kml:AbstractGeometryObjectExtensionGroup> [0..*]
   <kml:extrude>...</kml:extrude> [0..1]
    <kml:tessellate>...</kml:tessellate> [0..1]
    <kml:altitudeMode>...
    </kml:altitudeModeGroup> [0..1]
   <kml:seaFloorAltitudeMode>...
   </kml:seaFloorAltitudeMode> [0..1]
   <kml:AltitudeModeSimpleExtensionGroup>...
    </kml:AltitudeModeSimpleExtensionGroup> [0..1]
   <kml:AltitudeModeObjectExtensionGroup>...
    </kml:AltitudeModeObjectExtensionGroup> [0..1]
   <kml:when>...</kml:when> [0..*]
    <kml:coord>...</kml:coord> [0..*]
    <kml:angles>...</kml:angles> [0..*]
    <kml:Model>...</kml:Model> [0..1]
    <kml:ExtendedData>...</kml:ExtendedData> [0..1]
    <kml:TrackSimpleExtensionGroup>...
    </kml:TrackSimpleExtensionGroup> [0..*]
    <kml:TrackObjectExtensionGroup>...
    </kml:TrackObjectExtensionGroup> [0..*]
</kml:Track>
```

Figura 2.2: schema XML che rappresenta un tracciato in formato KML.

disponibile a tutti gli utenti registrati la possibilità di caricare le proprie tracce GPS anonimizzate, e le rende disponibili al pubblico che le può scaricare e riutilizzare nei modi consentiti dalla licenza. OSM generalmente tratta i propri dati secondo il formato GPX, ma utilizzando particolari estrattori è possibile ottenerli anche in formato KML.
Le tracce GPS possono essere ottenute anche attraverso i software geo spaziali di Google, ovvero Google Earth e Google Maps, in questo caso il formato principale in cui si possono ottenere i dati è ovviamente il formato KML, anche se sono rese disponibili funzionalità per il supporto GPX.

2.5 Creazione di nuove tracce

La creazione di tracce GPS è un'azione tecnicamente semplice, l'unico strumento necessario è un modulo GPS che consenta di ottenere la propria posizione ed un'applicazione che registri una serie di posizioni con il relativo istante di rilevamento. La creazione di una traccia GPS può quindi avvenire in svariati modi, attraverso un'applicazione per smartphone che ottiene le posizioni dal modulo integrato, attraverso l'uso di strumenti wearable (modalità utilizzata dai dispositivi garmin per la creazione di tracce TCX per le sessioni di allenamento), oppure banalmente attraverso il navigatore GPS presente nelle automobili.

Capitolo 3

Integrazione dei formati di navigazione GPS standard in Alchemist

In questo capitolo mostreremo prima i requisiti che sono stati raccolti durante una prima fase di analisi, e quindi quali sono state le scelte progettuali attuate per soddisfare tali richieste. Infine sarà analizzato il risultato ottenuto dall'implementazione della soluzione progettata.

3.1 Analisi dei requisiti

L'obbiettivo principale è quello di creare una struttura in grado di supportare i formati di navigazione GPS standard ed in primis il formato GPX.
Prima di iniziare a pensare a possibili soluzioni in grado di risolvere il problema, si è effettuato uno studio dettagliato del sistema esistente per capire come sono attualmente supportati i formati di navigazione e quindi individuare il miglior modo di intervenire. A seguito di questa analisi si è dedotto che è sconveniente riutilizzare o anche solo estendere il vecchio sistema di caricamento dei dati in quanto si basava sulla lettura di una risorsa binaria facente uso della serializzazione Java. Inoltre la gestione delle tracce GPS è demandata interamente all'ambiente, che si occupa del loro caricamento e della loro distribuzione alle azioni ogni qual volta ne fanno richiesta. Appare più naturale invece associare le tracce GPS alle azioni che le utilizzano, mentre l'ambiente si dovrebbe occupare

solamente di mantenere lo stato della simulazione. Un problema di modellazione riguarda il fatto che una posizione generica all'interno dell'ambiente e una posizione GPS vengano rappresentate come entità scollegate ed indipendenti, così come anche i percorsi composti da posizioni e posizioni GPS.
Riassumendo quindi i requisiti individuati sono:

1 Creare una struttura che consenta di supportare diversi formati di navigazione GPS standard e facilmente estendibile per supportare nuovi formati in futuro.

2 Il sistema deve essere utilizzabile senza modifiche sostanziali all'architettura del sistema di caricamento delle simulazioni

3 Revisionare il modello di rappresentazione di *Position*, *Route*, *GPSPoint* e *GPSTrace*

4 Sgravare l'ambiente dalla gestione delle tracce GPS

5 Assegnare le tracce GPS alle azioni che le utilizzano

6 Mantenere elevate prestazioni per non creare un collo di bottiglia nel simulatore (requisito non funzionale)

3.2 Modello del dominio

Prima di addentrarsi nello studio di soluzioni progettuali per l'integrazione dei formati standard di navigazione GPS, è importante rispondere prima al requisito 3, in quanto va a modificare parte del modello del simulatore.

3.2.1 Modifica del modello

Al fine di trovare la miglior soluzione si è studiato come fossero attualmente correlate tra loro le entità *Position*, *GPSPoint*, *Route* e *GPSTrace*. Nel risultato, visibile in Figura 3.1, manca una relazione tra *Position*, che rappresenta un generica posizione all'interno di un ambiente e *GPSPoint*, che rappresenta invece una posizione in un sistema a sole due coordinate (latitudine e longitudine) unite ad un istante temporale. La soluzione migliore consiste nell'introdurre una nuova entità per rappresentare una posizione geo spaziale (latitudine e longitudine) senza supporto al tempo, come specializzazione dell'entità *Position*, e

quindi trattare l'entità *GPSPoint* come specializzazione della nuova entità. Allo stato attuale l'interfaccia *Route* espone un metodo per ottenere il tempo totale necessario per percorrerla, ma essendo un parametro facoltativo, si è preferito toglierlo da questa entità ed inserirlo nella nuova entità *TimedRoute*, specializzazione di *Route*. Consideriamo quindi una *GPSTrace* come una specializzazione di *TimedRoute*, che espone funzionalità per maneggiare il percorso attraverso il tempo. La raffigurazione della soluzione proposta risulta quindi essere quella visibile in Figura 3.2.

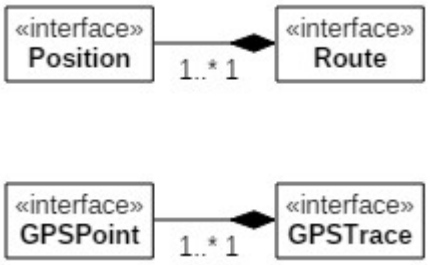

Figura 3.1: Modello prima del re-design; una *Route* è una composizione di *Position*, mentre una *GPSTrace* è una composizione di *GPSPoint*

3.3 Design dettagliato

In questa sezione è mostrato il design dettagliato della struttura per il caricamento delle tracce, assieme alle modifiche apportate al fine di delegare la gestione delle tracce GPS alle azioni che ne fanno uso.

3.3.1 Supporto formati GPS standard

Ora, dopo aver modificato il modello ed aver modificato le relazioni tra le varie entità in gioco, è finalmente possibile concentrarsi nello studio di una struttura che supporti l'importazione di tracce GPS in vari formati, tenendo bene a mente che sarà necessaria una strategia per associare ad ogni azione una traccia da seguire.
Una prima soluzione, raffigurata in figura 3.3, prevede l'utilizzo di un'entità

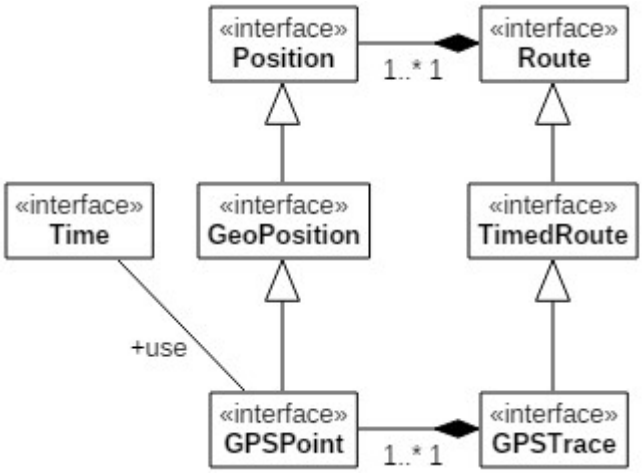

Figura 3.2: Modello dopo il re-design; le entità specifiche per la navigazione GPS (*GPSPoint*, *GPSTrace*) convergono con quelle basi del modello (*Position*, *Route*)

TraceLoader che avrà il compito dell'intera gestione delle tracce GPS, dal caricamento da file alla possibilità di fornirle alle azioni che ne fanno richiesta. In particolare, in questa versione, ci si è focalizzati sul creare una struttura altamente generica ed estendibile al fine da supportare la più vasta gamma di utilizzi possibili, per questo motivo si è pensato di introdurre l'entità *GPSDataLoader*, applicando il pattern Strategy, al fine di delegare a quest'ultima tutta la logica di caricamento dei dati. Questa entità quindi si dovrà occupare di caricare il mapping che associa alle azioni di un nodo una specifica traccia e dovrà quindi essere in grado di caricare le varie tracce richieste. Sempre per gli stessi obiettivi, ovvero ottenere una struttura altamente generica ed estendibile, si è optato per applicare il pattern Strategy, al fine di delegare questi due compiti ad altrettante entità, abbiamo quindi *NodeToTraceMapper* per la gestione del caricamento del mapping e *GPSFileLoader* per il caricamento delle tracce da file.
Una particolare considerazione va fatta anche in merito alla gestione del tempo, in quanto nel simulatore il tempo è rappresentato da un valore numerico con valore iniziale pari a 0, mentre nelle tracce GPS ogni punto ha associato un tempo che rappresenta l'istante in cui è stato rilevato. È quindi necessario determinare una strategia per mettere in relazione questi due concetti, oltre che per determinare quale relazione c'è tra i tempi delle varie tracce. Per risolvere questo problema

si è quindi scelto di adottare la stessa strategia utilizzata per il caricamento delle tracce GPS, ovvero applicare il pattern Strategy, per delegare questo compito ad una entità esterna, *GPSTimeAlignment*.

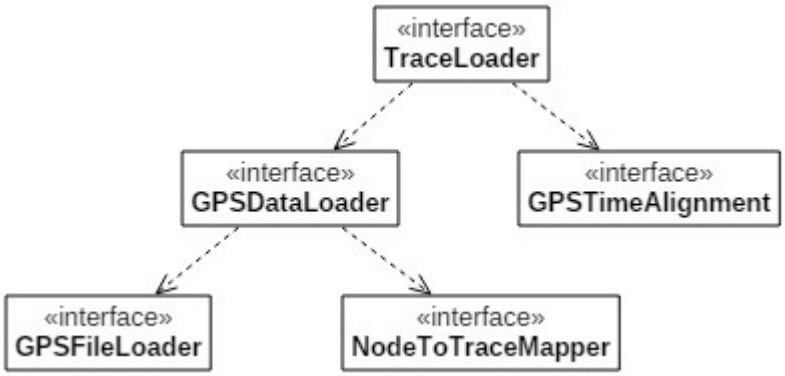

Figura 3.3: Struttura della prima versione del loader, si enfatizza estendibilità e genericità delegando ogni compito ad una entità diversa.

Questa soluzione sembrava essere quella ideale per il nostro scopo in quanto prevedeva un meccanismo per:

- Associare una traccia alle azioni di un nodo;
- caricare le traccie da diversi formati;
- allineare i tempi delle tracce caricate.

Il passo successivo è stato quindi quello di produrre un prototipo per verificare se la soluzione prodotta funzionasse e rispondesse realmente a tutti i nostri requisiti. Una volta terminato e testato il prototipo si è notato che effettivamente svolgeva interamente il suo compito come previsto, ma con uno svantaggio: la ricerca di una soluzione altamente generica ha portato a dover gestire una casistica di situazioni estremamente varia, che andava a complicare la creazione della configurazione della simulazione, richiedendo l'utilizzo di un secondo file di configurazione per specificare quale dovesse essere il mapping tra i nodi e la relativa traccia associata, con l'onere della creazione del file che sarebbe spettato all'utente ed avrebbe richiesto tempo. Si è valutato che tipicamente si associa una traccia GPS ad ogni nodo, e quindi il concetto di mapping tra le azioni di

un nodo e una traccia, proveniente dalla versione attuale, può essere eliminato riducendo il grado di complessità della soluzione precedente.
Il modello ottenuto dalla semplificazione, che rispetta tutti i requisiti imposti, riduce la figura del *TraceLoader* ad un semplice *Iterable* di *GPSTrace* ed elimina le due entità non più necessarie a causa dell'eliminazione del mapping nodo-traccia. Il nuovo modello, riportato in Figura 3.4, mantiene quindi solo i due strategy per definire come caricare i file delle tracce, e come allineare i tempi.

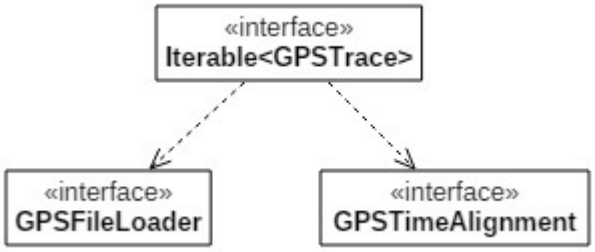

Figura 3.4: Struttura della seconda versione del loader, più semplice della precedente, ma che comunque mantiene la possibilità di estendere i compiti più critici delegandoli a entità esterne.

Il design che ne consegue, figura 3.5, prevede la presenza di un caricatore per i file in formato GPX, che utilizza la libreria jpx[1], la quale consente uno stile di programmazione funzionale per accedere al contenuto del file. Per l'allineamento dei tempi si è inserita una classe astratta, che incapsula la logica per far iniziare tutte le tracce a partire da un determinato tempo, con la definizione di quest'ultimo delegata alle classi che la estenderanno.

3.3.2 Gestione delle tracce assegnate alle azioni

Ora che abbiamo sia riallineato il modello tramite un'opera di re ingegnerizzazione, sia implementato una strategia per il caricamento delle tracce GPS stabile, estendibile e facilmente mantenibile è arrivato il momento di fare l'ultimo step per portare a compimento il lavoro, ovvero togliere la gestione delle tracce all'ambiente e delegare questo compito direttamente alle azioni. Per prima cosa elenchiamo quali sono gli attuali utilizzi che l'ambiente fa delle tracce:

[1] https://github.com/jenetics/jpx

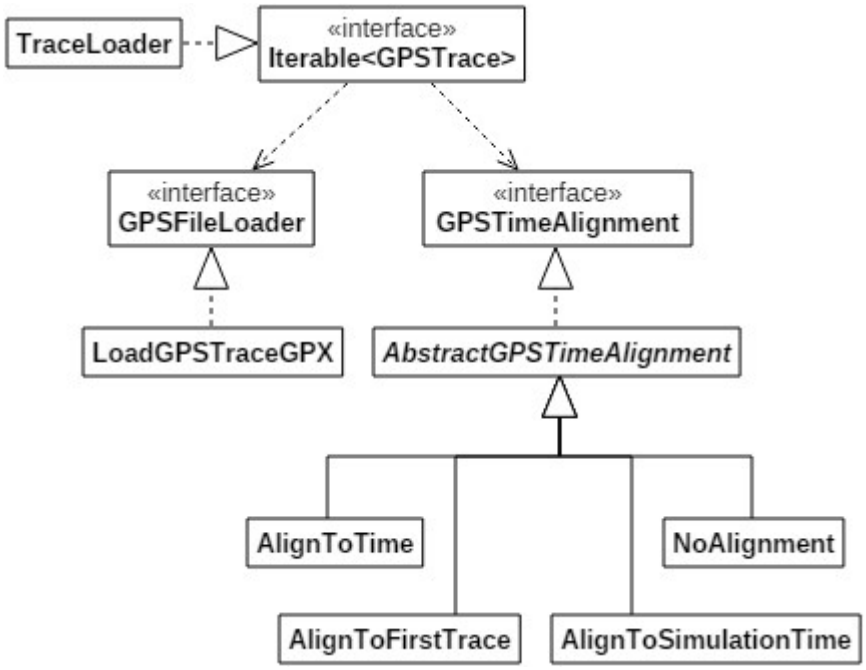

Figura 3.5: Design di dettaglio del loader, rappresenta tutte le implementazioni che è possibile utilizzare al momento

1. Caricamento da file;
2. utilizzo tracce per valutare se e dove posizionare inizialmente il nodo;
3. assegnamento delle tracce alle azioni che ne fanno richiesta.

Se è vero che il primo compito è già stato rimpiazzato dal loader appena creato, è anche vero che occorre delegare a qualcuno gli ultimi due compiti. Per implementare diversamente il secondo compito ci viene in aiuto un'entità già prevista all'interno del simulatore ed utilizzata dal caricatore del simulatore per inizializzare la simulazione. Questa entità è il *Displacement*, un *Iterable* di *Position*, ed ha il compito di fornire tutte le posizioni in cui è necessario posizionare dei nodi. Il nuovo *Displacement* non fa altro che richiedere al caricatore delle tracce di fornirgli l'insieme di quelle tracce che si vogliono utilizzare nella simulazione, e quindi restituirà il numero di posizioni richieste prelevando dalle tracce la loro posizione di partenza. Questo ci garantisce che tutti i nodi che parteciperan-

no alla simulazione saranno posizionati esattamente nei punti iniziali delle varie tracce.
Meno triviale è stato trovare una soluzione per assegnare la traccia all'azione. Per prima cosa è stata introdotta una classe intermedia, *MoveOnMapWithGPS*, al fine di rappresentare una generica azione per il movimento sulle mappe che utilizza una traccia GPS, utile per incapsulare la memorizzazione della traccia utilizzata dall'azione, Figura 3.6.

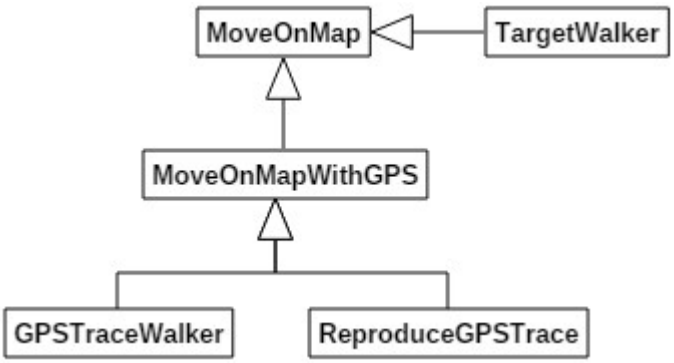

Figura 3.6: Introduzione della classe intermedia *MoveOnMapWithGPS* come punto di partenza per tutte le azioni che utilizzano una traccia GPS.

Uno dei problemi riscontrati in questa fase riguarda l'impossibilità di poter passare la traccia GPS all'azione in fase di istanziazione direttamente tramite il costruttore, in quanto questo è invocato dal caricatore del simulatore che può utilizzare solo un insieme ristretto dei tipi presenti nel progetto. I tipi riconosciuti dal caricatore sono:

- tutti i primitivi, ed i loro wrapper, escluso void;
- String;
- liste ed array;
- Incarnation;
- Environment;
- Node;

- Reaction;
- TimeDistribution;
- RandomGenerator.

Come si può vedere non è supportato il tipo *GPSTrace* in quanto non era ancora presente nel momento in cui è stato implementato il caricatore del simulatore, ed una modifica a questa parte è altamente sconsigliata e da valutare solo nel caso non ci sia nessun'altra soluzione applicabile dato l'elevato grado di complessità di questo componente. L'unica strada percorribile è quindi quella di passare ai costruttori delle azioni i parametri necessari al caricamento delle tracce e quindi delegare il compito di reperire la traccia direttamente all'azione. In particolare, questo compito ricadrà sulla nuova classe intermedia che ha già l'onere di memorizzare la traccia usata dall'azione. Risolto questo problema, cerchiamo un modo per assegnare la traccia ai suoi reali utilizzatori. I reali utilizzatori delle tracce non sono direttamente le azioni, ma gli strategy a cui queste delegano i tre compiti variabili a cui sono soggette, ovvero determinare la velocità del nodo, la prossima posizione da raggiungere, e definire la politica su come calcolare il percorso da seguire (usare solo strade o ignorandole). Volendo che la traccia assegnata agli strategy sia un parametro immodificabile, lo stato dell'arte prevede che questo parametro possa essere impostato solo tramite il costruttore dell'oggetto. Questa pratica nel nostro caso specifico non è attuabile in quanto questi strategy devono essere istanziati prima di avere a disposizione la traccia assegnata all'azione a cui fanno riferimento. Si sono valutate diverse soluzioni in termini di rispetto dei requisiti, semplicità, sicurezza ed estendibilità. Le diverse soluzioni valutate sono state:

1 Assegnare la traccia agli strategy dopo la loro creazione;

2 utilizzare una cache per memorizzare la traccia assegnata all'azione;

3 lasciare la gestione delle tracce nell'ambiente;

4 inserire nelle azioni un costruttore privato a cui passare la traccia come parametro.

La soluzione 3, è stata scartata in quanto continuerebbe ad essere l'ambiente a gestire le tracce. Inoltre, sarebbe necessario reintrodurre un mapping per riconoscere la traccia che appartiene ad una determinata azione. L'unico utilizzo

possibile della cache, soluzione 2, sarebbe quello di utilizzare come chiave l'azione stessa, ma essendo in via di costruzione non è possibile utilizzarla, rendendo così questa soluzione difficilmente praticabile. La soluzione 4, che prevede di aggiungere alle azioni che utilizzano una traccia GPS un costruttore privato a cui passare la traccia assegnata in modo da poterla girare agli strategy direttamente nel costruttore, renderebbe possibile la sua immodificabilità all'interno dello strategy, ma rimarrebbe sempre una soluzione fragile in termini di estendibilità, in quanto non possiamo forzare la presenza di tale costruttore all'interno delle azioni che verranno introdotte in futuro. L'altra soluzione percorribile è quella di aggiungere agli strategy un'interfaccia per consentire l'inizializzazione della traccia, attraverso un metodo, in un secondo momento, quando l'azione relativa sarà già stata creata e quindi avrà a disposizione la propria traccia GPS. Questa soluzione rende modificabile lo stato degli strategy e li costringe ad implementare la nuova interfaccia, cosa incapsulabile in una superclasse, e la si è preferita, rispetto alla soluzione precedente, in quanto l'invocazione di tale metodo è incapsulabile all'interno della classe *MoveOnMapWithGPS*, che già si occupa della gestione delle tracce per le azioni. Inoltre si considera meno probabile l'aggiunta di nuovi strategy che utilizzino le tracce, essendone il sistema già ricco, rispetto all'introduzione di nuove azioni, con una conseguente minor probabilità di problemi di estendibilità, , limitati anche dalla solo necessità di estendere la superclasse che gestisce la traccia GPS.

3.3.3 Estensibilità e supporto di ulteriori formati

Il nuovo modello studiato e raffigurato in Figura 3.4 garantisce di poter estendere la funzionalità di caricamento delle tracce a nostro piacimento in quanto si delegano ad entità esterne il caricamento di tracce, a seconda del formato, e l'allineamento dei tempi.
In particolare, nel caso in cui si decidesse di voler supportare un nuovo formato di tracce GPS, l'unica cosa necessaria da fare sarà quella di implementare un nuovo tipo di *GPSFileLoader*. Questo dovrà dichiarare quali sono le estensioni dei file di questo formato, ed essere quindi in grado di comprenderlo, riuscendo ad estrarre tutte le tracce incluse nel file, nessuna altra operazione sarà necessaria in quanto, per come ideato, sarà compito del *TraceLoader* individuare quale *GPSFileLoader* utilizzare per ogni file da caricare a seconda dell'estensione del file.
In modo analogo, nel caso si volessero allineare i tempi delle tracce caricate con

nuova strategia, sarà necessario solamente creare un nuovo *GPSTimeAlignment* ed indicare l'intento di volerlo utilizzare nel file di configurazione della simulazione.

3.4 Aspetti implementativi e di sviluppo

3.4.1 Modifica al modello

Come per la fase progettuale, la prima fase ad essere stata implementata è stata la modifica del modello al fine di convergere le entità di *Position* con *GPSPoint* e quella di *Route* con *GPSTrace*. Questa operazione ha semplicemente previsto l'implementazione della nuova interfaccia *GeoPosition* al fine di rappresentare un punto sulla superficie terrestre, dichiarando quali iterazioni sono consentite con oggetti di questo tipo, quindi si sono unificate tutte le entità in gioco come da diagramma in Figura 3.2, con relativo riallineamento dei metodi esposti dalle varie interfacce.

3.4.2 TraceLoader

Il *TraceLoader* incapsula tutta la logica per il caricamento delle tracce ed in particolare delega i compiti principali ad entità esterne, si veda la Figura 3.4. Al fine di consentire la possibilità di poter utilizzare più formati di tracce all'interno della stessa simulazione è necessario individuare quale *GPSFileLoader* utilizzare. In questa versione il *TraceLoader* si crea una mappa da estensione del file all'istanza di *GPSFileLoader* da utilizzare per quel file, l'insieme dei *GPSFileLoader* viene reperito tramite reflection, quindi mappato alle estensioni che dichiarano di supportare. In fase di caricamento si estrae l'estensione del file da caricare e si cerca nella mappa l'istanza di *GPSFileLoader* da utilizzare, verificando se è in grado di comprendere il file.
Comportamento diverso si è tenuto per l'utilizzo del *GPSTimeAlignment* in quanto deve essere utilizzato lo stesso per tutte le tracce e quindi può essere scelto dall'utente attraverso la configurazione. Il *TraceLoader* consente di ricevere come parametro direttamente un'istanza del *GPSTimeAlignment* oppure il nome della classe da utilizzare e l'insieme dei parametri per istanziarlo, si utilizza quindi la reflection per individuare la classe richiesta e i parametri per filtrare il costruttore da utilizzare, nel caso esista il costruttore richiesto si procede con l'istanziazione altrimenti si comunica l'errore.

3.4.3 Gestione delle tracce assegnate alle azioni

La parte fondamentale per l'assegnazione della traccia direttamente alle azioni che le utilizzano è certamente la classe *MoveOnMapWithGPS*. A questa classe è infatti assegnato il compito di distribuire le tracce tra le sue varie sottoclassi. Per implementare ciò si è inserito in questa classe una memoria rapida per il caricamento (*cache*) statica a due livelli con lo scopo di tenere traccia del punto a cui si è arrivati a distribuire le tracce. In particolare il primo livello di *cache* utilizza l'ambiente come chiave allo scopo di mantenere separate simulazioni diverse che possono utilizzare le stesse tracce, mentre il secondo livello di *cache* utilizza come chiave i parametri per il caricamento delle tracce (percorso con le risorse, allineatore dei tempi e il booleano per considerare le tracce cicliche o meno) al fine di consentire di avere azioni con diverse fonti di tracce. Accedendo a questa *cache* si ottiene un iteratore di *GPSTrace*, da cui è possibile ottenere la traccia da assegnare all'azione. Per motivi di performance si fa utilizzo di un'ulteriore *cache* che partendo dai parametri per caricare le tracce punta al corrispondente *TraceLoader* da cui è possibile ottenere l'iteratore che distribuisce le tracce. Quest'ultima *cache* mantiene le informazioni in memoria solo per un periodo di tempo limitato ed ha lo scopo di evitare il caricamento multiplo di tracce secondo gli stessi parametri, ma usate da ambienti differenti.
L'assegnazione delle tracce agli strategy avviene nel costruttore di questa classe, dove, dopo aver ottenuto la traccia relativa all'azione, si controlla se gli strategy necessitino della traccia GPS, e, nel caso, viene passata attraverso l'apposita interfaccia. La non modificabilità della traccia utilizzata dagli strategy deve essere garantita da quest'ultimi, acconsentendo ad un'unica assegnazione di questo parametro.

3.4.4 Profilazione e analisi delle prestazioni

Terminata l'implementazione si è eseguito un test per controllare il tempo impiegato per il caricamento. Una prima valutazione riportava un tempo di caricamento prossimo ai 27 secondi per circa 1500 tracce, Figura 3.7. Un tempo così alto era dovuto a un collo di bottiglia, Figura 3.8, che si verificava quando si identificava la risorsa come file o cartella, in particolare il problema si presentava in quanto la nostra risorsa era un file di cospicue dimensioni ed a causa di problemi di formattazione dovuti all'utilizzo di diverse sequenze di terminatori di riga tra i vari sistemi operativi, si considerava il file come avente un'unica riga. Al fine

di risolvere questo problema si è fissata una quantità di dati da importare per identificare il tipo di risorsa. Dopo questa modifica si sono rivalutati i tempi, che sono scesi sensibilmente, fino a 9 secondi nella medesima situazione, Figura 3.9, non introducendo nessun altro collo di bottiglia, Figura 3.10.

	Total Time (CPU)
it.unibo.alchemist.loader.displacements.GPSTraceDisplacement.**<init>** ()	26.898 ms
it.unibo.alchemist.boundary.gpsload.impl.TraceLoader.**<init>** ()	21.999 ms
it.unibo.alchemist.boundary.gpsload.impl.TraceLoader.**<init>** ()	21.999 ms
it.unibo.alchemist.boundary.gpsload.impl.TraceLoader.**loadTraces** ()	21.900 ms
it.unibo.alchemist.boundary.gpsload.api.AbstractGPSAlignment.**alignTime** ()	99,5 ms
Self time	0,000 ms
Self time	0,000 ms
it.unibo.alchemist.boundary.gpsload.impl.TraceLoader.**<clinit>**	4.899 ms
Self time	0,000 ms
it.unibo.alchemist.model.implementations.environments.OSMEnvironment.**<init>** ()	702 ms
Self time	0,000 ms

Figura 3.7: Snapshot dei tempi di caricamento prima della modifica. Il caricatore impiega in totale 27 secondi di cui 5 per creare la mappa tra estensioni e caricatori per i vari formati, e 22 secondi per il caricamento delle tracce e allineamento dei tempi.

Hot Spots - Method	Self Time [%]	Self Time (CPU)
it.unibo.alchemist.boundary.gpsload.impl.TraceLoader.**lambda$10** ()		36.193 ms
org.reflections.vfs.SystemFile.**openInputStream** ()		9.208 ms
javassist.bytecode.ClassFile.**read** ()		5.148 ms
io.jenetics.jpx.XMLReaderImpl.**read** ()		2.607 ms
org.reflections.ReflectionUtils.**forName** ()		2.550 ms
org.mapsforge.map.awt.graphics.AwtBitmap.**<init>** ()		1.859 ms
io.jenetics.jpx.ZonedDateTimeFormat.**formatParse** ()		1.418 ms
org.mapsforge.map.layer.download.TileDownloader.**getInputStream** ()		1.335 ms

Figura 3.8: Presenza del collo di bottiglia nella lambda che identifica la risorsa come file o cartella.

	Total Time (CPU)
it.unibo.alchemist.loader.displacements.GPSTraceDisplacement.**<init>** ()	9.098 ms
it.unibo.alchemist.boundary.gpsload.impl.TraceLoader.**<clinit>**	5.199 ms
it.unibo.alchemist.boundary.gpsload.impl.TraceLoader.**<init>** ()	3.899 ms
it.unibo.alchemist.boundary.gpsload.impl.TraceLoader.**<init>** ()	3.899 ms
it.unibo.alchemist.boundary.gpsload.impl.TraceLoader.**loadTraces** ()	3.800 ms
it.unibo.alchemist.boundary.gpsload.api.AbstractGPSTimeAlignment.**alignTime** ()	99,1 ms
Self time	0,000 ms
Self time	0,000 ms
Self time	0,000 ms
it.unibo.alchemist.model.implementations.environments.OSMEnvironment.**<init>** ()	601 ms
Self time	0,000 ms

Figura 3.9: Snapshot dei tempi di caricamento dopo la modifica. Il caricatore impiega in totale 9 secondi di cui 5 per creare la mappa tra estensioni e caricatori per i vari formati, e 4 secondi per il caricamento delle tracce e allineamento dei tempi.

Hot Spots - Method	Self Time [%]	Self Time (CPU)
org.reflections.vfs.SystemFile.**openInputStream** ()		8.328 ms
javassist.bytecode.ClassFile.**read** ()		4.077 ms
org.mapsforge.map.layer.download.TileDownloader.**getInputStream** ()		3.127 ms
org.reflections.ReflectionUtils.**forName** ()		2.600 ms
io.jenetics.jpx.XMLReaderImpl.**read** ()		2.303 ms
org.mapsforge.map.layer.download.TileDownloader.**downloadImage** ()		1.573 ms
io.jenetics.jpx.ZonedDateTimeFormat.**formatParse** ()		1.558 ms
javassist.bytecode.Utf8Info.**<init>** ()		1.347 ms

Figura 3.10: Dopo la modifica non sono più presenti colli di bottiglia.

3.4.5 Copertura dei test

Una volta terminata l'implementazione delle varie modifiche ed integrazioni si è passati a verificare che il lavoro svolto funzionasse come previsto. Per verificarlo si è controllato il comportamento delle nuove funzionalità attraverso un insieme di test automatici, in modo da evitare future regressioni. Al fine di valutare quanto del codice prodotto effettivamente è stato testato si è utilizzato Jacoco, che valuta appunto quante istruzioni e ramificazioni di ogni classe siano state valutate attraverso i test. Il modulo di GIS prima di questo rinnovamento presentava percentuali di copertura piuttosto basse, 33% di istruzioni testate e 19% di branch testate, mentre al termine del rinnovamento presenta percentuali più alte, 54% di istruzioni 42% di branch, Figura 3.11. La nuova parte implementata presenta un copertura superiore all'80%, con picchi fino al 100% per quanto riguarda il caricatore del formato GPX.

Element	Missed Instructions	Cov.	Missed Branches	Cov.
it.unibo.alchemist.boundary.gpsload.api		93%		95%
it.unibo.alchemist.boundary.gpsload.impl		84%		75%
it.unibo.alchemist.model.implementations		66%		75%
it.unibo.alchemist.model.implementations.actions		12%		3%
it.unibo.alchemist.model.implementations.environments		54%		34%
it.unibo.alchemist.model.implementations.linkingrules		0%		0%
it.unibo.alchemist.model.implementations.movestrategies		0%		0%
it.unibo.alchemist.model.implementations.movestrategies.routing		50%		n/a
it.unibo.alchemist.model.implementations.movestrategies.speed		0%		0%
it.unibo.alchemist.model.implementations.movestrategies.target		26%		0%
it.unibo.alchemist.model.implementations.positions		52%		36%
it.unibo.alchemist.model.implementations.routes		64%		61%
it.unibo.alchemist.model.interfaces		94%		n/a
it.unibo.alchemist.utils		56%		n/a
Total	1.744 of 3.862	54%	139 of 240	42%

Figura 3.11: Statistiche di copertura dei test nel modulo di GIS.

3.5 Correzione dei problemi preesistenti

Durante la fase di implementazione ed utilizzo del simulatore si sono riscontrati alcuni problemi già presenti nel sistema e a cui si è posto rimedio. I problemi riscontrati sono stati:

- Importazione delle mappe come risorse

- Mancata comunicazione degli errori all'utente
- Impossibilità di ricaricare lo stesso progetto
- Creazione di un nuovo progetto in Windows

3.5.1 Importazione delle mappe come risorse

Durante la fase di implementazione della modifica dell'ambiente, si è notato che si voleva supportare il caricamento della mappa da un file interno o esterno al classpath, ma in entrambi i casi vi si accedeva utilizzando un oggetto di tipo *File*, che impedisce di accedere a risorse presenti nel classpath. La soluzione è stata quella di reperire il percorso del file che rappresenta la mappa, quindi creare una cartella di lavoro sul file system univoca per la mappa rappresentata. Infine si è proceduto ad implementare la copia del file all'interno della cartella con meccanismi di mutua esclusione, necessari per evitare copie multiple del file all'interno della stessa cartella, causate dalla presenza di più simulazioni concorrenti. La copia del file è stata resa obbligatoria in quanto, la libreria per l'utilizzo della mappa, richiede necessariamente l'input della mappa sotto forma di oggetti di tipo *File*.

3.5.2 Mancata comunicazione degli errori all'utente

Durante l'utilizzo del simulatore con file di configurazione errato per verificare la comprensibilità degli errori mostrati agli utenti del simulatore, si è notato che non veniva mostrato in alcun modo la presenza di errori, e la GUI restava aperta in attesa di richieste dell'utente. Investigando sul comportamento della GUI si è notato che era prevista la comunicazione dell'errore verificato attraverso un apposito alert. Il problema era dovuto alla creazione dell'alert in un thread diverso da quello che aveva il compito di mostrarlo, comportamento che si è scoperto sbagliato controllando l'implementazione della relativa libreria, al fine di risalire al problema che ne impediva la visualizzazione, e la relativa documentazione. Il problema è stato risolto spostando la creazione dell'alert all'interno del thread che aveva il compito di mostrarlo.

3.5.3 Impossibilità di ricaricare lo stesso progetto

In Alchemist, dalla versione 4.0.0, con l'introduzione della nuova GUI, è stato introdotto anche il concetto di progetto, inteso come impostazioni con cui lanciare la simulazione. Con impostazioni si intende la definizione del file di configurazione della simulazione, il file in cui salvare gli output prodotti, gli effetti da applicare alla simulazione e se eseguirla in modalità batch o meno. Per la gestione del progetto era già stata prevista la sua creazione, salvataggio e strategia per il caricamento del precedente salvataggio, ma mancava la parte in cui la GUI veniva inizializzata con i parametri precedentemente salvati, impedendo la riapertura dello stesso progetto. Il problema è stato risolto completando la parte di caricamento di un progetto inserendo la parte di inizializzazione della GUI.

3.5.4 Creazione di un nuovo progetto in Windows

In Alchemist, quando si decide di creare un nuovo progetto, dopo aver selezionato la cartella in cui salvare il progetto, occorre selezionare il template con cui inizializzarlo. Al momento della copia del template nella cartella di lavoro si generava un errore in quanto non era in grado di reperire il template richiesto. La causa di questo problema era dovuta alla manipolazione del percorso della cartella che contiene tutti i template disponibili, ed in particolare, alla *regex* che aveva il compito di estrarre i nomi dei template disponibili, infatti non si teneva in considerazione che i sistemi Windows utilizzano un file separator differente rispetto agli altri sistemi. Il problema è stato risolto modificando il file separator del percorso, nel caso di utilizzo su sistemi Windows, prima dell'utilizzo della *regex*.

3.6 Utilizzo lato utente

Un utente che desideri utilizzare una mappa ed eventualmente le tracce GPS nella propria simulazione deve definire in modo appropriato tre parametri del file di configurazione:

1. Environment;
2. Displacement;
3. Action.

Environment

Dei tre è l'unico parametro obbligatorio. Al fine di poter utilizzare le mappe, è necessario scegliere un ambiente che le supporti, al momento l'unico è *OSMEnvironment*. È l'unico parametro obbligatorio in quanto non necessariamente siamo costretti ad utilizzare anche le tracce GPS insieme alle mappe. *OSMEnvironment* mette a disposizione diversi costruttori, dove l'unico obbligatorio è il percorso del file che contiene le mappe. Gli altri costruttori consentono di definire come posizionare i nodi, ovvero se spostarli sulla strada e se questo spostamento sia obbligatorio o meno, e il grado di approssimazione, ovvero quanto vicino alla destinazione posso arrivare al fine di consentire l'utilizzo di percorsi simili a quello ottimale ma già presenti in memoria al fine di limitare il tempo per reperire il percorso richiesto. Esempi di configurazione sono presentati in Figura 3.12 e Figura 3.13

```
environment:
  type: OSMEnvironment
  parameters: ["/vcm.pbf"]
```

Figura 3.12: definizione dell'ambiente per il supporto alle mappe con unico parametro il percorso del file con la mappa all'interno del classpath. I nodi se possibile vengono posizionati su una strada vicina, altrimenti non vengono spostati.

```
environment:
  type: OSMEnvironment
  parameters: ["/vcm.pbf", true, true]
```

Figura 3.13: definizione dell'ambiente per il supporto alle mappe con parametro il percorso del file con la mappa all'interno del classpath. I due valori booleani indicano che i nodi devono essere posizionati su una strada vicina, altrimenti vengono scartati.

Displacement

Come già spiegato nella Sezione 1.3 il *Displacement* ha il compito di fornire le posizioni in cui inserire inizialmente i nodi, quindi nel caso non si sia intenzionati ad utilizzare le tracce GPS si è liberi di scegliere quello preferito tra quelli disponibili, altrimenti si è obbligati ad utilizzare il *Displacement FromGPSTrace* che posiziona i nodi nelle posizioni iniziali delle tracce da utilizzare. Questo *Displacement*, oltre a quanti nodi posizionare, richiede tutte le informazioni necessarie per caricare le tracce, ovvero il percorso in cui sono presenti, se considerarle cicliche o meno, e come allineare i tempi. Un esempio di configurazione è presentata in Figura 3.14

```
displacements:
  - in:
      type: FromGPSTrace
      parameters: [1497, "/vcmuser.gpx", false,
            "AlignToTime", 1365922800, false, false]
    programs:
      - *move
    contents: []
```

Figura 3.14: definizione del *Displacement* che posiziona 1479 nodi, utilizza il file ”/vcmuser.gpx” presente all’interno del classpath come fonte di tracce da utilizzare. Il terzo parametro determina se distribuire ciclicamente le tracce in caso siano minori dei nodi richiesti, quindi viene specificato quale allineatore di tempi utilizzare ed i parametri per la sua creazione. Ad ogni nodo che sarà creato sui suoi punti associa una reazione ”move”.

Action

La definizione delle azioni da utilizzare è una parte fondamentale della configurazione in quanto determina come evolverà il sistema. Nel caso si voglia utilizzare un’azione che provochi lo spostamento del nodo seguendo una traccia GPS, si può scegliere tra due diverse azioni a seconda che si desideri seguire una strada (*GPSTraceWalker*) oppure raggiungere le posizioni successive in modo rettilineo (*ReproduceGPSTrace*), in entrambi i casi è necessario definire come caricare le

tracce da seguire definendo il percorso in cui sono presenti le tracce, se considerarle cicliche o meno e come allineare i tempi. Un esempio di configurazione è presentata in Figura 3.15

```
programs:
  - time-distribution: 0.1
  type: Event
  actions:
    - type: ReproduceGPSTrace
      parameters: ["/vcmuser.gpx", false,
            "AlignToTime", 1365922800, false, false]
```

Figura 3.15: definizione di un programma con associata un'azione che segue una traccia GPS reperita grazie alle informazioni fornite nella lista dei parameters in modo analogo del *Displacement*.

Ora che abbiamo visto nel dettaglio come configurare i singoli parametri fondamentali, mostriamo un esempio completo di file di configurazione per l'utilizzo delle tracce GPS nelle azioni, Figura 3.16. Una demo che mostra il comportamento del simulatore con questo file di configurazione è disponibile al link https://github.com/Placu95/AlchemistDemo.

```
environment:
  type: OSMEnvironment
  parameters: ["/vcm.pbf"]

incarnation: protelis

pools:
  - pool: &move
    - time-distribution: 0.1
      type: Event
      actions:
        - type: ReproduceGPSTrace
          parameters: ["/vcmuser.gpx", false,
            "AlignToTime", 1365922800, false, false]

displacements:
  - in:
      type: FromGPSTrace
      parameters: [1497, "/vcmuser.gpx", false,
            "AlignToTime", 1365922800, false, false]
    programs:
      - *move
```

Figura 3.16: file di configurazione completo per istanziare una simulazione che utilizzi azioni che seguono tracce GPS.

Capitolo 4

Conclusioni e lavori futuri

In questa tesi si è migliorato il modulo di geographic information system di Alchemist, introducendo la possibilità di caricare tracce GPS in formati standard, migliorando l'architettura, e risolvendo vari problemi preesistenti.
I requisiti definiti in fase di analisi sono stati tutti soddisfatti, in particolare ci si è focalizzati sul design della struttura d'importazione. Inizialmente si è prodotta una struttura altamente generica in grado di supportare diverse casistiche, quindi si è semplificata andando ad eliminare il grado di complessità introdotto in fase di configurazione della simulazione, pur continuando a garantire una facile estendibilità, al fine di supportare diversi formati di tracce GPS. Infine si sono valutate le prestazioni del simulatore e come questa struttura di caricamento incideva sulle prestazioni generali, andando a scoprire che introduceva un collo di bottiglia significativo, quindi si è provveduto ad eliminarlo. Per valutare la correttezza della struttura e favorire future espansioni di questa di quest'ultima, evitando possibili regressioni, si sono prodotti test automatici con il compito di valutarne l'integrità.
In futuro si prevede di affiancare al formato GPX altri formati come KML, TCX, e tutti quelli che saranno considerati utili.

Ringraziamenti

Ringrazio il professore Mirko Viroli e l'ingegnere Danilo Pianini per l'opportunità che mi è stata concessa e per tutto quello che ho potuto apprendere durante questo periodo. Ringrazio anche la mia famiglia e i miei amici che mi hanno accompagnato in questo percorso e mi hanno supportato durante i periodi più intensi.

Bibliografia

[1] B. Anzengruber, D. Pianini, J. Nieminen, and A. Ferscha. Predicting social density in mass events to prevent crowd disasters. In A. Jatowt, E. Lim, Y. Ding, A. Miura, T. Tezuka, G. Dias, K. Tanaka, A. J. Flanagin, and B. T. Dai, editors, *Social Informatics - 5th International Conference, SocInfo 2013, Kyoto, Japan, November 25-27, 2013, Proceedings*, volume 8238 of *Lecture Notes in Computer Science*, pages 206–215. Springer, 2013.

[2] J. Beal, D. Pianini, and M. Viroli. Aggregate programming for the internet of things. *IEEE Computer*, 48(9):22–30, 2015.

[3] J. Beal, M. Viroli, D. Pianini, and F. Damiani. Self-adaptation to device distribution changes. In G. Cabri, G. Picard, and N. Suri, editors, *10th IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2016, Augsburg, Germany, September 12-16, 2016*, pages 60–69. IEEE Computer Society, 2016.

[4] O. Ben-Kiki, C. Evans, and I. döt Net. Yaml ain't markup language (yaml™), version 1.2, 3rd edition, 2009-10-01. http://yaml.org/spec/1.2/spec.pdf.

[5] D. Burggraf. Ogc kml 2.3, 2015-08-04. http://docs.opengeospatial.org/is/12-007r2/12-007r2.html.

[6] F. Damiani, M. Viroli, D. Pianini, and J. Beal. Code mobility meets self-organisation: A higher-order calculus of computational fields. In S. Graf and M. Viswanathan, editors, *Formal Techniques for Distributed Objects, Components, and Systems - 35th IFIP WG 6.1 International Conference, FORTE 2015, Held as Part of the 10th International Federated Conference on Distributed Computing Techniques, DisCoTec 2015, Grenoble, France,*

June 2-4, 2015, Proceedings, volume 9039 of *Lecture Notes in Computer Science*, pages 113–128. Springer, 2015.

[7] D. Foster. Gpx: the gps exchange format. http://www.topografix.com/gpx.asp.

[8] M. A. Gibson and J. Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *The Journal of Physical Chemistry A*, 104(9):1876–1889, mar 2000.

[9] S. Montagna, A. Omicini, and D. Pianini. Extending the gillespie's stochastic simulation algorithm for integrating discrete-event and multi-agent based simulation. In B. Gaudou and J. S. Sichman, editors, *Multi-Agent-Based Simulation XVI - International Workshop, MABS 2015, Istanbul, Turkey, May 5, 2015, Revised Selected Papers*, volume 9568 of *Lecture Notes in Computer Science*, pages 3–18. Springer, 2015.

[10] S. Montagna, D. Pianini, and M. Viroli. Gradient-based self-organisation patterns of anticipative adaptation. In *Sixth IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2012, Lyon, France, September 10-14, 2012*, pages 169–174. IEEE Computer Society, 2012.

[11] S. Montagna, D. Pianini, and M. Viroli. A model for drosophila melanogaster development from a single cell to stripe pattern formation. In Ossowski and Lecca [15], pages 1406–1412.

[12] S. Montagna, D. Pianini, and M. Viroli. A model for drosophila melanogaster development from a single cell to stripe pattern formation. In Ossowski and Lecca [15], pages 1406–1412.

[13] S. Montagna, M. Viroli, D. Pianini, and J. L. Fernandez-Marquez. Towards a comprehensive approach to spontaneous self-composition in pervasive ecosystems. In F. D. Paoli and G. Vizzari, editors, *Proceedings of the 13th Workshop on Objects and Agents, Milano, Italy, September 17-19, 2012*, volume 892 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012.

[14] S. Montagna, M. Viroli, M. Risoldi, D. Pianini, and G. D. M. Serugendo. Self-organising pervasive ecosystems: A crowd evacuation example. In E. Troubitsyna, editor, *Software Engineering for Resilient Systems - Third International Workshop, SERENE 2011, Geneva, Switzerland, September*

29-30, 2011. Proceedings, volume 6968 of *Lecture Notes in Computer Science*, pages 115–129. Springer, 2011.

[15] S. Ossowski and P. Lecca, editors. *Proceedings of the ACM Symposium on Applied Computing, SAC 2012, Riva, Trento, Italy, March 26-30, 2012*. ACM, 2012.

[16] D. Pianini, J. Beal, and M. Viroli. Improving gossip dynamics through overlapping replicates. In A. Lluch-Lafuente and J. Proença, editors, *Coordination Models and Languages - 18th IFIP WG 6.1 International Conference, COORDINATION 2016, Held as Part of the 11th International Federated Conference on Distributed Computing Techniques, DisCoTec 2016, Heraklion, Crete, Greece, June 6-9, 2016, Proceedings*, volume 9686 of *Lecture Notes in Computer Science*, pages 192–207. Springer, 2016.

[17] D. Pianini, S. Montagna, and M. Viroli. Chemical-oriented simulation of computational systems with ALCHEMIST. *J. Simulation*, 7(3):202–215, 2013.

[18] D. Pianini, M. Viroli, and J. Beal. Protelis: practical aggregate programming. In R. L. Wainwright, J. M. Corchado, A. Bechini, and J. Hong, editors, *Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, April 13-17, 2015*, pages 1846–1853. ACM, 2015.

[19] G. Stevenson, J. Ye, S. Dobson, D. Pianini, S. Montagna, and M. Viroli. Combining self-organisation, context-awareness and semantic reasoning: the case of resource discovery in opportunistic networks. In S. Y. Shin and J. C. Maldonado, editors, *Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC '13, Coimbra, Portugal, March 18-22, 2013*, pages 1369–1376. ACM, 2013.

[20] M. Viroli, A. Bucchiarone, D. Pianini, and J. Beal. Combining self-organisation and autonomic computing in cass with aggregate-mape. In S. Elnikety, P. R. Lewis, and C. Müller-Schloer, editors, *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W), Augsburg, Germany, September 12-16, 2016*, pages 186–191. IEEE, 2016.

[21] M. Viroli, R. Casadei, and D. Pianini. Simulating large-scale aggregate mass with alchemist and scala. In M. Ganzha, L. A. Maciaszek, and M. Paprzycki, editors, *Proceedings of the 2016 Federated Conference on Computer Science and Information Systems, FedCSIS 2016, Gdańsk, Poland, September 11-14, 2016.*, pages 1495–1504, 2016.

[22] M. Viroli, D. Pianini, and J. Beal. Linda in space-time: An adaptive coordination model for mobile ad-hoc environments. In M. Sirjani, editor, *Coordination Models and Languages - 14th International Conference, COORDINATION 2012, Stockholm, Sweden, June 14-15, 2012. Proceedings*, volume 7274 of *Lecture Notes in Computer Science*, pages 212–229. Springer, 2012.

[23] M. Viroli, D. Pianini, S. Montagna, G. Stevenson, and F. Zambonelli. A coordination model of pervasive service ecosystems. *Sci. Comput. Program.*, 110:3–22, 2015.

[24] F. Zambonelli, G. Castelli, L. Ferrari, M. Mamei, A. Rosi, G. D. M. Serugendo, M. Risoldi, A. Tchao, S. Dobson, G. Stevenson, J. Ye, E. Nardini, A. Omicini, S. Montagna, M. Viroli, A. Ferscha, S. Maschek, and B. Wally. Self-aware pervasive service ecosystems. In E. Giacobino and R. Pfeifer, editors, *Proceedings of the 2nd European Future Technologies Conference and Exhibition, FET 2011, Budapest, Hungary, May 4-6, 2011*, volume 7 of *Procedia Computer Science*, pages 197–199. Elsevier, 2011.

[25] F. Zambonelli, A. Omicini, B. Anzengruber, G. Castelli, F. L. D. Angelis, G. D. M. Serugendo, S. A. Dobson, J. L. Fernandez-Marquez, A. Ferscha, M. Mamei, S. Mariani, A. Molesini, S. Montagna, J. Nieminen, D. Pianini, M. Risoldi, A. Rosi, G. Stevenson, M. Viroli, and J. Ye. Developing pervasive multi-agent systems with nature-inspired coordination. *Pervasive and Mobile Computing*, 17:236–252, 2015.