Progettazione e sviluppo di un simulatore per l'Ambient Intelligence

Tesi di Laurea Magistrale in Ingegneria Informatica

Giovanni Pecoraro

Relatore: Prof. Giuseppe Lo Re

Correlatore: Ing. Pierluca Ferraro

UNIVERSITA' DEGLI STUDI DI PALERMO

LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

PROGETTAZIONE E SVILUPPO DI UN SIMULATORE PER L'AMBIENT INTELLIGENCE

Tesi di Laurea di
Dott. Giovanni Pecoraro

Relatore:
Ch.mo Prof. Giuseppe Lo Re

Correlatore:
Ing. Pierluca Ferraro

Sommario

Le soluzioni di Ambient Intelligence sono parte integrante della vita quotidiana delle persone, a partire dagli ambienti domestici, dove è in continua crescita l'introduzione di nuovi dispositivi sempre più avanzati e di gestori centralizzati come ad esempio gli assistenti virtuali. Lo sviluppo di queste nuove tecnologie richiede anche la valutazione del loro comportamento all'interno di un ambiente dove sono presenti altri dispositivi con i quali si ritroveranno ad interagire. Gli strumenti offerti dalla letteratura sono orientati allo studio di uno specifico aspetto del settore o di determinati casi di studio, pertanto con questo lavoro di tesi si è realizzato un simulatore che permettesse di affrontare tutti i principali scenari applicativi dell'Ambient Intelligence e rimanesse aperto a nuove integrazioni ed aggiornamenti. Il simulatore fornisce un ambiente virtuale general purpose personalizzabile e senza vincoli che modella con le dovute semplificazioni il mondo reale. La struttura software che lo caratterizza è scalabile e di tipo modulare, permettendo l'integrazione di nuovi elementi da introdurre nella simulazione, oppure moduli che estendano le funzionalità del simulatore o che permettano l'interfacciamento con altri sistemi esterni come ad esempio i gestori di Ambient Intelligence. Un'altra funzionalità di rilievo offerta dal simulatore è la possibilità di introdurre nell'ambiente un utente che svolge delle attività di vita quotidiana, per rendere più realistica e dinamica la simulazione e poter valutare il comportamento degli elementi oggetto del caso di studio anche a fronte dell'imprevedibilità delle azioni svolte dall'utente virtuale, il cui comportamento è generato dinamicamente ed in autonomia mediante tecniche di intelligenza artificiale. Gli scenari di applicazione del simulatore di Ambient Intelligence permettono di valutare anche aspetti collaterali al settore di studio, quali ad esempio la simulazione di anomalie o guasti, la presenza di rumore nelle letture dei sensori, l'impatto di azioni malevole ai fini della sicurezza e molte altre applicazioni. La valutazione dei risultati ottenuti è stata effettuata mediante l'applicazione di metriche analitiche oggettive, le quali hanno confermato il raggiungimento degli scopi prefissati ed è emersa una buona similarità con i dati di esperimenti reali, rendendo quindi il simulatore di Ambient Intelligence adeguato alle applicazioni di utilizzo per cui è stato progettato.

Indice

Introduzione

L'Ambient Intelligence (AmI) è un paradigma multidisciplinare che riguarda tutti quegli scenari dove la tecnologia è a supporto degli ambienti in cui l'uomo vive. Si tratta di un paradigma *user-centric* [1] supportato da metodologie di intelligenza artificiale con il fine di operare in maniera pervasiva, non invadente e trasparente per l'utente. Il concetto è stato sviluppato per la prima volta dall'ente *IST Programme Advisory Group* (ISTAG), consulente della Commissione Europea, nel documento *Scenarios for Ambient Intelligence* [2].

Negli ultimi anni si è osservato un trend fortemente crescente per quanto riguarda l'impiego pratico di tecniche di Ambient Intelligence in contesti di vita quotidiana che coinvolgono sia i luoghi pubblici che le abitazioni private. Questo è stato possibile grazie al supporto delle moderne tecnologie le quali hanno raggiunto un livello di miniaturizzazione, autonomia ed intelligenza capace di fornire funzionalità avanzate. Tutto ciò viene supportato da una infrastruttura di comunicazione che permette lo scambio dati veloce e sicuro tra tutte le parti coinvolte.

Tale diffusione è ulteriormente favorita dalla sempre crescente integrazione delle tecnologie nella vita quotidiana. Basti pensare che ad esempio i comuni elettrodomestici oggi non svolgono più solamente un unico compito in maniera isolata ma sono spesso forniti di ulteriori sensori e funzionalità che offrono vantaggi e semplicità di utilizzo all'utente finale. Questi elementi forniscono un primo livello di integrazione tra tecnologia ed ambiente di vita quotidiana e possono rivelarsi una base di partenza per applicazioni di Ambient Intelligence.

Gli sviluppi più recenti hanno permesso una ulteriore evoluzione di questi aspetti sfruttando l'interconnessione di tutti gli elementi presenti nell'ambiente con l'obiettivo dell'automazione e di fornire una maggiore autonomia nelle operazioni più comuni. L'avvento *dell'Internet of Things* (IoT) ha contribuito in maniera considerevole alla diffusione di applicazioni di Ambient Intelligence [3] accrescendo l'integrazione degli elementi nell'ambiente. Un'altra tecnologia di recente introduzione che ha ulteriormente favorito sia la distribuzione che l'integrazione delle soluzioni di Ambient Intelligence nella vita di tutti i giorni è rappresentata sicuramente dagli assistenti virtuali i quali offrono nuovi metodi di interfacciamento con gli utenti e possono anche includere funzioni dedicate alla gestione degli ambienti. Gli assistenti virtuali risultano infatti capaci di sfruttare la connessione con vari dispositivi per fornire all'utente informazioni centralizzate e gestione autonoma. Questo incoraggia l'aggiunta di nuovi elementi per ampliare la rete esistente in modo da fornire maggiori funzionalità, comfort e semplificazioni alla vita quotidiana, il tutto completamente adattabile ai più svariati contesti e scenari che coinvolgono le attività degli esseri umani.

Un tale insieme di elementi in cooperazione tra loro risulta fortemente eterogeneo e complesso da gestire. Nelle applicazioni pratiche vi sono inoltre ulteriori variabili che entrano in gioco, riguardanti le leggi che regolano un ambiente reale e la specifica implementazione dei singoli elementi che dovrebbero potersi integrare a prescindere dal produttore o dalla specifica tecnologia impiegata. Per questi e diversi altri motivi, nella fase di sviluppo di un nuovo

componente destinato ad applicazioni di Ambient Intelligence risulta necessario eseguire un notevole numero di test volti sia al miglioramento del progetto sia alla verifica delle possibili interazioni con il mondo circostante. Tali esperimenti richiedono una notevole quantità di risorse e tempo, ma dato che la componente software è sicuramente la parte più complessa da sviluppare, si può sfruttare la virtualizzazione e risulta estremamente vantaggioso affrontare alcune fasi di sviluppo mediante delle prove virtuali capaci di accorciare in maniera considerevole i tempi. In questo modo le prove con un prototipo nell'ambiente reale possono essere affrontate quando il progetto risulta già consolidato così che le prove fisiche siano volte principalmente a confrontare i modelli matematici usati nella simulazione in un ambiente regolato dalle vere leggi della fisica.

Quello appena descritto è il motivo alla base della realizzazione del simulatore software di Ambient Intelligence oggetto del presente lavoro di tesi. È stato progettato per essere il più generico possibile e facilmente adattabile al più vasto numero di scenari, ulteriormente ampliato dalla capacità di poter introdurre qualsiasi tipologia di elementi nelle simulazioni. L'obiettivo principale del simulatore di Ambient Intelligence sviluppato è quello di fornire un ambiente virtuale nel quale si possano modellare senza vincoli le proprietà fisiche che lo regolano e permettere di riprodurre gli scenari in cui gli elementi coinvolti si possano ritrovare così da valutarne il comportamento e l'evoluzione delle interconnessioni. Il poter creare condizioni arbitrarie apre la strada ad una serie di test che non sono realizzabili nel mondo reale o che sono difficilmente riproducibili. Un altro aspetto di notevole importanza è quello di non dover attendere realmente il tempo di evoluzione della simulazione in quanto è possibile elaborare ore di simulazione in pochi secondi di calcolo.

Il simulatore software inoltre si presta particolarmente bene ad essere impiegato come strumento di supporto allo sviluppo degli algoritmi di Ambient Intelligence e di intelligenza artificiale. Quest'ultima è una disciplina fortemente legata all'Ambient Intelligence e che ne determina senza dubbio l'efficacia, andando a ricoprire un ruolo di fondamentale importanza nel contesto degli ambienti pervasivi. Il coinvolgimento di tecniche di intelligenza artificiale inizia già dalla raccolta dei dati ambientali e dalla loro aggregazione, essendo spesso dati eterogenei. Nelle fasi successive questi sono impiegati per analizzare il contesto e prendere eventuali decisioni sulle azioni da compiere in base agli obiettivi prefissati. L'impiego di tecniche di intelligenza artificiale si riscontra in molte delle principali applicazioni dell'Ambient Intelligence, una particolare evidenza è costituita dagli assistenti virtuali i quali sfruttando queste tecniche forniscono la più naturale delle modalità di interazione con l'utente, ovvero quella verbale basata sulla lingua naturale. Questa nuova modalità di interfacciamento oltre a fornire un'elevata semplicità di utilizzo per gli esseri umani abbatte anche delle barriere che potrebbero costituire una difficoltà per persone con disabilità, fornendo accesso alle funzionalità senza che siano necessarie competenze o conoscenze specifiche. Tutta la relativa parte di riconoscimento e contestualizzazione dei comandi è elaborata mediante tecniche

proprie dell'intelligenza artificiale che si sono evolute raggiungendo un livello di efficacia capace anche di sostenere una conversazione.
In altri scenari invece l'applicazione di tecniche di intelligenza artificiale è orientata non all'interazione con gli utenti ma alla gestione in maniera parzialmente o del tutto autonoma delle condizioni ambientali, adattandosi anche ai cambiamenti che possono verificarsi. Un classico scenario di questa tipologia è la gestione automatica delle condizioni ambientali come la temperatura, l'umidità e la circolazione dell'aria che si possono riscontrare nei luoghi pubblici o all'interno dei mezzi di trasporto come treni ed aerei, per citare alcuni esempi; le stesse tecniche possono essere impiegate anche in contesti più sensibili come le sale operatorie degli ospedali o gli impianti industriali, ovvero luoghi in cui vi è la necessità di mantenere perennemente determinate condizioni ambientali.
L'intelligenza artificiale ricopre un ruolo anche all'interno del simulatore, consentendo di simulare in maniera sufficientemente realistica la presenza di persone. Un utente virtuale ha la facoltà di interagire direttamente con gli elementi presenti per poterne regolare manualmente i parametri così da ottenere le condizioni desiderate. L'obiettivo è quello di rendere più realistica la simulazione per gli elementi oggetto di studio valutando anche l'impatto delle azioni che un utente potrebbe effettuare.

Struttura della tesi

Nella prima parte di questo lavoro di tesi si introduce il contesto scientifico di riferimento per quanto riguarda la simulazione e le motivazioni che hanno portato alla progettazione e allo sviluppo del simulatore di Ambient Intelligence. Nel capitolo 1 sono esposti gli obiettivi progettuali e le scelte che hanno portato alla realizzazione delle principali proprietà che caratterizzano il simulatore, analizzando i vantaggi e gli svantaggi che ne derivano. È stato effettuato anche uno studio della letteratura presente sui principali simulatori operanti nello stesso settore per valutare le soluzioni offerte.
La parte centrale del documento presenta i dettagli di ogni componente del simulatore di Ambient Intelligence. Nel capitolo 2 si espone l'architettura logica e l'algoritmo che caratterizza la simulazione e gli elementi correlati. Si affronta nel dettaglio la metodologia usata dal simulatore per modellare il mondo reale e le scelte progettuali effettuate, illustrando le differenti tipologie di comportamento o proprietà che caratterizzano e classificano i componenti. Il capitolo 3 prosegue descrivendo la struttura software del simulatore di Ambient Intelligence dove sono affrontati nel dettaglio tutti i componenti che possono far parte della simulazione con le proprietà che li contraddistinguono nel modellare il corrispettivo elemento del mondo reale. Ogni elemento della simulazione ha infatti uno specifico ruolo a prescindere dalla sua implementazione o dalle proprietà che presenta in un determinato scenario. Il capitolo si conclude con la descrizione dei componenti di sistema, ovvero quei moduli software che non

partecipano ai calcoli di simulazione ma che ne supportano l'esecuzione e vanno a costituire il simulatore stesso, compresa anche la trattazione nel dettaglio dell'algoritmo di simulazione, dei moduli software a supporto e della variante che permette il calcolo parallelo.
Una particolare attenzione è stata data allo studio dei due componenti più complessi presenti nel simulatore di Ambient Intelligence e proprio per questo motivo sono stati dedicati dei capitoli per approfondire il relativo settore scientifico, la progettazione ed i risultati ottenuti volti a verificare il raggiungimento degli obiettivi di progetto.
Il primo di questi componenti è il linguaggio di configurazione della simulazione descritto nel capitolo 4. Creato specificatamente per questo lavoro di tesi, il linguaggio permette al simulatore di poter configurare ed eseguire le simulazioni in maniera automatica descrivendone solamente la configurazione. Il modulo software provvederà ad interpretare le istruzioni secondo la grammatica che regola il linguaggio ed eseguirà le opportune azioni.
Nel capitolo 5 viene descritto il secondo dei componenti appena citati il quale riguarda il modulo di intelligenza artificiale che ha il compito di generare dinamicamente il comportamento dell'utente virtuale incluso opzionalmente nella simulazione. Il ruolo dell'utente virtuale è quello di compiere le azioni di vita quotidiana in maniera simile a quanto farebbe un essere umano all'interno di un ambiente reale. Per lo sviluppo di questo modulo si è fatto ricorso ad una rete neurale addestrata con dati reali per costruire un modello realistico all'interno della simulazione e permettere di impiegare il simulatore di Ambient Intelligence anche negli scenari che coinvolgono gli utenti. La validazione è stata eseguita mediante l'impiego di una metrica analitica che permettesse di valutare in modo oggettivo la similarità tra i dati prodotti ed i pattern che regolano la vita di una persona reale.
La parte finale del lavoro è stata completamente dedicata alla fase sperimentale in cui sono state eseguite diverse simulazioni per poter valutare in maniera incrementale tutti gli elementi coinvolti nella simulazione, le interazioni che questi autonomamente svolgono tra loro e gli effetti che si ripercuotono sulle proprietà fisiche caratterizzanti l'ambiente. La simulazione conclusiva è stata configurata in maniera completa così da coinvolgere tutte le tipologie di elementi presenti. È stato previsto un utente che interagisce in due ambienti virtuali che caratterizzano le due zone principali di una abitazione, ovvero la zona giorno e la zona notte. Questo ha permesso di eseguire una simulazione che affrontasse uno scenario di notevole interazione tra tutti gli elementi coinvolti.

Capitolo 1
Contesto scientifico

Nel presente capitolo vengono esposti i principi alla base della simulazione che hanno ispirato lo sviluppo di questo lavoro di tesi per la realizzazione del simulatore di Ambient Intelligence. È stato anche condotto uno studio dell'attuale letteratura al fine di valutare le soluzioni presenti ed effettuare un confronto con i vantaggi offerti dal simulatore.

1.1 Confronto tra simulazione e sperimentazione reale

Imitazione di un sistema.

Viene così definita la simulazione in quello che è uno dei libri più completi sull'argomento, "*Simulation. The Practice of Model Development and Use*" [4], scritto da Stewart Robinson (Professor of Management Science at the School of Business and Economics, Loughborough University, UK).

Un modello di simulazione predice le performance di un sistema sottoposto ad uno specifico input con l'obiettivo di una maggiore comprensione e di un miglioramento di quel sistema.

Si ricorre alla simulazione in tantissimi scenari poiché fornisce notevoli vantaggi ed in alcuni casi permette di studiare degli aspetti non riproducibili o difficilmente riproducibili nel mondo reale. Altri metodi possono essere usati per lo studio del comportamento di un sistema ed uno di questi è la sperimentazione diretta con la versione reale del sistema in oggetto. In [4] vengono identificati diversi vantaggi nell'utilizzo della simulazione rispetto agli esperimenti con un sistema reale. Questi sono: il *Costo*, realizzare la sperimentazione con il sistema reale prevede uno sforzo economico e vari interventi volti a provare diversi approcci, idee e cambiamenti. Inoltre questo può alterare il normale funzionamento del sistema stesso. Il *Tempo*, perché le osservazioni del sistema reale possono impiegare molto tempo per produrre i risultati ed ulteriore ne potrebbe servire per ottenere le performance di interesse. *Controllo delle condizioni*, quando si provano approcci diversi risulta molto utile poter variare a piacimento le condizioni alle quali è sottoposto il sistema durante lo svolgimento degli esperimenti, cosa non facile da effettuare su un sistema reale o impossibile in determinati contesti. Nel modello simulato si possono invece ripetere molte volte gli esperimenti con le stesse identiche condizioni. *Esistenza del sistema*, un valido motivo per scegliere la simulazione è lo studio di un sistema che non esiste nel mondo reale oppure che non è direttamente osservabile e per il quale risulta necessario quindi sviluppare un modello.

I vantaggi appena descritti sono forniti dal simulatore di Ambient Intelligence sviluppato. I risultati della simulazione sono generalmente prodotti in pochi secondi di calcolo permettendo di effettuare simulazioni di durata arbitraria. Aspetto particolare del software è quello di permettere nella simulazione l'occorrenza di determinati eventi scatenati da una specifica condizione o da un determinato momento temporale al fine di poter valutare specifici scenari. Ad esempio è possibile sfruttare questa caratteristica per simulare come reagisce il sistema nel

caso in cui uno o più elementi dell'ambiente virtuale smettano di funzionare o funzionino in modo anomalo generando letture sbagliate.
Tra le funzionalità fornite, il simulatore permette anche di introdurre nell'ambiente virtuale un utente che possa interagire con gli elementi della simulazione al fine di valutare gli scenari che contemplano la presenza di persone. Una ulteriore libertà di personalizzazione permette di determinare il tipo di comportamento che caratterizza tale utente e le modalità di interazione con gli elementi così da poter simulare svariati casi d'uso. Ulteriori altri benefici e funzionalità offerte dal simulatore di Ambient Intelligence saranno evidenti dopo aver affrontato nei capitoli successivi l'architettura software che lo caratterizza.
Il processo di simulazione presenta anche qualche svantaggio rispetto all'osservazione di un sistema nel mondo reale. Proprio per la sua natura di imitazione, rimane una previsione; non risulta quindi equivalente alla sperimentazione reale in quanto l'accuratezza di tale previsione dipende in maniera diretta dalla qualità e da quanto siano sofisticati gli algoritmi che ne effettuano i calcoli. Un altro fattore da tenere in considerazione è la configurazione della simulazione da parte dell'operatore il quale fornisce i dati di input e le condizioni iniziali.
Questo processo, oltre alla discretizzazione delle grandezze e delle misurazioni coinvolte introduce delle semplificazioni rispetto all'esperimento nel mondo reale. Tale aspetto non è da considerarsi negativo in quanto a volte è desiderabile lavorare con una versione semplificata che permette comunque di cogliere la legge che regola il processo. In questi casi lavorare con i dettagli completi richiederebbe grandi quantità di tempo ed elaborazioni che possono risultare eccessive [4].

1.2 Tipologia della simulazione

Il simulatore di Ambient Intelligence appartiene alla categoria dei sistemi discreti, dato che i calcoli sono effettuati su grandezze discrete le quali modellano delle grandezze continue.
L'ambito di applicazione prevede che nell'ambiente simulato vi siano delle grandezze fisiche da modellare le quali nel mondo reale sono ovviamente continue. La grandezza fisica cruciale per la simulazione è sicuramente il tempo.
Sulla base di come viene modellato l'avanzamento del tempo le simulazioni vengono classificate in due categorie principali [4]. Una utilizza il metodo del *time slicing* in cui il tempo viene suddiviso in intervalli di uguale durata ognuno dei quali calcolato iterativamente in ordine. L'altra categoria utilizza il metodo *discrete event* nel quale la simulazione procede avanti nel tempo solamente quando si manifesta un evento che ne modifica lo stato.

1.2.1 Obiettivi della progettazione

Nella fase di progettazione del simulatore di Ambient Intelligence si è valutato che il paradigma di modellazione del tempo più efficace per raggiungere gli obiettivi preposti risulta il metodo *time slicing*.
Il simulatore di Ambient Intelligence è stato ideato come un simulatore generico ed obiettivo che quindi non fosse focalizzato su uno specifico aspetto della simulazione ma fornisse un mondo virtuale in cui è possibile valutare qualsiasi tipo di scenario applicabile. Ogni scenario è costituito da due elementi principali che sono le grandezze fisiche coinvolte e gli elementi in esso contenuti. Gli ulteriori elementi che possono caratterizzare una simulazione come ad esempio le attività svolte da un utente all'interno dell'ambiente virtuale non vengono trattate come componenti della simulazione, né tantomeno la simulazione viene basata su queste come fanno alcuni simulatori. Tali attività caratterizzano il comportamento dell'utente virtuale e sarà solo quest'ultimo ad interagire con l'ambiente nella stessa maniera in cui avverrebbe nel mondo reale, fornendo così un'imitazione più fedele della realtà con le dovute semplificazioni. Questo comporta che per monitorare una grandezza fisica nell'ambiente virtuale si deve inserire almeno un sensore adatto allo scopo, altrimenti non è possibile osservarne i valori, allo stesso modo di come accadrebbe nel mondo reale.

1.2.2 Paradigma time slicing

La tipologia di simulazione *time slicing* prevede che il tempo venga suddiviso in intervalli costanti Δt chiamati *time step* i quali vengono tutti elaborati dall'algoritmo di simulazione in ordine cronologico.
La scelta è ricaduta su questo paradigma perché si è ritenuto particolarmente adatto per il problema trattato. L'elaborazione di un intervallo temporale alla volta permette di effettuare un campionamento delle grandezze fisiche che caratterizzano l'ambiente virtuale emulando la situazione che si riscontrerebbe in un esperimento reale. Le operazioni effettuate da ogni elemento dell'ambiente sono eseguite ad ogni intervallo temporale ed i calcoli svolti riguardano l'intera durata di Δt stimando così l'evoluzione completa della simulazione.
L'alternativa a questo approccio sarebbe stata quella di usare il paradigma *discrete event simulation* [5] il quale prevede che l'elaborazione dello stato dell'ambiente avvenga solamente quando si presenta un evento che ne apporti delle modifiche. In questo caso non si avrebbe un'elaborazione costante delle grandezze coinvolte ed i risultati in output sarebbero stati disponibili solamente a simulazione conclusa o ad intervalli irregolari dettati dal verificarsi di determinati eventi. Inoltre, come descritto in [5], questo approccio per poter effettuare una simulazione richiede che sia fornito un elenco di eventi che vada a specificare in quale momento temporale ognuno di questi dovrà manifestarsi, introducendo anche la necessità di dover gestire delle code di priorità.
Durante la fase di progettazione si è quindi concluso che il paradigma *discrete event simulation* non rispecchia la filosofia adottata per il simulatore di Ambient Intelligence, ovvero quella di

riprodurre nelle modalità e nella maniera più simile possibile la realtà, e pertanto è stato scartato.
Il paradigma *time slicing* è caratterizzato anche da due possibili svantaggi [4]. Uno riguarda l'efficienza di calcolo, in quanto vengono elaborati tutti i *time step*, anche quelli in cui non succede nulla di rilevante mentre nel paradigma *discrete event simulation* vengono elaborati solo i momenti temporali dove si manifesta un evento. Questo aspetto non è da ritenersi negativo nel presente progetto in quanto risulta necessario calcolare tutti gli intervalli di evoluzione della simulazione al fine di fornire costantemente lo stato attuale del sistema ed aggiornare con l'intervallo di campionamento specificato le grandezze fisiche coinvolte.
La seconda problematica identificata in [4] riguarda la scelta dell'intervallo Δt, in quanto il valore selezionato va ad incidere sulla precisione con la quale viene effettuata l'elaborazione ed il monitoraggio delle grandezze. Per quanto riguarda le attività di vita quotidiana che possono caratterizzare una simulazione, anch'esse sono influenzate dalla scelta di tale intervallo perché determina la precisione temporale del collocamento nella simulazione. Se ad esempio un'attività inizia in un momento temporale a che si colloca all'interno di un intervallo di elaborazione, ovvero si verifica che $\Delta t_i \leq time(a)$, l'attività verrà eseguirà all'inizio del successivo intervallo di simulazione Δt_{i+1}. Ciò rispecchia esattamente la problematica della tecnica del campionamento discreto di un segnale continuo e come nel caso reale si interviene scegliendo l'intervallo più opportuno per il caso di studio trattato e proprio per questo motivo il simulatore permette all'utente di specificare tale valore nella fase di configurazione della simulazione.

1.3 Letteratura

In letteratura sono presenti diverse soluzioni che trattano la simulazione nel settore dell'Ambient Intelligence, tuttavia lo studio dei principali simulatori ha messo in evidenza la mancanza di uno strumento di simulazione general purpose che fosse capace di adattarsi a svariati scenari possibili dell'ambito di studio e che fornisse una facile integrazione sia con nuovi elementi che con altri sistemi capaci di elaborare determinati aspetti della simulazione.
Le soluzioni studiate si sono rivelate tutte focalizzate su un determinato aspetto del caso di studio o progettate per fornire solamente delle funzionalità specifiche per determinati scenari di impiego. Ne consegue una limitata capacità di adattamento alle diverse necessità che possono insorgere nell'osservare le svariate tipologie di scenari presenti nel mondo moderno. Tali simulatori risultano inoltre alquanto datati il che comporta delle limitazioni di applicabilità agli scenari moderni dove sono presenti nuove tipologie di elementi e dove possono essere disponibili algoritmi più sofisticati. Per questi motivi le soluzioni presenti in letteratura non sono state ritenute adatte al raggiungimento degli scopi.

Di seguito si riportano i principali strumenti presenti in letteratura che sono stati valutati. Per ognuno di questi si riportano gli obiettivi progettuali accompagnati da una breve panoramica sull'architettura in modo da poter effettuare un confronto.

3DSim

Simulatore basato sulla grafica tridimensionale per la prototipizzazione rapida e la simulazione di ambienti intelligenti [6]. L'obiettivo di 3DSim è quello di simulare un ambiente equipaggiato con dispositivi ed attuatori, come ad esempio una sala riunioni, senza richiedere di configurare il motore fisico che regola la simulazione in quanto tale processo è stato effettuato nelle fasi di sviluppo.

Risulta orientato alla valutazione dell'usabilità di nuovi dispositivi intelligenti nell'ambiente virtuale e fornisce interfacce standardizzate che consentono di integrare ed interagire con nuovi dispositivi e sensori. Tali elementi di Ambient Intelligence possono essere applicati anche ad ambienti reali.

L'architettura è costituita da un server adibito alla gestione dell'ambiente tridimensionale che detiene la logica della simulazione e dal client che vi si collega per permettere all'utente di interagire tramite un'interfaccia grafica. Ogni elemento della simulazione si può posizionare nell'ambiente variandone posizione ed orientamento e dopo tale operazione il software effettua anche dei controlli di collisione tra gli elementi. Risulta possibile interagire con la scena mediante l'interfaccia grafica e i dati ricevuti dai sensori si rispecchiano sugli oggetti tridimensionali della scena.

Persim

Simulatore scalabile guidato da eventi ed usato principalmente per la generazione di dataset di attività di vita quotidiana in un formato standard [8]. *Persim* può essere configurato in base al caso di studio ed usato su piccola o grande scala. Si basa su un set di dati chiamato *Sensory Dataset Description Language* (SDDL) contenente i dati raccolti dal monitoraggio di sensori reali.

La creazione di una simulazione avviene attraverso cinque passaggi di configurazione:

1. definizione delle aree che rappresentano l'ambiente virtuale oggetto di studio;
2. aggiunta di sensori e attuatori di varia tipologia e funzione;
3. definizione dell'elenco di attività che verranno eseguite nella simulazione (ad esempio l'azione di sedersi su una sedia, camminare da un punto ad un altro etc.);
4. specifica di una corrispondenza tra le attività aggiunte alla simulazione ed i sensori inseriti nelle aree virtuali. Tale azione va ad indicare quali sensori sono determinanti per il rilevamento di una di queste attività;
5. nel passo finale viene avviata la simulazione e sono generati i risultati.

L'architettura del simulatore fornisce all'utente una interazione mediante un pannello di controllo su pagina web che permette di gestire il progetto della simulazione. Il progetto viene poi caricato nel motore di simulazione per essere eseguito e vengono prodotti i risultati mediante il generatore di SDDL usato per la conversione dell'output.
Persim mette al centro della simulazione le attività di vita quotidiana, che oltre ad essere direttamente legate ad un sottoinsieme di sensori devono essere predeterminate dall'utente che configura il software. Al fine di raffinare la configurazione ed ottenere dei risultati soddisfacenti, la fase 4 necessita di essere ripetuta diverse volte come viene suggerito dagli autori [8], poiché le relazioni tra attività e sensori sono un elemento fondamentale per l'evoluzione della simulazione e di conseguenza per i risultati generati.

A simulator of sensor data for ambient intelligence
Simulatore adattivo ed estensibile di ambienti intelligenti basato sull'approccio della produzione e visualizzazione dei dati sensoriali permettendo anche di simulare fallimenti o comportamenti inaspettati da parte dei sensori [9].
L'architettura segue un approccio guidato dai componenti, principalmente basato sul pattern MVC, fornendo diversi sensori. È composta e caratterizzata dai seguenti elementi:

- un ambiente contenente la configurazione del simulatore ed i controller dei sensori;
- i controller dei sensori, che sono i componenti principali del simulatore ed hanno lo scopo di distribuire i segnali ai rispettivi *handler* ed effettuare delle manipolazioni tra le quali anche l'applicazione di filtri;
- gli *handler* dei segnali nell'implementazione di base riportano semplicemente il segnale ricevuto su file o su console ai fini di logging;
- il segnale è costituito dai dati del sensore e rappresenta il suo stato interno;
- la *view* di un sensore la quale rappresenta l'interfaccia grafica di interazione con l'utente;
- un *parser* di segnale necessario per la visualizzazione dei segnali registrati dall'ambiente.

Interactive Smart home Simulator
Soluzione focalizzata sul simulare e controllare il comportamento di una abitazione intelligente. Il sistema mira a fornire uno strumento utile a capire le interazioni tra l'ambiente e le persone che vi si trovano, potendo quindi valutare anche l'impatto sulle tecnologie pervasive presenti [10].
L'architettura è composta dai componenti elementari i quali modellano l'ambiente virtuale, la persona che lo abita ed i dispositivi in esso presenti. Un servizio server si occupa dell'aggregazione dei dati ed effettua le necessarie elaborazioni per l'esecuzione della simulazione.

Context-Aware Simulation System for Smart Home
Il sistema genera informazioni contestuali associate a sensori virtuali in un ambiente domestico [11]. Questo simulatore è orientato ad una branca particolare dell'Ambient Intelligence che è *l'ubiquitous computing*, ovvero un modello di interazione uomo-macchina in contrapposizione al modello classico *desktop based*, il cui obiettivo è quello di aiutare ad organizzare le interazioni sociali ovunque e ogni volta che queste situazioni potrebbero verificarsi [12].
In questo contesto risulta necessario raccogliere informazioni dai sensori presenti nell'ambiente al fine di effettuare le opportune elaborazioni. L'architettura del simulatore è composta da quattro elementi principali: *l'interfaccia grafica* con la quale l'utente configura ed interagisce con il software. Un gestore delle regole che controllano la simulazione. Un gestore degli attuatori che interviene nel momento in cui una regola viene soddisfatta ed interviene sui dispositivi virtuali. Un gestore dei sensori il quale permette di simulare il comportamento dei sensori facendo a meno dei dati provenienti da sensori reali.

1.4 Il simulatore di Ambient Intelligence

L'aspetto principale che rende il simulatore di Ambient Intelligence sviluppato preferibile rispetto alle soluzioni già esistenti è quello di essere un simulatore general purpose che non si focalizza su un aspetto specifico del settore di studio e permette l'integrazione di qualsiasi tipologia di elementi. L'algoritmo che esegue la simulazione non effettua alcuna elaborazione ma ha solamente il delicato compito di stabilire l'ordine di esecuzione dei vari componenti che costituiscono la simulazione. Ciò che il simulatore fornisce è un mondo virtuale libero ed orientato all'Ambient Intelligence in cui la simulazione procede anche se non vi è nessun elemento nell'ambiente. L'obiettivo dello strumento è quello di fornire uno spazio di lavoro che imiti il mondo reale con delle semplificazioni tali da poterne tranne vantaggio nelle varie fasi di sviluppo di un progetto.
Queste caratteristiche garantiscono una notevole adattabilità agli scenari di simulazione ed è accompagnata da una struttura modulare aperta che permette di espandere il simulatore con nuovi moduli software fornendo così la più grande libertà possibile.
Un altro notevole vantaggio risulta essere l'autonomia dai dati di simulazione grazie al modulo di intelligenza artificiale che mediante una rete neurale genera dinamicamente le attività dell'utente virtuale, lasciando comunque l'operatore libero di fornire un proprio dataset. Il modulo rimuove così i limiti di lunghezza dell'intervallo temporale di simulazione coperto.

Capitolo 2

Architettura del simulatore

Nel presente capitolo verrà descritta l'architettura logica della simulazione ed i relativi componenti, illustrando come questi modellano il mondo reale e come è organizzata la gerarchia che li caratterizza. La seconda parte approfondisce la logica dell'algoritmo che sta alla base della simulazione e come questo caratterizza il modello di emulazione del mondo reale.

2.1 Progettazione

Il simulatore di Ambient Intelligence è stato progettato per realizzare simulazioni di ambienti pervasivi popolati da diverse tipologie di elementi volti principalmente a rilevare o modificare le proprietà fisiche dell'ambiente con l'obiettivo finale di mantenerne i valori all'interno dei parametri di comfort per gli utenti che vi si trovano. Gli utenti sono anch'essi modellati ed inclusi nella simulazione dove rappresentano uno degli elementi di principale importanza in quanto permettono di valutare l'influenza che una persona potrebbe esercitare sull'ambiente ed il conseguente impatto che si avrebbe sugli elementi in esso contenuti, contribuendo così a rendere più realistica la simulazione ed aumentare notevolmente gli scenari di utilizzo del simulatore.
La progettazione del processo di simulazione è stata effettuata in modo da renderlo completamente obiettivo. Il suo unico scopo è quello di emulare l'ambiente virtuale così come le leggi fisiche regolano un ambiente reale, con le dovute semplificazioni. Questo porta a designare due compiti fondamentali che caratterizzano il processo di elaborazione della simulazione, identificati nella valutazione dell'evoluzione nel tempo delle grandezze fisiche che caratterizzano l'ambiente virtuale e nell'eseguire le interazioni tra i vari elementi che compongono la simulazione.
L'algoritmo di simulazione non effettua direttamente alcun calcolo ma ha il delicato compito di scandire nel tempo le esecuzioni dei comportamenti di ogni singolo elemento. Facendo un'analogia, l'algoritmo può essere paragonato al direttore di un'orchestra sinfonica che ne dirige i singoli elementi senza però suonare uno strumento musicale.
L'obiettivo principale della progettazione è stato proprio quello di garantire la realizzazione a livello tecnico di tutti i requisiti di adattabilità ed espandibilità previsti offrendo la più alta libertà possibile. A tal fine è stata sviluppata una architettura software completamente modulare ed aperta in maniera tale che sia possibile espanderla con nuovi moduli e che questi abbiano il pieno accesso a tutti i parametri della simulazione in corso di svolgimento per far sì che possano svolgere senza alcuna limitazione il proprio compito.

2.2 Architettura software

La struttura del simulatore di Ambient Intelligence risulta basata su due elementi principali che sono la logica di simulazione ed i moduli in essa utilizzati. Dal punto di vista tecnico anche la logica di simulazione risiede in un modulo software pertanto risulta che tutti i moduli sono trattati allo stesso modo e sono visti allo stesso livello gerarchico, godendo tutti degli stessi privilegi per quanto riguarda la possibilità di osservare l'andamento della simulazione ed intervenire nel suo flusso di elaborazione. Questo aspetto della progettazione offre ai moduli la massima libertà di accesso ai dati e di spazio di manovra all'interno del processo di simulazione. L'unico vincolo che un modulo deve soddisfare per poter essere incluso nel processo di elaborazione è quello di rispettare l'interfaccia software che stabilisce il protocollo di comunicazione necessario ad una corretta interazione con il simulatore.
Il funzionamento logico è garantito da una gerarchia che stabilisce le relazioni di appartenenza dei vari elementi all'interno della simulazione in modo che ognuno di essi conosca il proprio contesto. Questa gerarchia sarà affrontata nel dettaglio nel capitolo successivo.

2.3 Elementi della simulazione

La simulazione è composta da diversi elementi che, in base alle proprie caratteristiche ed al modo con il quale sono coinvolti nel processo di simulazione, si possono classificare in due tipologie. Un componente viene identificato come *passivo* quando si limita al monitoraggio delle grandezze fisiche o di altri elementi della simulazione senza effettuare alcuna azione volta alla modifica dell'ambiente in cui opera. Un componente si classifica come *attivo* quando ha lo scopo di influenzare l'ambiente in cui opera secondo un determinato comportamento stabilito dai criteri con cui è stato programmato. Generalmente le modifiche sono il risultato di una elaborazione effettuata sui dati raccolti nei momenti precedenti della simulazione tramite gli elementi passivi presenti nell'ambiente virtuale.
Gli elementi di simulazione si possono suddividere in due ulteriori tipologie, questa volta legate alle caratteristiche interne che ne regolano il comportamento in quanto vi sono elementi che si limitano ad eseguire sempre la propria routine statica, come ad esempio i sensori e gli attuatori, ed elementi dinamici più intelligenti i quali hanno un comportamento non prevedibile a priori che si sviluppa durante la simulazione. Esempi di quest'ultima tipologia di componenti sono l'utente virtuale che opera all'interno dell'ambiente di simulazione e lo influenza intervenendo direttamente sulla regolazione degli attuatori presenti, oppure i moduli di Ambient Intelligence progettati proprio per gestire un ambiente pervasivo tramite il monitoraggio e l'elaborazione dei dati al fine di determinare le azioni da intraprendere per intervenire sugli elementi dell'ambiente.

2.3.1 Modelli della realtà

Gli elementi più comuni che caratterizzano gli scenari di Ambient Intelligence ed i quali risultano essenziali strumenti di base per gli studi del settore sono modellati anch'essi seguendo il principio general purpose in maniera tale da garantire la massima adattabilità.
I principali elementi che compongono una simulazione sono di seguito descritti e si approfondiranno nel capitolo 3 dedicato alla struttura software del simulatore.

- ambiente: elemento centrale della simulazione, rappresenta un ambiente pervasivo in cui si possono trovare svariati dispositivi per il monitoraggio e l'alterazione delle condizioni che lo caratterizzano;

- proprietà fisica: modella una generica grandezza fisica associata all'ambiente, può assumere diverse tipologie di valori ed è caratterizzata da una unità di misura;

- sensore: elemento che permette la rilevazione dei valori di una grandezza fisica presente nell'ambiente in cui viene collocato. Può essere configurato con autonomia limitata che si va esaurendo in base all'utilizzo oppure senza vincoli. Le letture fornite possono simulare anche la presenza di diverse tipologie di rumore;

- attuatore: elemento attivo che modifica una grandezza fisica dell'ambiente in cui viene collocato in base alla modalità operativa selezionata e ai comandi ricevuti dagli altri moduli;

- utente: modulo che impiega algoritmi di intelligenza artificiale per simulare il comportamento di una persona all'interno dell'ambiente virtuale. Ad esso viene associato un comportamento da adottare durante la simulazione ed ha la capacità di interagire direttamente con gli attuatori al fine di apportare alle condizioni ambientali le variazioni desiderate;

- modulo di Ambient Intelligence: modulo software che può essere introdotto nella simulazione per eseguire la gestione automatica delle condizioni ambientali così come avverrebbe in un contesto nel mondo reale. Possiede una capacità di intervento completa in quanto oltre al comando degli attuatori può anche agire sugli elementi passivi ad esempio per effettuare una gestione energetica.

2.3.2 Agenti software

L'architettura modulare, tra tutti i vantaggi offerti, permette al simulatore di Ambient Intelligence di poter introdurre nel processo di simulazione degli agenti software che eseguano i propri algoritmi di elaborazione ed abbiano facoltà di intervenire nella simulazione, allo stesso

modo di come si comporterebbero nel caso di una implementazione in un elemento del mondo reale. Nel contesto della simulazione si ha inoltre il vantaggio di poter presentare a questi agenti degli scenari limite che difficilmente si presenterebbero nella realtà così da valutarne la reazione e migliorare gli algoritmi che li caratterizzano.
Gli agenti software appena citati possono essere dei sistemi per l'Ambient Intelligence che in uno scenario del mondo reale hanno il compito di gestire gli ambienti in cui si trovano perseguendo diversi obiettivi come ad esempio l'efficienza energetica nella gestione intelligente di una abitazione oppure il mantenimento di determinati parametri ambientali all'interno di specifici intervalli. Proprio quest'ultimo caso è stato trattato in [13] dove è stato effettuato lo sviluppo di un'architettura multilivello capace di capire lo stato dell'ambiente pervasivo monitorato, al fine di prendere delle decisioni sulle azioni da eseguire per fornire le migliori condizioni ambientali possibili all'utente finale. Per lo scopo il sistema unisce ed analizza dati eterogenei mediante l'utilizzo di moduli adattivi dedicati. Quello appena riportato è un sistema che si integrerebbe in modo ottimale con il presente simulatore di Ambient Intelligence, dove lo svolgimento della simulazione fornirebbe i dati richiesti in input dal sistema e le azioni intraprese sarebbero i comandi di gestione inviati agli elementi della simulazione, dal punto di vista del simulatore un tale sistema è un modulo che si occupa della gestione intelligence dell'ambiente virtuale.
Un ulteriore vantaggio nell'effettuare la sperimentazione simulata degli agenti software è quello di poter mettere alla prova le interazioni che essi effettuerebbero in scenari ove siano presenti determinati componenti nell'ambiente, oppure l'interazione tra diversi agenti che controllano vari aspetti dello stesso ambiente. Si possono affrontare anche scenari dove risulti possibile simulare dei malfunzionamenti nei sensori usati per il monitoraggio o negli attuatori usati dagli agenti per intervenire sulle caratteristiche ambientali. Tali scenari permetterebbero di valutare con semplicità ed immediatezza un'enorme varietà di situazioni con l'obiettivo del miglioramento degli agenti che, essendo dei software, si prestano perfettamente all'integrazione con il simulatore. Tali scenari potrebbero rivelarsi utili anche nella fase di sviluppo degli agenti stessi al fine di metterne alla prova gli algoritmi.

2.4 Scenari di applicazione

La grande adattabilità permette al simulatore di Ambient Intelligence di poter essere impiegato in una grande quantità di scenari. Risulta molto interessante evidenziare anche altri aspetti che

possono risultare collaterali in questo ambito di studio. Si può impiegare il simulatore per studiare aspetti relativi ad esempio all'affidabilità dei moduli, al rilevamento ed alla gestione dei malfunzionamenti negli elementi della simulazione, alla gestione di attacchi alla sicurezza e così via.
I casi appena citati potrebbero essere ulteriormente estesi alle reti di elementi come ad esempio una rete di sensori per il monitoraggio dell'ambiente e quindi approfondire lo studio dell'affidabilità della rete, della gestione dei malfunzionamenti e della protezione da attacchi alla sicurezza. Nel caso di scenari più avanzati si potrebbero integrare nella simulazione degli strumenti specializzati per lo scopo. Ad esempio in [14] è stato presentato un sistema per la valutazione di modelli di gestione della reputazione che potrebbe essere interfacciato con l'insieme dei nodi presenti nell'ambiente virtuale oggetto di studio.

2.5 Logica di simulazione

L'algoritmo che regola la simulazione non risulta particolarmente complesso in quanto il compito che assolve è quello di dirigere, rigorosamente nell'ordine corretto, i calcoli che vanno a costituire l'elaborazione della simulazione.
Dopo una preliminare fase di inizializzazione volta alla preparazione dei moduli, la logica è costituita da un ciclo di istruzioni che vengono eseguite finché non si esaurisce l'intervallo temporale della simulazione. L'algoritmo va ad interpellare ogni singolo elemento che compone la simulazione, iniziando dagli ambienti, eseguendo così gli algoritmi che implementano la logica dei moduli. L'ordine che stabilisce quale tipologia di elementi viene elaborata per prima è importante in quanto caratterizza la modalità di propagazione degli effetti risultanti dalle singole elaborazioni.
Nella fase di progettazione del simulatore di Ambient Intelligence si è previsto un ordine di esecuzione che rispecchiasse il più possibile il comportamento del mondo reale. L'algoritmo, il cui pseudocodice è di seguito riportato, inizia con i calcoli relativi agli attuatori in maniera tale che i sensori, elaborati subito dopo, possano leggere i valori correnti ed aggiornati delle grandezze fisiche monitorate. Completato l'aspetto fisico dell'ambiente, il passo successivo calcola la carica residua delle batterie dei sensori, se previste, a seguito delle attività svolte per passare poi all'esecuzione dei moduli di Ambient Intelligence. Come ultimo passo dell'algoritmo vengono eseguiti gli algoritmi comportamentali dell'utente simulato, così che possa agire considerando lo stato aggiornato dell'ambiente. Le azioni effettuate si rispecchieranno su quest'ultimo nella successiva iterazione del ciclo di simulazione.

```
Algorithm 1: Simulation
if Simulation not ready to execution then
    stop and return
Initialize sensor's data
while time keep on do
    for Ambients in simulation do
        Calculate and apply Actuators effects
        Refresh sensors
        Calculate sensor's battery discharge
        Execute Ambient Intelligence module's logic
        Perform user actions
    Time go to next step
```

Algoritmo 1: Pseudocodice della logica di simulazione.

2.6 Parallelizzazione computazionale

L'architettura permette l'elaborazione parallela multi-thread delle simulazioni in caso di ambienti multipli. Un thread di elaborazione dedicato viene creato e lanciato per ogni ambiente, la cui elaborazione procede in autonomia alla produzione dei propri risultati.
Ogni thread, una volta ricevuta nella fase iniziale la configurazione della simulazione, procede in maniera indipendente e pertanto una tale elaborazione risulta possibile solamente nel caso in cui non vi sono moduli che prevedano l'interazione inter-ambiente. Lo sviluppo di un sottosistema di gestione delle interazioni multi-thread non è stato previsto in quanto non ritenuto un aspetto avente una rilevanza tale da sottrarre risorse allo sviluppo di altre funzionalità più importanti. Inoltre non potendo effettuare in maniera sincrona le letture di ambienti multipli, sarebbe stato necessario un ulteriore sistema di condivisione dei risultati parziali di ogni singolo ambiente. La scelta di non includere tali componenti in questo lavoro è stata poi avvalorata anche dal fatto che le simulazioni effettuate hanno sempre presentato dei tempi di elaborazione dell'ordine dei secondi, anche per lunghi periodi di tempo simulato, pertanto vengono efficientemente completate in tempi più che accettabili con i calcolatori odierni.

Capitolo 3
Struttura software

Il simulatore di Ambient Intelligence è stato sviluppato con l'obiettivo di creare un software dinamico ed adattabile agli innumerevoli scenari di cui il settore tratta. Il principale punto di forza che lo caratterizza è sicuramente la modularità che permette al software di poter utilizzare componenti esterni non previsti in fase di progettazione ed inserirli nel processo di simulazione, rendendo il software adattabile e scalabile.
Un modulo può essere un semplice elemento di simulazione oppure un software a sé stante che esegue la propria logica in autonomia ed in maniera indipendente dal simulatore di Ambient Intelligence. L'unico vincolo che qualsiasi modulo deve osservare per poter essere usato nel processo di simulazione è quello di rispettare l'interfaccia software di comunicazione la quale fa sì che il simulatore possa comprendere le richieste del modulo ed usarlo correttamente all'interno della simulazione. Ogni modulo ha inoltre accesso a tutti i dati di simulazione così che possa monitorare o interagire con gli altri elementi.
La struttura di interconnessione del simulatore permette ad ogni elemento di comunicare con un qualsiasi altro componente della simulazione e la quasi totalità dei componenti implementa le interfacce della relativa categoria di appartenenza. Queste interfacce software stabiliscono le *signature* dei metodi che devono essere implementati dalle classi così da garantire un protocollo comune.
Per riportare un semplice esempio pratico dell'approccio appena esposto si può osservare uno dei componenti più usati nel software, ovvero il *logger* degli eventi. L'interfaccia che definisce questo componente stabilisce le funzionalità fornite e per essere utilizzato richiede che vengano forniti determinati parametri tra i quali la tipologia di evento ed i relativi dettagli. Ogni singolo componente che fa parte del simulatore di Ambient Intelligence è predisposto all'utilizzo di un logger, ma la tipologia del logger usato è nota solo in fase di esecuzione in quanto è l'utente a scegliere se e quale logger usare, pertanto la struttura delle classi conosce l'interfaccia del logger e sa come interagire con esso a prescindere dalla tipologia che gli verrà assegnata. Possono esserci svariati tipi di logger che riportano gli eventi in flussi di output diversi, ad esempio su file, database, terminale oppure in formati diversi e filtrati per categoria di eventi.

3.1 Componenti

La struttura del simulatore di Ambient Intelligence è composta da diversi elementi i cui ruoli si suddividono in due categorie: una è a supporto della simulazione e l'altra garantisce il funzionamento del simulatore.
Gli elementi rispettano una gerarchia logica, come mostrato in Fig. 1, la quale rispecchia le caratteristiche dell'ambiente del mondo reale. L'elemento fondamentale risulta essere l'oggetto che modella la simulazione in quanto contiene gli oggetti della simulazione e la relativa configurazione, costituendo la radice dell'albero. Tuttavia, nella gerarchia il modulo di Ambient Intelligence può indifferentemente essere ricollocato al livello superiore nell'albero in maniera tale da poter gestire più ambienti. Il modulo ha accesso all'intera simulazione ed ha una relazione diretta con l'ambiente solo se vi si trova fisicamente contenuto nel modello che si vuole replicare. Possono infatti coesistere diversi moduli che afferiscono a diversi ambienti.

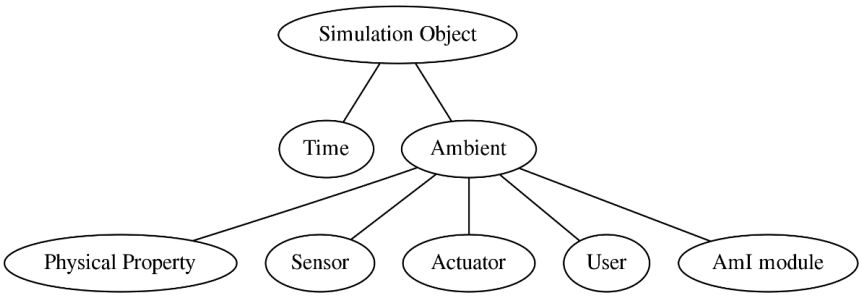

Figura 1: Gerarchia logica degli elementi di simulazione

Ogni elemento ha accesso agli elementi contenuti ma non vale il contrario: un elemento non può accedere ad un altro che si trova ad un livello superiore nell'albero, fatta eccezione come detto per il modulo di Ambient Intelligence il quale ha accesso all'elemento di simulazione e quindi a tutti quelli contenuti. La Fig. 2 riporta lo schema esaustivo della struttura dove si evince la collocazione degli elementi del simulatore di Ambient Intelligence.

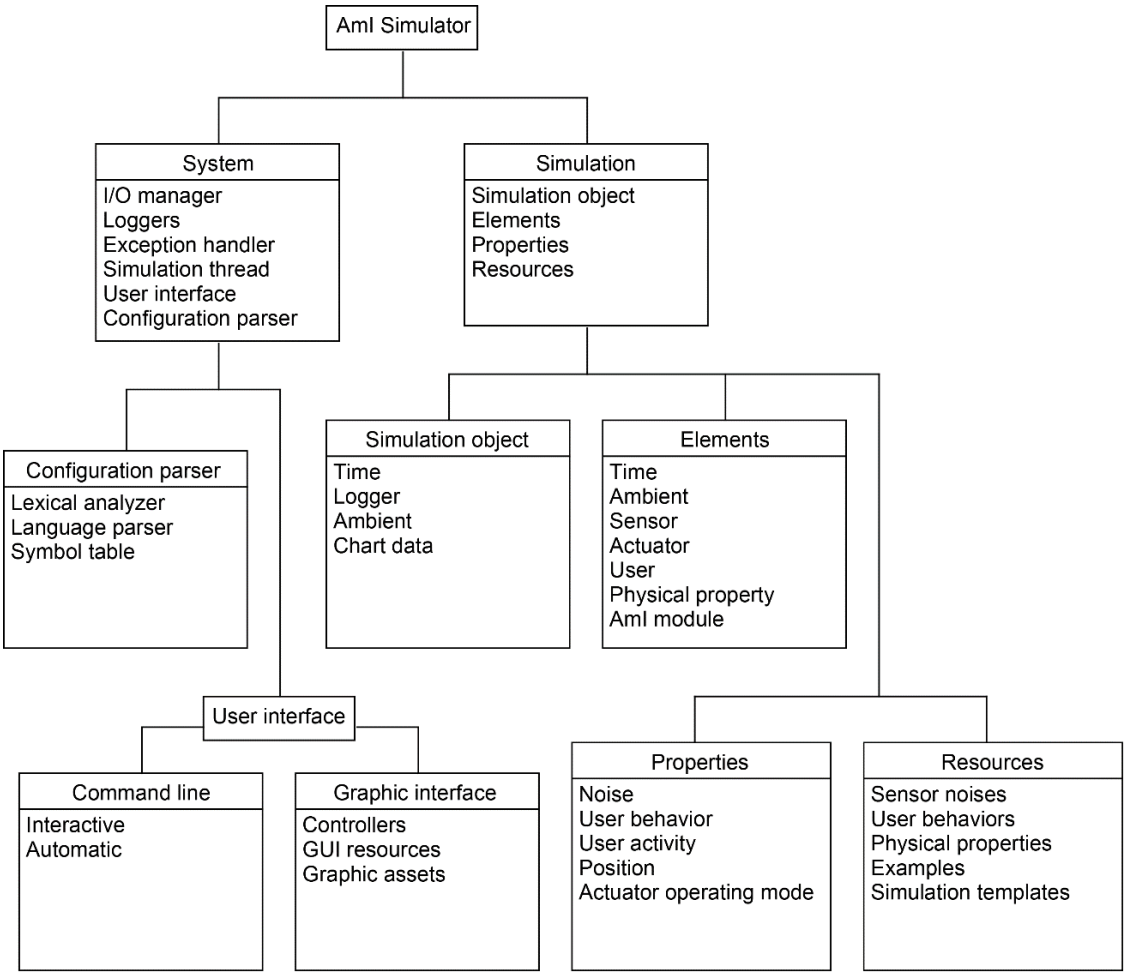

Figura 2: Schema della struttura software

3.1.1 Simulation object

La simulazione risulta composta da diversi elementi con una gran quantità di parametri di configurazione, pertanto si è progettato un oggetto software che racchiudesse tutte queste informazioni e rappresentasse l'intera struttura della simulazione. Questo approccio risulta essere in perfetta armonia con la struttura modulare facilitando notevolmente anche uno scambio di tali dati tra un modulo e l'altro qualora si rivelasse necessario, come è successo nel caso del *parser* del linguaggio formale di configurazione della simulazione. Il *parser* è stato progettato in modo da creare ed assemblare l'oggetto di simulazione durante l'interpretazione delle istruzioni del linguaggio modellando direttamente l'oggetto ed evitando così ulteriori passaggi necessari a comunicare le informazioni interpretate al simulatore.

L'oggetto prevede la specifica di un nome identificativo assegnato dall'operatore e tutti gli elementi di simulazione vengono memorizzati in apposite strutture dati rispettando l'ordine gerarchico. Un logger è obbligatoriamente richiesto dal costruttore della classe in quanto risulta necessario per indirizzare il flusso di log degli eventi della simulazione dove vengono riportati, in canali dedicati, tutte le operazioni effettuate a partire dalla configurazione, compreso

qualsiasi cambiamento come l'aggiunta, la rimozione e la modifica degli elementi. A vigilare su tutte le operazioni svolte vi è un *handler* delle eccezioni che nel momento in cui si verifica un errore lo segnala riportando i dettagli del problema riscontrato.
Lo smistamento delle diverse tipologie di log è a carico del logger, il *Simulation Object* in questo caso ha solamente l'onere di selezionare il canale più appropriato al dato da riportare basandosi sul contenuto dell'informazione e sul contesto in cui è scaturita. Ogni elemento della simulazione possiede il proprio logger di eventi anche se solitamente coincide con quello assegnato al Simulation Object. Questo aspetto permette di poter utilizzare output di log completamente differenti per ogni elemento.
L'oggetto di simulazione contiene anche le informazioni di dettaglio di ogni elemento destinate alla visualizzazione nell'apposita sezione dell'interfaccia grafica. Al fine di fornire tali informazioni in tempo reale, all'aggiunta di un nuovo elemento il Simulation Object memorizza le informazioni di interesse già formattate in base alla tipologia di elemento in un'apposita struttura dati ad accesso diretto. La struttura dati utilizzata è un ArrayList di stringhe in cui la chiave di una entry è l'identificativo dell'elemento in questione ed il valore è la stringa contenente le informazioni da fornire. Tali dati devono essere aggiornati in occasione di ogni cambiamento apportato ai parametri degli elementi pertanto è stato previsto un metodo eseguito ad ogni modifica o su richiesta che rigenera le informazioni di tutti gli elementi della simulazione andando così da aggiornare i dati. Questo approccio è stato scelto perché essendo la visualizzazione delle informazioni di simulazione una operazione abbastanza frequente nell'interfaccia grafica, si va ad utilizzare questa sorta di memoria cache evitando così di dover interrogare ogni volta tutti gli elementi per leggerne le proprietà, in particolar modo nei momenti in cui sono impegnati nel calcolo della simulazione. L'unico svantaggio è quello di dover tenere aggiornate queste informazioni centralizzate, compito che viene assolto dalla funzione dedicata la quale viene eseguita solamente quando si verifica un cambiamento.
La struttura dati più importante all'interno dell'oggetto di simulazione è sicuramente quella contenente gli oggetti *Ambient* che modellano gli ambienti virtuali. Questi a loro volta e secondo l'ordine gerarchico contengono ulteriori elementi con i relativi parametri di configurazione andando a costruire l'effettiva struttura della simulazione.
L'ultima struttura dati presente nel Simulation Object è adibita alla memorizzazione dei dati prodotti dalla simulazione i quali vengono utilizzati per la generazione dei grafici di output nell'interfaccia grafica del simulatore.
La classe, a differenza della quasi totalità delle altre, prevede un unico costruttore il quale richiede solamente i seguenti elementi fondamentali per la creazione della simulazione:

1. *Identificativo* arbitrario assegnato alla simulazione;
2. oggetto *Time* per la specifica dell'intervallo temporale di simulazione e del valore di ampiezza dei passi di calcolo della simulazione;

3. il *logger* da utilizzare per l'output delle informazioni e delle azioni relative alla simulazione, compresi i dati generati ed eventuali errori o eccezioni che si potrebbero verificare durante lo svolgimento dei calcoli.

I dati memorizzati nell'oggetto di simulazione sono accessibili mediante un'interfaccia costituita da appositi metodi per l'interscambio e l'aggiornamento dei dati.
La classe *SimulationObject* non possiede il semplice ruolo di contenitore della struttura della simulazione ma bensì è essa stessa a contenere gli algoritmi che permettono di eseguire i calcoli necessari allo svolgimento della simulazione. Il Simulation Object è stato progettato in questo modo per via della stretta correlazione che intercorre tra i dati della simulazione e l'algoritmo di elaborazione, questo infatti necessita dell'accesso completo agli elementi ed allora si è rivelata essere la soluzione più conveniente quella di implementare il ciclo di simulazione all'interno della classe SimulationObject, riducendo notevolmente le interazioni inter-classe necessarie per il calcolo di uno step di simulazione.

3.1.2 Time

La classe *Time* modella il tempo di simulazione e memorizza tutti i parametri necessari a determinare lo stato di avanzamento nel processo di elaborazione della simulazione. L'oggetto ricopre un ruolo fondamentale in quanto calcola i passi di simulazione determinandone l'intervallo di elaborazione.
Essendo il simulatore di Ambient Intelligence un'applicazione informatica, per sua natura lavora con grandezze discrete le quali idealmente campionano la corrispondente grandezza continua. Nel processo bisogna quindi stabilire l'entità del "passo di campionamento" che viene utilizzata dal simulatore per tarare gli algoritmi di calcolo delle grandezze coinvolte nella simulazione. Il valore del passo di simulazione è un numero intero positivo che specifica i minuti in tempo simulato di cui è composto uno step di elaborazione.
L'oggetto memorizza nel formato *Date*, nativo di Java, il timestamp che permette di determinare la data e l'ora di inizio della simulazione, specificata dall'operatore in fase di configurazione. Allo stesso modo viene specificata la data di fine simulazione, in corrispondenza della quale il processo di simulazione verrà arrestato. Tutte le informazioni relative all'intervallo temporale della simulazione vengono derivate da queste due informazioni senza memorizzare ulteriori dati che risulterebbero ridondanti.
Un ulteriore oggetto, anch'esso di tipo *Date*, memorizza il corrente punto temporale in cui il processo di simulazione è arrivato. Le altre variabili presenti contengono il valore del passo di simulazione, il numero progressivo dell'ultimo passo di simulazione calcolato ed una stringa contenente una eventuale descrizione assegnata alla simulazione, utile per fare delle annotazioni da consultare durante la navigazione dei risultati.

La classe prevede tre diversi costruttori. Il primo è di servizio ed è specificatamente volto ad effettuare il clone dell'oggetto, viene quindi fornito come argomento un altro oggetto *Time* in modo tale che tutti i parametri di configurazione e quelli relativi all'attuale stato di avanzamento vengano copiati.
I due rimanenti sono costruttori dedicati alla creazione di un nuovo oggetto *Time* a partire dai parametri specificati negli argomenti di input. Tali parametri prevedono obbligatoriamente le date di inizio e fine della simulazione in formato stringa, solitamente prelevato dalla relativa istruzione del linguaggio di configurazione la cui espressione regolare collegata effettua già il controllo di correttezza della sintassi. Queste stringhe sono passate alla classe Java *SimpleDateFormat* che si occupa della creazione del relativo oggetto nel formato nativo Date.
I due costruttori sono discriminati solamente dalla specifica del parametro relativo al valore del passo di simulazione: in uno è specificato mentre nell'altro è implicito e torna utile nel caso in cui non si conosce tale valore e quindi verrà assegnato in un momento, caso comune durante la generazione automatica dell'oggetto di simulazione. Il valore del passo di simulazione internamente alla classe è memorizzato in millisecondi.
I costruttori dell'oggetto prima di assegnare i valori specificati effettuano alcuni controlli di base sulla validità dai dati ricevuti, come verificare che la data di inizio simulazione non sia temporalmente successiva a quella di fine simulazione. Anche il valore del passo di simulazione viene sottoposto a verifiche volte ad assicurarsi che non sia negativo, nel qual caso viene arbitrariamente impostato ad un minuto notificando l'evento nel log di simulazione.
I metodi principali che vengono utilizzati per l'esecuzione della simulazione sono due, il primo ha la funzione di verificare che la data corrente sia ancora all'interno della finestra temporale di simulazione e quindi l'elaborazione deve continuare il processo ed il secondo metodo ha il compito di effettuare tutte le operazioni necessarie per portare l'oggetto *Time* al successivo stato temporale in relazione al passo di simulazione specificato. Queste operazioni includono il calcolo della nuova data e l'aggiornamento di quella corrente.
Completano le funzionalità offerte dalla classe i metodi di interfaccia necessari alla lettura ed aggiornamento dei vari parametri.

3.1.3 Ambient

La modellazione dell'ambiente virtuale avviene con delle semplificazioni ritenute un necessario compromesso per mantenere le caratteristiche general purpose della progettazione ed allo stesso tempo garantire la massima adattabilità possibile agli scenari che si possono presentare. Tali semplificazioni in particolare riguardano la conformazione fisica dell'ambiente relativamente a dimensioni, posizioni degli oggetti e materiali che lo compongono ma anche per quanto riguarda le proprietà dalle grandezze fisiche che ne regolano lo stato.

Queste scelte progettuali sono state adottate per evitare complicazioni talmente articolate da limitare in maniera importante l'usabilità del simulatore di Ambient Intelligence da parte degli operatori e compromettere l'adattabilità agli scenari, caratteristica posta come uno degli obiettivi progettuali principali. In una tale condizione la specifica sia della conformazione fisica che delle caratteristiche appartenenti ad un determinato ambiente avrebbe richiesto che l'operatore fornisse una notevole mole di dati durante la fase di configurazione di una simulazione, anche la più semplice, e nel caso in cui non si stia modellando un ambiente reale allora i dati richiesti sarebbero dovuti essere stimati in maniera tale da risultare coerenti richiedendo quindi all'operatore ulteriori conoscenze specifiche.
Inoltre tali dettagli dell'ambiente non si sarebbero potuti sfruttare a dovere senza dei complessi algoritmi basati su specifici modelli matematici il cui studio e la relativa implementazione esulano dal presente lavoro. Per riportare un esempio, la temperatura ambientale nel simulatore è considerata uniforme, ovvero in qualunque posizione dell'ambiente virtuale la si misuri in un determinato istante temporale assumerà lo stesso valore, ma in un ambiente reale non lo è affatto ed un modello dovrebbe tenere in considerazione le interazioni chimico-fisiche del fluido nell'ambiente di lavoro al fine di poter effettuare delle stime.
Tuttavia si è tenuto conto di questo aspetto e nella fase di progettazione si è predisposto il simulatore alla possibile introduzione di moduli avanzati adibiti a tali scopi e negli elementi di simulazione sono già disponibili parametri aggiuntivi come la posizione fisica, la categoria di appartenenza, la tipologia di grandezza fisica monitorata o influenzata correlata dalle informazioni sulle unità di misura, etc.
La classe che modella l'ambiente prevede una ulteriore funzionalità che emula la naturale tendenza a raggiungere la temperatura di equilibrio con l'esterno nel caso in cui sia specificata e non vi siano variazioni introdotte artificialmente dagli attuatori. L'algoritmo che regola questa funzionalità offre una logica semplice in quanto non viene impiegato dal ciclo di simulazione per non introdurre delle variazioni artificiose nei dati. Al contrario, è stato pensato come funzionalità di supporto ad eventuali moduli che gestiscono le interazioni multi-ambiente dove possibilmente la temperatura interna di uno costituisce la temperatura esterna di un altro.

3.1.4 Physical property

Questa classe modella le grandezze fisiche all'interno del simulatore, costituisce una delle componenti principali, detiene i valori di riferimento della grandezza e costituisce la fonte di verità della simulazione, cioè l'elemento da interrogare se uno dei moduli vuole conoscere il valore corrente della grandezza.
La struttura è dinamica e grazie all'utilizzo dei tipi generici di Java la classe può accettare diverse tipologie di dati il cui tipo non è conosciuto a priori. Le funzionalità offerte sono

semplicemente quelle di lettura ed aggiornamento del valore corrente assunto durante il processo di simulazione, la classe non effettua alcun tipo di manipolazione dei dati.
Oltre al valore della grandezza fisica la classe detiene anche le informazioni descrittive quali il nome assegnato, necessario per identificarla all'interno della simulazione, una descrizione opzionale e l'unità di misura utilizzata dai moduli per effettuare in maniera consapevole i calcoli nel caso in cui siano predisposti per la gestione di diverse tipologie di unità di misura. Queste informazioni sono riportate anche nei risultati della simulazione per rendere significativi i dati prodotti.

3.1.5 Sensor

La classe *Sensor* modella un generico sensore all'interno della simulazione il quale può monitorare una o più proprietà fisiche presenti all'interno dell'ambiente simulato in cui si trova. Il modello progettato prevede una moltitudine di configurazioni per ottenere diverse tipologie di sensori e ricostruisce abbastanza fedelmente le caratteristiche che presenta un sensore reale. I modelli che la classe permette di configurare vanno dalla versione basilare con il solo scopo di leggere il valore corrente di una proprietà fisica alla creazione di una sorta di nodo di sensori capace di monitorare diverse grandezze contemporaneamente.
Come tutti gli elementi della simulazione, al sensore viene assegnato un identificativo, una descrizione, ed è possibile assegnare anche un oggetto posizione il quale permette di definire i dati di localizzazione fisica all'interno dell'ambiente. È previsto un logger per le azioni ed i valori rilevati.
La classe modella e gestisce anche lo stato operativo del sensore. Se si spegne non verranno prodotte letture, condizione molto utile per valutare il comportamento degli altri elementi della simulazione, in particolar modo dei moduli di Ambient Intelligence. L'assenza di letture da parte di un sensore apre la via a svariati scenari come lo studio del comportamento del sistema in presenza di guasti o anomalie di uno o più sensori, dove l'elemento ha anche la facoltà di produrre letture errate o falsate dei valori ambientali, condizione che offre molti spunti ad ulteriori scenari nell'ambito della sicurezza del sistema simulando ad esempio l'introduzione di dati non attendibili o con fini malevoli con lo scopo di valutare la robustezza ed il comportamento degli elementi della simulazione. Da questo si evince la grande flessibilità che l'elemento e la struttura del simulatore di Ambient Intelligence possiedono grazie alla progettazione orientata in questa direzione che predispone l'applicazione al più ampio spettro di scenari possibili.
Un sensore reale però difficilmente riesce ad ottenere una lettura perfetta della grandezza fisica monitorata. Ciò si può verificare osservando diverse letture consecutive effettuate dallo stesso sensore le quali probabilmente presentano delle lievi variazioni l'una dall'altra. Il modello utilizzato tiene in considerazione variabili di disturbo affini all'aspetto informatico riguardanti

il rumore possibilmente introdotto dalle componenti del sensore stesso, tralasciando le variabili dovute ai fenomeni fisici quali la posizione relativa nell'ambiente, la pressione dell'aria, le condizioni di illuminazione etc. Il modello prevede l'applicazione di un algoritmo per l'introduzione di rumore ai valori che l'elemento sensore acquisisce dalla grandezza fisica. L'algoritmo che si occupa della generazione del rumore è un altro elemento del simulatore modellato dalla classe *Noise* dedicata il cui oggetto istanziato viene assegnato al sensore e si va ad inserire nel flusso dei dati subito prima dell'output del sensore.
L'algoritmo dell'oggetto *Noise* può trattare i dati come ritiene più opportuno applicando diverse tecniche più o meno raffinate e stabilite dalla specifica implementazione. Nello scenario in cui non si desidera introdurre un rumore, il comportamento dell'oggetto potrebbe semplicemente essere quello di restituire il valore originale ricevuto in input dal sensore senza alcun cambiamento, comportamento già utilizzato come azione di default.
Il modello del sensore è ulteriormente arricchito dalle funzionalità di gestione energetica. Questo aspetto permette di simulare la fonte energetica del sensore, la quale può essere illimitata o limitata come ad esempio una batteria. Le funzioni di gestione energetica effettuano delle stime ogni volta che il sensore si attiva per effettuare delle operazioni, calcolando il consumo della carica energetica.
Lo stato della batteria influenza anche il comportamento del sensore. Ad esempio verrà imposto lo spegnimento qualora si esaurisse, oppure il livello di carica potrebbe essere usato per introdurre una fluttuazione sull'errore applicato alle letture fornite così da raffinare ulteriormente la simulazione emulando ad esempio un maggiore errore in corrispondenza ad un basso valore di carica residua.
I valori energetici, come anche altri parametri del sensore, sono sempre a disposizione dei moduli della simulazione, proprietà pensata in modo particolare ai moduli di Ambient Intelligence. I parametri necessari alla specifica della batteria sono la sua capacità espressa in mAh, valore che il costruttore dell'oggetto assegnerà anche come valore di carica iniziale se non diversamente specificato, ed il valore stimato del consumo orario medio impiegato dall'algoritmo per stimare i consumi del sensore durante il suo utilizzo.

3.1.6 Noise

La classe che modella il rumore è stata progettata per gestire un insieme di tipologie di valori che sia il più ampio possibile così da poter far fronte alle più svariate applicazioni ed essere adattabile. Nell'implementazione attuale del simulatore di Ambient Intelligence l'elemento *Noise* è previsto che possa essere utilizzato solamente con la classe *Sensor* in quanto non risulta necessario ad altri moduli ma potenzialmente si può applicare a qualsiasi classe che tratta un flusso di dati. Per tale motivo si è scelto di progettare la classe in maniera generica così che possa essere applicata sia a numeri interi che decimali, sia a valori booleani che addirittura

anche a caratteri o stringhe di testo purché l'algoritmo di elaborazione sia in grado di trattarli correttamente.
Lo scopo dell'elemento è quello di simulare la presenza di un rumore nei valori forniti così da valutare l'influenza che questo produce sugli altri elementi della simulazione oppure semplicemente per rendere più realistico il modello. La parte principale dell'oggetto risulta quindi l'algoritmo di generazione del rumore il quale a partire dal dato in input effettua delle elaborazioni e fornisce in output un dato differente permettendo quindi di introdurre diverse tipologie di rumore nei dati.
La struttura della classe risulta abbastanza semplice, prevedendo un identificativo assegnato all'istanza dell'elemento rumore accompagnato eventualmente da una descrizione testuale, tralasciando le funzioni che permettono l'interfacciamento e la modifica dei suddetti parametri. L'unica funzione di interesse è quella dedicata ad eseguire l'algoritmo per la manipolazione del dato.
Il rumore applicato alle letture di un sensore può essere di qualsiasi tipologia si desideri, come ad esempio un rumore lineare, gaussiano, casuale, etc. Un modello pseudo realistico dovrebbe però generare un rumore secondo determinati criteri che producano valori additivi o sottrattivi su uno specifico intervallo limitato di parametri che tengano conto della situazione reale.

3.1.7 Actuator

La classe che si occupa di modellare l'elemento attuatore risulta un po' più complessa delle altre in quanto ha un comportamento definito dalla sua configurazione e dipendente dai parametri specificati, che viene inoltre interpretato durante la simulazione e non risulta statico. La funzione di base di un attuatore è quella di intervenire sulle grandezze fisiche di sua competenza per variarne il valore. Il modello dell'attuatore simulato è molto simile al funzionamento di un condizionatore d'aria (HVAC, heating ventilation and air conditioning) ovvero una macchina termica capace di immettere aria calda o fredda nell'ambiente.
Oltre a fornire tutte le caratteristiche di base che garantiscono le classi del simulatore quali l'identificativo, una descrizione, la posizione nell'ambiente ed il logger, la classe contiene ulteriori elementi che vanno a specificare e gestire lo stato dell'elemento. Necessarie al funzionamento dell'attuatore sono le modalità in cui questo è capace di operare le quali vanno specificate nella configurazione dell'elemento correlate dai parametri che le caratterizzano.
Allo scopo è stata dedicata la classe *ActuatorOperatingMode*, che modella proprio le modalità di funzionamento degli attuatori le quali vengono memorizzate all'interno dell'oggetto *Actuator* in una struttura dati ad accesso casuale quale è l'*HashMap*, utilizzando come chiave l'identificativo univoco della modalità operativa così da garantirne intrinsecamente l'univocità delle entry evitando ulteriori controlli e sfruttando questa proprietà anche per l'aggiornamento dei valori esistenti.

Una modalità operativa si compone di un identificativo univoco, della proprietà che va a modificare espressa mediante il suo identificativo ed accompagnata da un coefficiente che riassume i valori stimati di influenza che essa esercita sulla relativa grandezza fisica. Una tale semplificazione potrebbe risultare eccessiva in alcuni contesti, pertanto dove sia necessaria una maggiore precisione del comportamento dal punto di vista fisico si è prevista la possibilità di sostituire l'implementazione attuale con un'altra capace di gestire tali fattori.
Un analogo discorso può essere fatto per quanto riguardo la stima del coefficiente di consumo energetico che la modalità apporta quando utilizzata, anche se in realtà in questo caso la semplificazione risulta meno incidente in quanto risulta fattibile una stima ragionevole, a volte addirittura fornita dal produttore dell'attuatore che si sta modellando.
A differenza della classe sensore, la presente non prevede una fonte energetica limitata in quanto il consumo energetico è un fattore molto incidente e che caratterizza diversi studi del settore. La stima calcolata è orientata ad essere utilizzata dai moduli di Ambient Intelligence che valutino anche la gestione energetica.
L'ultimo aspetto che caratterizza la classe *Actuator* è lo stato di attività. Similmente alla classe sensore, è possibile impostare uno stato che determina se l'attuatore è attivo e quindi sta influenzando la simulazione, con la differenza che è necessario specificare quale delle modalità operative previste è quella attiva.

3.1.8 User

L'utente virtuale è sicuramente l'elemento più elaborato tra quelli che attualmente si trovano nel simulatore di Ambient Intelligence. La classe comprende come di consueto i parametri comuni a tutti gli elementi che sono l'identificativo, una descrizione testuale ed un logger. Vi sono poi diversi elementi specifici della classe quali a partire dalla proprietà fisica dell'ambiente percepita, cioè quella proprietà che l'utente monitora per decidere se eseguire delle azioni nell'ambiente. Solitamente la grandezza fisica di riferimento è la temperatura.
Basandosi quindi sul valore corrente della grandezza percepita l'utente sceglie se intervenire sugli attuatori e per evitare dei cicli di accensione e spegnimento degli attuatori si è previsto un valore di tolleranza, specificato sempre nella classe, il quale ricopre un ruolo fondamentale nella valutazione delle azioni dell'utente a fronte delle variazioni della proprietà percepita.
Al fine di garantire un'interfaccia comune tra gli elementi, così da risultare intercambiabili, è stato mantenuto anche in questa classe lo stato di attività che determina se l'elemento è attivo o disattivo in ogni momento della simulazione. In questo contesto non ha molto senso perché se un utente virtuale si trova all'interno della simulazione non si dovrebbe avere alcun controllo diretto sul suo comportamento ma potrebbe tornare utile in qualche contesto particolare e si è quindi deciso di mantenere tale funzionalità.

Potendo svolgere delle azioni, ci si aspetta anche che l'utente si sposti all'interno dell'ambiente e tra un ambiente simulato e l'altro, in caso di più ambienti previsti nella simulazione. Per questo motivo la classe tiene traccia dell'ambiente in cui correntemente si trova l'utente così da scegliere su quali elementi intervenire.
Completa la classe l'elemento che modella il comportamento, che costituisce l'aspetto principale dell'utente virtuale e regola le azioni svolte durante il processo di simulazione. La realizzazione è stata effettuata impiegando tecniche di intelligenza artificiale così da apprendere un modello comportamentale dai dati relativi alle attività di vita quotidiana di una persona reale e poter caratterizzare in maniera realistica il comportamento dell'utente virtuale. Questo introduce nella simulazione una componente di imprevedibilità moltiplicando così i possibili scenari di impiego.
Aspetto importante è che l'intero processo risulta autonomo ed indipendente da alcun tipo di dataset in input in quanto i dati vengono generati direttamente dal modulo. Tenuto conto della notevole complessità relativa alla progettazione e realizzazione di questo modulo, l'esposizione è affrontata nel dettaglio nel Capitolo 5.

3.1.9 Ambient intelligence

Il modulo dedicato all'Ambient Intelligence è stato progettato tenendo conto che molto probabilmente non sarebbe stato un modulo funzionale come tutti gli altri ma piuttosto un'interfaccia con un sistema esterno vista la grande complessità che solitamente caratterizza un tale sistema.
La classe è di fatto una struttura vuota che non effettua elaborazioni dei dati ma crea un flusso di comunicazione. Possiede un identificativo, un logger ed uno stato di attività ma a differenza degli altri elementi ha accesso all'intera simulazione ed agli ambienti in essa contenuti. Una parte è dedicata alle operazioni core, ovvero le operazioni di elaborazione fondamentali per la logica del modulo ed un'altra parte dedicata ad applicare alla simulazione le azioni risultanti dall'elaborazione.
L'attuale versione del simulatore di Ambient Intelligence non include un tale sistema in quanto lo studio e lo sviluppo richiedono un lavoro di altrettanta portata.
L'utilizzo di un modulo di Ambient Intelligence nella simulazione può essere effettuato impiegando sistemi già esistenti come ad esempio quello presentato in [13] dove è stato effettuato lo sviluppo di un'architettura multilivello capace di capire lo stato dell'ambiente pervasivo al fine di prendere delle decisioni sulle azioni da eseguire per raggiungere un obiettivo che soddisfi l'utente finale.

3.2 Componenti di sistema

Oltre agli elementi che modellano aspetti e componenti della simulazione vi sono altre classi dedicate al funzionamento del simulatore di Ambient Intelligence le quali forniscono funzionalità di supporto e non sono elementi della simulazione.

3.2.1 Logger

Il simulatore possiede un componente dedicato al logging degli eventi sia relativi alla simulazione ed alla sua configurazione sia relativi agli eventi ed eventuali errori a livello dell'applicazione. La modularità della struttura software permette di integrare qualsiasi tipologia di logger ed alcune sono già implementate. Tutti i logger sono intercambiabili in quanto rispettano l'interfaccia stabilita per la categoria la quale permette di utilizzare le diverse tipologie di logger allo stesso modo.
Alla creazione di un logger è necessario fornire l'oggetto *Time* che racchiude tutte le informazioni temporali riguardanti la simulazione così che il logger si possa allineare ad essa e riportare il momento esatto dell'occorrenza degli eventi nel tempo simulato.
L'interfaccia prevede diversi canali suddivisi per contenuto logico. Dato che un logger è capace di integrarsi con qualsiasi elemento del simulatore deve fornire le funzionalità necessarie a raccogliere correttamente tutte le tipologie di dati. La suddivisione in canali permette di suddividere facilmente i flussi di dati ed opzionalmente fornire anche output multipli. Ogni canale è caratterizzato da un *Tag* identificativo in formato stringa ed un formato di logging che ne permette l'esaustiva memorizzazione delle informazioni. Ad esempio, nel caso si verificasse un'eccezione non gestita, oltre al messaggio che la descrive vengono memorizzati i dettagli e lo *stacktrace* degli eventi che hanno portato all'errore.
I canali previsti sono: *Configuration,* dedicato a notificare gli eventi di creazione degli elementi della simulazione e dell'assegnazione o modifica dei parametri che li regolano; in *Simulation* vengono riportati gli eventi operativi della simulazione; i valori di output dei passi di simulazione sono riportati nel canale *Values* ed i cambiamenti degli elementi in *Events*. Gli eventi relativi all'operatività del software sono riportati nei canali *Errors*, *Debug*, *Generic* e *Raw*, quest'ultimo omette formattazione e tag identificativo.

Tutti questi canali si traducono in metodi dell'interfaccia i quali rielaborano l'input secondo regole specifiche per canale o come definito dall'implementazione dello specifico logger. In questo modo ogni funzione che rappresenta un canale ha solamente lo scopo di formattare o rielaborare le informazioni ricevute per poi passare alla funzione comune a tutti i canali del logger che effettivamente effettuerà le operazioni per la scrittura delle informazioni nel flusso di output previsto il quale può essere un file, un database, il terminale o l'interfaccia grafica. Il

funzionamento appena descritto è quello della modalità centralizzata dove le funzioni dei canali afferiscono ad un unico metodo di output che esegue materialmente la memorizzazione dei dati. La classe logger può funzionare anche in modo decentralizzato, lasciando che ogni canale effettui le proprie operazioni di memorizzazione dati in autonomia.
La classe fornisce anche delle funzioni di utilità per quelle tipologie di output che necessitano di operazioni preliminari all'utilizzo solitamente per la preparazione del canale al flusso dei dati e di finalizzazione come ad esempio la chiusura del database o di un canale di comunicazione remoto. Le operazioni preliminari si possono effettuare nel costruttore e per quelle finali è previsto un metodo dedicato che il gestore del simulatore dovrà eseguire prima del termine naturale di esecuzione del software che solitamente avviene subito dopo la fine del processo di simulazione.

3.2.2 Moduli di supporto

Exception handler

La gestione delle eccezioni che si possono verificare durante l'esecuzione del software è delegata ad un modulo centralizzato dedicato. In fase di progettazione si è scelto questo approccio per avere un controllo globale di errori ed eccezioni uniforme ed in un modulo così da garantire anche l'indipendenza e l'intercambiabilità dell'algoritmo. Essendo esterno alla classe che lancia un'eccezione, il modulo ha la facoltà di intraprendere la scelta più opportuna relativamente alla prosecuzione del programma e ha una visione più ampia dello stato del software che gli fornisce una maggiore capacità di intervento sui vari componenti.
La struttura è caratterizzata da una classe statica con metodi pubblici per permettere l'accesso globale a tutte le classi. Anche questa classe richiede un logger nella fase preliminare di inizializzazione che userà per notificare gli eventi e memorizzare i dati di errori ed eccezioni. Inoltre è già fornito il supporto all'interfaccia grafica e la classe autonomamente è capace di mostrare dei popup di avviso o errore dando anche la possibilità di espandere i dettagli dove è riportato ad esempio lo stacktrace nel caso si tratti di eccezioni.

Simulation thread

A supporto dell'elaborazione multithread della simulazione sono state create due classi di thread che il gestore principale sceglie di usare in base al contesto di elaborazione. In una classica simulazione vengono lanciati i thread della classe *SimulationThread*, mentre nel caso di esecuzione dell'interfaccia grafica vengono usati i thread della classe *SimulationThreadGUI*. La differenza tra le due tipologie consiste nell'integrazione che le elaborazioni grafiche comportano nel flusso di calcolo dell'algoritmo di simulazione le quali si rivelano necessarie per aggiornare in tempo reale l'interfaccia grafica ed interagire con questa, in particolar modo per quanto riguarda l'aggiornamento dei grafici.

3.3 Algoritmo di simulazione

L'algoritmo che dirige le operazioni di simulazione non è soggetto a cambiamenti anche a fronte di modifiche considerevoli ai moduli del simulatore, in quanto le operazioni logiche che lo regolano rimangono invariate. L'algoritmo ha il compito di eseguire in ordine i moduli di elaborazione che fanno procedere lo svolgimento della simulazione. Gli algoritmi che effettuano le effettive elaborazioni risiedono nei rispettivi moduli degli elementi coinvolti pertanto, grazie alla struttura modulare, la logica eseguita è sempre quella dei moduli i quali possono essere implementati e collegati con la massima libertà possibile, per questo motivo l'algoritmo principale di simulazione non necessita di aggiornamenti.
I calcoli del tempo effettuati dall'algoritmo si basano sui parametri contenuti nell'oggetto *Time* assegnato alla simulazione, sfruttando l'intervallo in esso specificato ed il valore del passo di simulazione, determina quando arrestare la simulazione.
L'oggetto di simulazione possiede un metodo per eseguire le istruzioni di simulazione e questo rappresenta il fulcro dell'intero processo. Inoltre implementando l'algoritmo di simulazione, la classe deve anche avere i necessari metodi di verifica della struttura volti ad assicurare una simulazione ben formata, coerente e che possieda gli elementi minimi per essere consistente così da poter eseguire correttamente l'algoritmo. Tale metodo di verifica preliminare viene eseguito nel momento in cui si richiede di avviare il processo di simulazione e controlla che vi sia almeno un ambiente presente. Le ulteriori verifiche riguardanti i moduli sono implicitamente effettuate dagli stessi, ad esempio le verifiche di correttezza dei parametri temporali di simulazione vengono effettuate nella classe *Time*. La correttezza della struttura viene poi garantita dalla gerarchia che la logica delle classi impone quindi non risulta necessario effettuare ulteriori controlli degli aspetti tecnici della simulazione.
L'algoritmo di simulazione è supportato da una funzione di inizializzazione dei sensori la quale viene eseguita una sola volta come operazione preliminare e si occupa di aggiornare i valori dei sensori presenti nei vari ambienti affinché siano allineati con i valori iniziali delle grandezze fisiche monitorate. Questa operazione è necessaria in quanto nell'algoritmo di simulazione ci sono delle azioni preliminari all'aggiornamento dei sensori che utilizzano proprio questi elementi ed i sensori quando vengono creati non possiedono dati ma conoscono solamente la grandezza fisica da monitorare. La richiesta dell'aggiornamento avviene in cascata secondo la gerarchia della struttura e la simulazione richiede a tutti gli ambienti di aggiornare i propri sensori.
Completata la fase preliminare si può procedere all'esecuzione vera e propria della simulazione il cui pseudocodice dell'algoritmo è riportato nel capitolo precedente e di seguito riproposto per comodità di lettura.

```
Algorithm 1: Simulation
if Simulation not ready to execution then
    stop and return
Initialize sensor's data
while time keep on do
    for Ambients in simulation do
        Calculate and apply Actuators effects
        Refresh sensors
        Calculate sensor's battery discharge
        Execute Ambient Intelligence module's logic
        Perform user actions
    Time go to next step
```

Algoritmo 1: Pseudocodice dell'algoritmo di simulazione.

L'algoritmo di simulazione vero e proprio inizia con l'istruzione del ciclo *while*, dove procede ad interpellare con una determinata sequenza gli elementi che compongono la simulazione ed aggiornando il tempo di simulazione gestito dal relativo oggetto *Time*.
Le interazioni tra i diversi elementi della simulazione avvengono in maniera diretta tra gli stessi senza l'ausilio dell'algoritmo di simulazione principale. Questo è possibile grazie alla modularità della struttura software la quale garantisce anche la raggiungibilità tra i vari elementi della simulazione.
Le iterazioni dell'algoritmo sono invece determinate dal numero di volte in cui è possibile avanzare all'interno dell'intervallo temporale di simulazione con il valore del singolo passo di simulazione. La condizione del ciclo principale più esterno verifica proprio questo avanzamento nel tempo di simulazione il cui oggetto *Time* ne stabilisce lo stato. Se risulta quindi possibile effettuare un ulteriore passo di simulazione l'algoritmo continua raggiungendo il successivo ciclo di elaborazione. Questo ciclo ha il compito di scorrere tutti gli ambienti presenti nella simulazione così da poter effettuare le azioni su ognuno di essi.
Nel corpo del ciclo *for* interno vi sono le azioni eseguite dall'algoritmo di simulazione sugli ambienti virtuali e l'algoritmo interagisce in modo diretto soltanto con questa tipologia di elementi della simulazione, i quali a loro volta agiranno in cascata con tutti gli elementi contenuti e secondo le operazioni richieste dall'algoritmo principale di simulazione. Queste operazioni vengono eseguite ad ogni iterazione e quindi ad ogni passo di simulazione.
L'approccio utilizzato per stabilire l'ordine con il quale devono interagire gli elementi della simulazione è volto a modificare dapprima l'ambiente con gli effetti dell'avanzamento del tempo di simulazione e quindi la stima degli effetti subiti dalle grandezze fisiche presenti. Poi interviene l'azione degli eventuali attuatori attivi i quali vanno a modificare le grandezze dell'ambiente e di conseguenza la modifica interessa i sensori presenti i quali a loro volta rilevano i nuovi valori assunti dalle grandezze fisiche. Raggiunto tale punto l'ambiente si può considerare stabile dal punto di vista delle grandezze fisiche per l'attuale passo di simulazione, pertanto si passa agli eventuali interventi che il modulo di Ambient Intelligence, se presente, decide di effettuare dopo aver analizzato la situazione attuale ed aver eseguito i propri calcoli.

Come ultima operazione viene eseguita la classe che modella l'utente all'interno della simulazione, il quale trova le condizioni ambientali aggiornate e se non si rivelano di suo gradimento potrebbe effettuare delle azioni volte alla modifica diretta, come ad esempio accendere o modificare la modalità di uno o più attuatori o, se previsto dal modulo di Ambient Intelligence, impartire dei comandi al gestore dell'ambiente per portare l'ambiente alle condizioni desiderate. Le azioni impartite avranno effetto al successivo passo di simulazione ovvero alla successiva iterazione dell'algoritmo dove viene ricalcolato il nuovo effetto sugli elementi.
I vantaggi della struttura modulare si rispecchiano anche nell'algoritmo di simulazione il quale può essere facilmente modificato al fine di cambiare l'ordine di simulazione oppure aggiungere ulteriori tipologie di elementi espandendo la simulazione. Sarebbe anche possibile modificare facilmente l'algoritmo per far affluire gli stati degli ambienti di simulazione direttamente ad un unico modulo di Ambient Intelligence centralizzato che gestirebbe tutti gli ambienti della simulazione.
La struttura del simulatore non impone alcun limite alla scalabilità, considerando che non tiene conto delle dimensioni spaziali degli ambienti nella modellazione di base attualmente in uso e pertanto a livello di elaborazione non vi è differenza nei dati trattati da parte del simulatore di Ambient Intelligence. Risulta inoltre agevole cambiare elementi e parametri coinvolti al fine di adattarsi agli scenari.

3.3.1 Varianti

La classe *SimulationObject* possiede anche delle varianti dell'algoritmo di simulazione per far fronte ai diversi contesti in cui il simulatore di Ambient Intelligence può essere impiegato.
La versione standard è quella appena esposta ed esegue l'intera simulazione seguendo l'ordine temporale fino alla fine dell'intervallo stabilito senza prevedere interruzioni tra una iterazione e la successiva. Qualora si rivelasse utile eseguire dei calcoli esterni oppure effettuare una simulazione un passo alla volta è possibile modificare facilmente l'algoritmo sostituendo il ciclo *While* che ne caratterizza l'avanzamento con la condizione che si reputa più opportuna allo scopo. A livello di elaborazione, in presenza di più di un ambiente virtuale il calcolo viene effettuato in serie per le proprietà intrinseche dell'algoritmo. Ciò non costituisce affatto uno svantaggio in quanto gli ambienti non si influenzano l'un l'altro ed anche i relativi calcoli sono indipendenti, così l'elaborazione in serie avrebbe lo stesso effetto di una elaborazione in parallelo.
Nella classe è stata prevista una variante dell'algoritmo che effettua in parallelo i calcoli dei singoli ambienti presenti nella simulazione implementando una struttura software *multithread*. L'algoritmo lancia un *thread* per ogni ambiente della simulazione il quale effettua i relativi

calcoli in autonomia fino al termine della simulazione. Questo permette di velocizzare notevolmente l'elaborazione della simulazione qualora ve ne fosse l'esigenza.
La stessa struttura dell'algoritmo multithread è utilizzata per la variante della simulazione che presenta i risultati tramite l'interfaccia grafica, questa oltre a svolgere gli usuali calcoli si occupa anche di generare i grafici volti a mostrare l'andamento nel tempo delle proprietà di interesse.
Un'ultima variante dell'algoritmo consente di effettuare la simulazione allo stesso modo di come viene effettuata nella versione di default ma con la differenza di calcolare solamente uno specifico ambiente. Questa funzionalità si rivela utile per un eventuale ricalcolo dovuto alla modifica di qualche parametro oppure ad una necessità che si può verificare nell'utilizzo del simulatore mediante l'interfaccia grafica o comunque da parte di un modulo esterno.

3.3.2 Integrazione esterna

Vi sono due tecniche per collegare alla simulazione un'altra applicazione integrandola come modulo. La prima è l'integrazione del codice sorgente così da diventare un modulo software direttamente implementato nel simulatore di Ambient Intelligence con il vantaggio che l'applicativo eseguibile non ha dipendenze esterne. L'altra è l'implementazione di un modulo di interfacciamento tra l'applicazione software che si desidera collegare alla simulazione ed il simulatore; questa tecnica prevede che il modulo faccia da tramite per garantire la comunicazione traducendo i comandi e convertendo i dati scambiati all'occorrenza. Tale modalità crea una dipendenza esterna al simulatore da tenere in considerazione per lo specifico scenario di utilizzo e distribuzione. Il principale vantaggio fornito da questo metodo di integrazione è quello di poter usare qualsiasi tipo di applicazione informatica, anche remota, scritta in qualsiasi linguaggio di programmazione; l'unico requisito richiesto è il rispetto dell'interfaccia software per la comunicazione necessaria alla corretta interazione tra le parti.
Le applicazioni che si possono collegare con la tecnica appena esposta non devono necessariamente essere locali nella macchina sulla quale è in esecuzione il simulatore di Ambient Intelligence ma possono anche essere remote. Il modulo che permette l'interfacciamento con un sistema remoto dovrebbe implementare anche un protocollo per la gestione della comunicazione sulla rete con tutti gli accorgimenti necessari alla gestione delle classiche problematiche che si possono presentare.

3.4 Linguaggio di programmazione

La scelta del linguaggio di programmazione utilizzato per lo sviluppo del simulatore di Ambient Intelligence è ricaduta su Java in quanto le caratteristiche che lo contraddistinguono ben si adattano alla struttura progettata.
Il linguaggio è stato sviluppato da *Sun Microsystems* poi acquisita da *Oracle Corporation* la quale si occupa attualmente del mantenimento e della distribuzione. La versione utilizzata per lo sviluppo del presente simulatore inizialmente è stata la *Java Standard Edition 8*, aggiornata in seguito alla versione 11. Java è un linguaggio di programmazione di alto livello general-purpose orientato agli oggetti. Tra i vantaggi offerti sono stati sfruttati il polimorfismo per la realizzazione di classi di simulazione capaci di adattarsi per utilizzare elementi diversi ed i tipi generici[1] per la realizzazione di elementi che non ponessero limitazioni sulla tipizzazione dei dati e potessero adattarsi agli svariati contesti di simulazione, estendendo tali libertà anche ad eventuali nuovi moduli software.
L'utilizzo del linguaggio di programmazione Java offre inoltre la possibilità di essere eseguito facilmente su diverse piattaforme hardware in quanto il codice viene elaborato dalla *Java Virtual Machine*, una macchina virtuale con l'onere di interpretare sulla specifica architettura hardware le istruzioni del linguaggio intermedio in cui è stato tradotto il codice sorgente, ovvero il *bytecode*, linguaggio che contiene le istruzioni dell'*instruction set* della *Java Virtual Machine*. Per quanto riguarda la distribuzione del software, Java permette la creazione di un archivio in formato JAR (Java Archive) il quale assembla i file contenenti il bytecode delle classi compilate, i relativi metadati associati, le risorse e le librerie utilizzate dall'applicazione.

JavaFX

La realizzazione dell'interfaccia grafica del simulatore di Ambient Intelligence è stata effettuata utilizzando la libreria *JavaFX* presente nella *Java Standard Edition*. Questa libreria aggiunge le funzionalità grafiche necessarie alla realizzazione delle interfacce grafiche e garantisce l'integrazione nativa con il codice Java, sostituendo e ampliando tutte le funzioni grafiche fornite in passato dall'ormai obsoleta libreria *Java Swing*.
La libreria è stata però disaccoppiata a partire dalla versione 11 di Java, pertanto *JavaFX* va inclusa come libreria esterna ed è proprio questo il motivo della migrazione dell'SDK effettuata così che il simulatore fosse in linea con la versione attuale al momento dello sviluppo. La nuova configurazione del progetto rende il simulatore di Ambient Intelligence pronto all'integrazione di altri moduli sviluppati con le più recenti versioni di Java. La nuova versione della libreria *JavaFX* è adesso fornita dal portale Open Source dedicato al progetto[2].

1. Java generics: astrazione sui tipi di dati che permette la definizione di tipi parametrizzati i quali vengono esplicitati successivamente in fase di compilazione.
2. JavaFX official website: Openjfx.io

3.5 Interfacce utente

La configurazione di una nuova simulazione può essere effettuata in diversi modi mediante una delle interfacce utente oppure in maniera automatica fornendo il file di configurazione. In alternativa alle interfacce si può agire mediante il codice di programmazione, infatti il simulatore di Ambient Intelligence prevede anche l'esecuzione di configurazioni preconfigurate via codice.
Le interfacce utente sono due, una in riga di comando particolarmente utile in applicazioni remote e l'altra è una interfaccia grafica guidata che segue l'utente tra i vari passi di configurazione della simulazione verificando in tempo reale la correttezza di ogni dato inserito.
L'utilizzo da riga di comando permette due modalità di configurazione, una automatica e l'altra manuale. È possibile avviare l'eseguibile passando come parametro il path ad un file di configurazione così da eseguire direttamente il processo di parsing e simulazione. La modalità manuale da riga di comando è stata progettata per essere interattiva con l'utente che la utilizza presentando un menù a singola selezione il quale permette di navigare tra le funzionalità offerte dal simulatore. Tramite questa modalità è anche possibile immettere manualmente il codice di configurazione della simulazione, il quale non viene interpretato riga per riga durante la scrittura ma solamente al rilevamento del lessema "end" che indica la fine dell'elenco delle istruzioni. Questo assicura la corretta interpretazione ed è necessario anche per la determinazione del contesto delle istruzioni così da verificare i vincoli e le regole grammaticali del linguaggio.

La possibilità di utilizzo da riga di comando permette notevoli vantaggi dovuti alla rapidità di esecuzione della simulazione e, di conseguenza, di produzione dei risultati usando magari un file di configurazione ma soprattutto perché permette l'integrazione del simulatore di Ambient Intelligence in altri sistemi software.
L'utilizzo mediante interfaccia grafica risulta sicuramente molto più comodo da parte dell'utente finale, sia per la facilità di immissione dei dati che per la comodità di consultazione dei risultati. Viene inoltre fornisce una procedura guidata che accompagna l'utente durante il processo di configurazione. Le funzionalità presenti sono le stesse di quelle dell'interfaccia a riga di comando ed i risultati sono alla stessa maniera prodotti in tempo reale con l'aggiunta della generazione dei grafici di evoluzione della simulazione dei singoli ambienti.
Le diverse tipologie di configurazione di una simulazione però non prevedono lo stesso livello di dettaglio a causa delle tecnologie coinvolte nella realizzazione. In particolare in caso di introduzione di nuovi moduli software nel simulatore, andrebbe aggiornata l'interfaccia grafica per fornire una sezione di configurazione dedicata al nuovo modulo, cosa non richiesta invece per l'interfaccia a riga di comando la quale utilizza il linguaggio di configurazione, ma anche quest'ultimo si dovrebbe aggiornare per poter creare e gestire in maniera automatica i nuovi moduli sconosciuti al simulatore fino a quel momento. In quest'ultimo caso diventa anche

necessaria una modifica alla grammatica del linguaggio formale se lo si vuole utilizzare per configurare in maniera automatica i nuovi moduli aggiunti al simulatore ed una tale operazione non è alla portata di tutti in quanto sono richieste specifiche conoscenze del settore. In un caso del genere, la via più breve per l'introduzione di nuovi moduli risulta la specifica mediante codice sorgente in quanto il simulatore prevede già dei modelli da seguire correlati con i relativi esempi. L'unico svantaggio di questo approccio è l'inevitabile necessità di competenze di programmazione da parte dell'operatore che intende introdurre il nuovo modulo software nel processo di simulazione.

L'interfaccia grafica permette la specifica di tutte le proprietà degli elementi della simulazione accompagnata da una limitata possibilità di aggiungerne di nuovi. Le principali schermate di configurazione sono di seguito riportate. La schermata iniziale (Fig. 3) permette di specificare i parametri di base della simulazione per poi passare alla creazione degli elementi contenuti (Fig. 4) con la possibilità di specificare ambienti multipli. Infine lo svolgimento dei calcoli avviene mostrando in tempo reale i grafici dell'andamento delle grandezze fisiche presenti suddivisi per ambiente di appartenenza (Fig. 5).

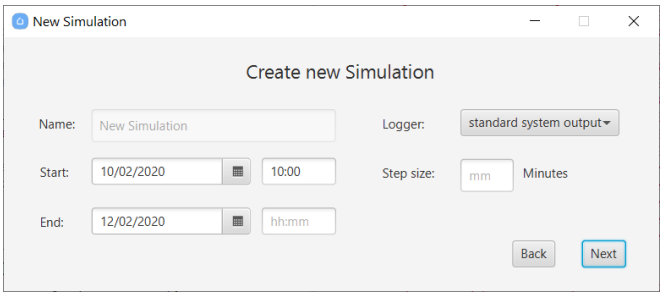

Figura 3: Interfaccia configurazione di base della simulazione

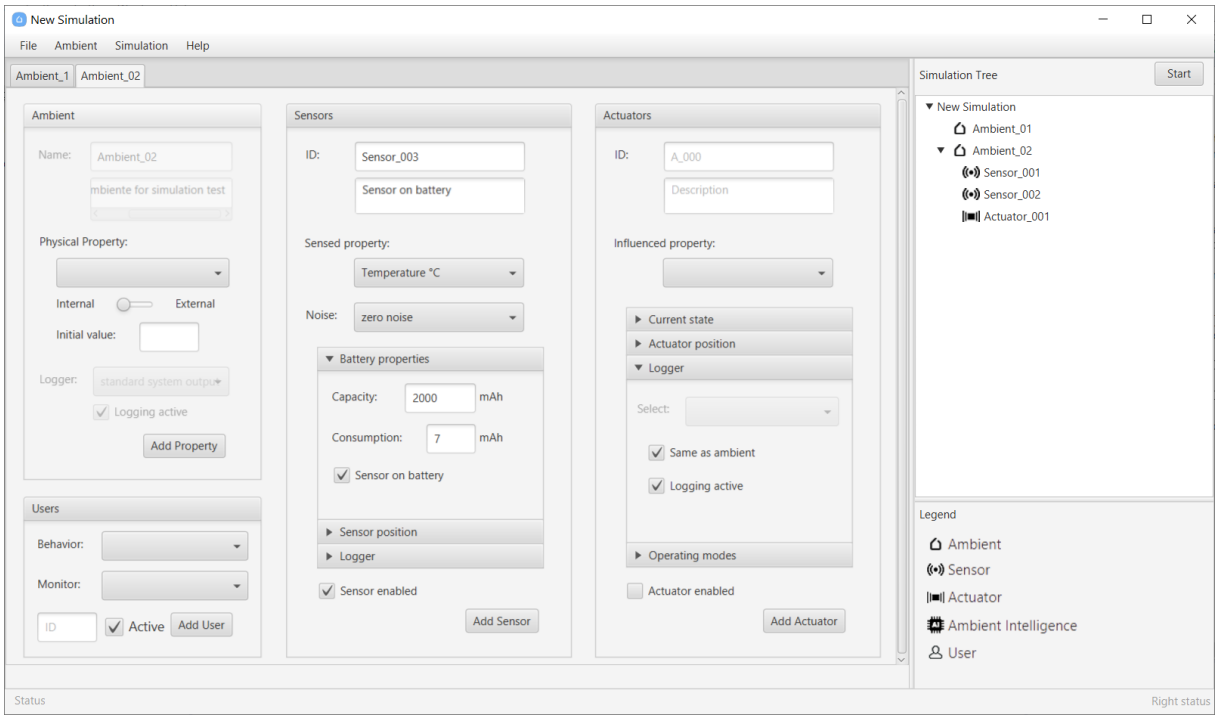

Figura 4: Interfaccia per la creazione degli elementi di simulazione

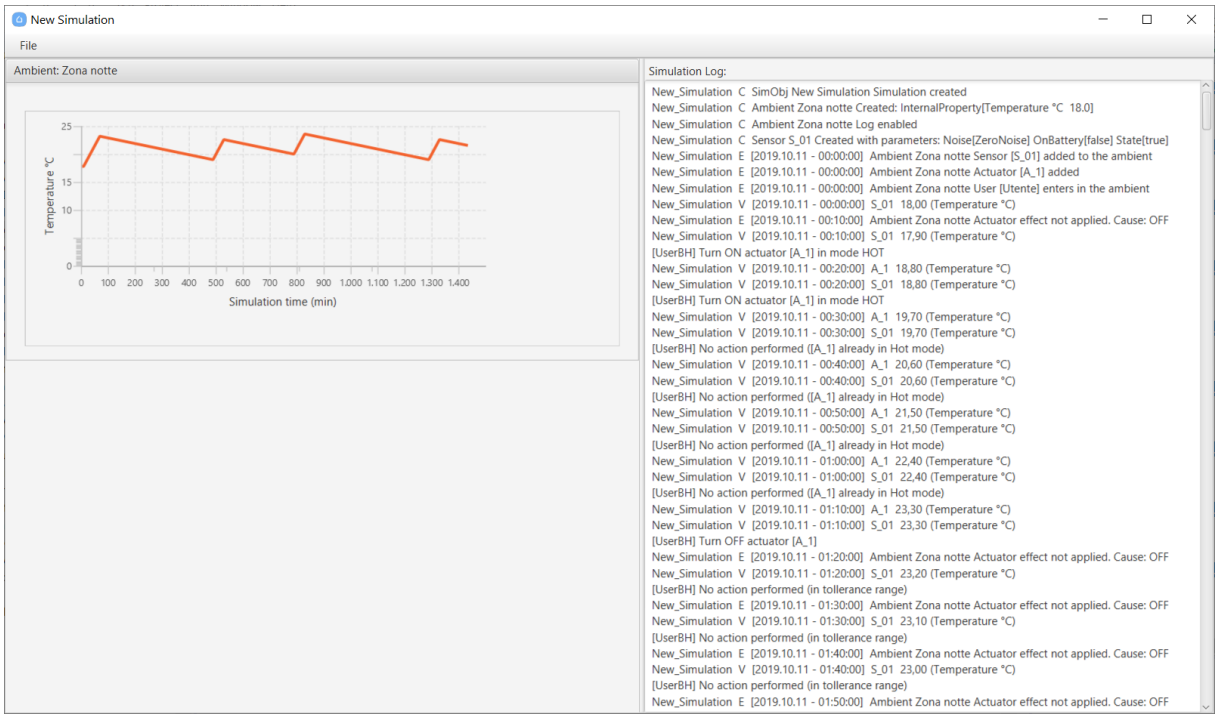

Figura 5: Risultati della simulazione

Capitolo 4
Linguaggio di configurazione

Nella fase di progettazione del Simulatore di Ambient Intelligence si è scelto di sviluppare un linguaggio di configurazione delle simulazioni volto a fornire una modalità di specifica delle istruzioni che fosse concisa, immediata ed espressiva.
Il linguaggio è impiegato sia nella logica interna al simulatore per la creazione e la configurazione di tutti gli elementi che compongono la simulazione sia per il salvataggio delle simulazioni che si desiderano eseguire nuovamente in futuro. Infatti le proprietà con cui è stato sviluppato il linguaggio ne permettono una notevole espressività di configurazione in poche istruzioni, il che favorisce la memorizzazione su file. I file di configurazione possono essere visti come dei veri e propri file di progetto i quali ogni volta che si importano vanno ad eseguire con un singolo passaggio una simulazione completa e già configurata.
Un ulteriore vantaggio del linguaggio di configurazione è quello di mantenere la completa leggibilità da parte degli esseri umani in quanto è stato pensato con l'obiettivo principale di permettere la configurazione della simulazione tramite riga di comando. Per tale motivo si è strutturato con brevi ma espressive istruzioni in maniera tale che risultassero il più concise possibile per garantirne la compattezza e la comodità di utilizzo anche su macchine remote, mantenendo allo stesso tempo una forma tale da poter essere compresa facilmente da parte dell'operatore. Allo stesso modo dei linguaggi di programmazione, anche se in maniera notevolmente più semplice, una volta imparate le regole che caratterizzano il linguaggio di configurazione della simulazione si può facilmente e velocemente scrivere senza alcun ausilio.
Tutte le caratteristiche appena esposte hanno reso il linguaggio particolarmente adatto alla memorizzazione su file per essere salvato o trasmesso, e questo ha portato allo sviluppo dei moduli di importazione delle configurazioni da file per eseguire delle simulazioni configurate in precedenza oppure eseguire nuovamente la stessa simulazione con magari dei piccoli cambiamenti senza quindi immettere nuovamente tutta la configurazione che la caratterizza. La simulazione potrebbe anche essere configurata da un altro software purché rispetti la logica del linguaggio.

4.1 Teoria dei linguaggi

I linguaggi di programmazione sono formalismi per descrivere dei procedimenti di calcolo alle persone e alle macchine [15]. Un insieme di istruzioni costituisce un programma il cui scopo è

quello di assolvere ad un determinato compito e per farlo non può essere utilizzato direttamente ma risulta necessaria una procedura di traduzione dal linguaggio sorgente a quello di destinazione capace di essere compreso ed eseguito da un calcolatore.
Le operazioni di traduzione sono eseguite da altri programmi chiamati *compilatori*. Questo processo è composto da tre fasi che effettuano l'analisi lessicale, l'analisi sintattica e quella semantica.
I linguaggi utilizzati dalle applicazioni informatiche sono *linguaggi formali*, le cui caratteristiche che li contraddistinguono sono quelle di avere un alfabeto composto da un numero finito di simboli. Ogni stringa del linguaggio è costituita da un numero finito di caratteri dell'alfabeto ed il linguaggio si definisce come un insieme finito o infinito di stringhe sull'alfabeto.
La prima fase di traduzione viene affrontata da un analizzatore lessicale il quale legge il programma in ingresso come un flusso di caratteri che raggruppa in sequenze complete e dotate di significato dette *lessemi*. Per ogni lessema l'analizzatore lessicale produce un *token* costituito da una coppia nome-attributo in cui il nome è un simbolo astratto utilizzato nella successiva analisi sintattica, mentre il valore dell'attributo punta all'elemento della *tabella dei simboli* relativo al token in esame [15]. La tabella dei simboli è una struttura dati usata per memorizzare le informazioni riguardanti i costrutti del linguaggio sorgente i quali vengono integrati nella tabella incrementalmente durante il processo di analisi.
L'insieme dei token generati viene passato alla seconda fase di traduzione che effettua l'analisi sintattica la quale si occupa di verificare la correttezza delle frasi del linguaggio. In questo processo l'analizzatore sintattico è chiamato *parser* ed utilizza in ordine i token ricevuti per costruire una rappresentazione intermedia ad albero la quale delinea la struttura grammaticale della sequenza dei token, ovvero l'*albero sintattico*. Le fasi successive utilizzano la struttura grammaticale qui generata come supporto per l'analisi del programma.
L'ultima fase di traduzione è costituita dall'*analisi semantica* la quale utilizza l'albero sintattico e le informazioni della tabella dei simboli per verificare la consistenza del programma sorgente rispetto alla definizione del linguaggio.
I compilatori oltre ai tre processi di analisi canonici effettuano ulteriori fasi di elaborazione prima di arrivare alla generazione del linguaggio target (Fig. 6), tali processi non sono tuttavia necessari nel simulatore di Ambient Intelligence in quanto, come sarà approfondito più avanti nel capitolo, si è sviluppato un modulo di parsing che nel momento in cui riesce con successo a comprendere il linguaggio di configurazione ricevuto in input ha già automaticamente costruito gli oggetti software che costituiscono gli elementi di simulazione e pertanto la simulazione stessa, così tutte le successive fasi di elaborazione non hanno motivo di essere eseguite. L'oggetto viene direttamente passato al simulatore il quale non deve far altro che eseguirlo all'interno dell'algoritmo di simulazione previa verifica di coerenza interna dei dati ricevuti.

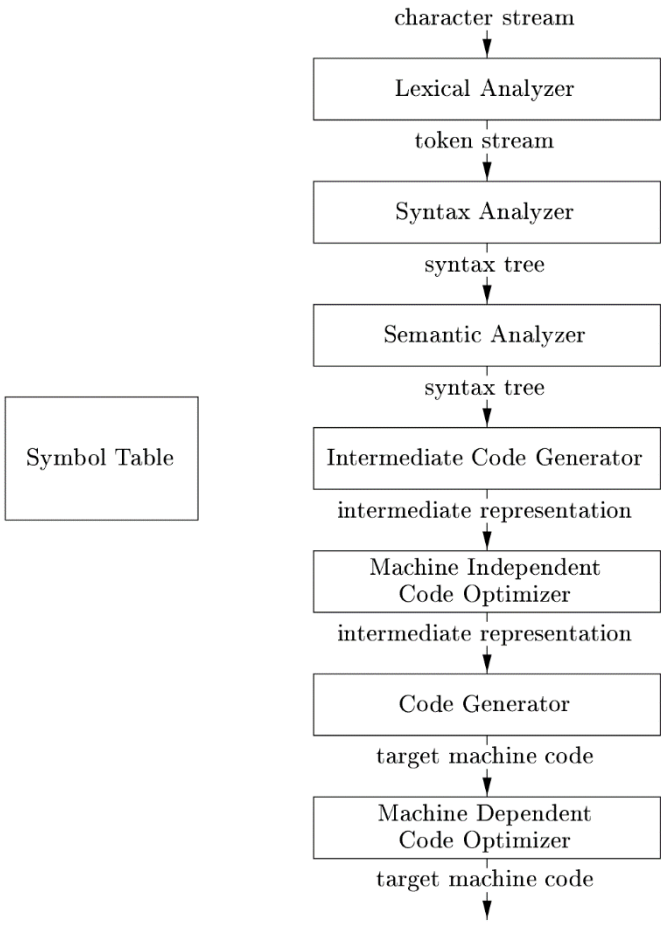

Figura 6: Fasi di un compilatore [15]

4.2 Proprietà

Il linguaggio progettato è interpretato, ovvero il simulatore è provvisto del modulo di *language processor* che esegue direttamente le operazioni specificate dalle istruzioni del linguaggio sorgente. L'interprete costruisce dinamicamente un oggetto di simulazione il quale si va formando durante il processo di parsing della configurazione.

Le regole di verifica della correttezza del linguaggio sorgente sono abbastanza restrittive e nell'eventualità che una di queste non venga soddisfatta viene sollevata un'eccezione interrompendo il processo di parsing. Non sono state previste regole di recupero dagli errori del linguaggio in quanto non è desiderabile continuare il processo. Se un errore si verifica a questo livello è indice di una configurazione che non rispetta la grammatica del linguaggio di configurazione.

Il *type check*, controllo dei tipi, viene eseguito in tempo reale essendo il linguaggio interpretato e gode anche della proprietà di *strong typing*, ovvero la capacità di prevenire l'esecuzione di

istruzioni o, in particolare nella presente applicazione, di argomenti aventi il tipo errato rispetto a quello che l'istruzione prevede e che quindi l'interprete del linguaggio si aspetta di trovare.
Le proprietà appena esposte permettono anche di soddisfarne una ulteriore, il linguaggio risulta essere *type safe*, cioè non permette che avvengano delle conversioni o qualsiasi altro tipo di operazioni che possano portare ad una condizione errata del software.

4.3 Strumenti utilizzati

Dopo aver progettato il linguaggio ed averne definito la *grammatica formale*, costituita da un insieme di *regole di produzione* che determinano come devono essere formate le stringhe di simboli dell'alfabeto affinché siano ritenute valide, si necessita di strumenti software capaci di applicare le regole stabilite ed eseguire le istruzioni descritte dal linguaggio.
A seguito di un'attenta valutazione degli strumenti disponibili sono stati selezionati due dei migliori aventi l'ulteriore caratteristica di essere nativamente compatibili con il linguaggio di programmazione utilizzato per sviluppare il simulatore di Ambient Intelligence.

4.3.1 JFlex

JFlex è un generatore open source di *lexical analyzer* in Java [17]. Si forniscono come input le specifiche del linguaggio costituite da un set di *espressioni regolari*[3] correlate a delle azioni da eseguire quando un lessema soddisfa una specifica espressione regolare.
Il programma generato in output prende il nome di *lexer*. Questo assolve tre funzioni:

- lettura del linguaggio ricevuto in input;
- confronto con le espressioni regolari specificate nella grammatica del linguaggio;
- esecuzione dell'azione associata alle espressioni regolari soddisfatte dalle stringhe analizzate.

I lexer generati da JFlex sono basati sugli automi a stati finiti (Deterministic Finite Automata[4]) i quali risultano veloci ed efficienti nel riconoscimento del linguaggio senza effettuare *backtracking*, azione computazionalmente costosa che consiste nell'analizzare oltre al token corrente anche uno o più token precedenti per cercare delle configurazioni grammaticali valide. Questo solitamente si effettua in linguaggi che implementano delle regole grammaticali complesse.

3. Espressione regolare: sequenza di simboli che definisce un pattern di ricerca per l'identificazione di stringhe.
4. Deterministic Finite Automata: è un sistema dinamico discreto tempo-invariante, cioè che si comporta sempre allo stesso modo, il quale accetta o rifiuta la stringa fornita eseguendo la sequenza univoca dei simboli che questa determina. Il termine deterministico si riferisce all'unicità della computazione.

Un ulteriore vantaggio nell'uso di JFlex consiste nel fatto che sia stato progettato per funzionare in maniera complementare con il generatore di parser CUP [18], anch'esso selezionato come strumento impiegato nello sviluppo del presente simulatore di Ambient Intelligence.

4.3.2 CUP

Acronimo di *Construction of Useful Parsers* è un generatore open source di parser in Java il quale si integra particolarmente bene con i lexer generati da JFlex ed è esso stesso scritto in Java.
I parser generati da CUP sono di tipo *LALR(1)*, ovvero *Lookahead LR parser*, i quali producono tabelle di parsing più piccole per le espressioni grammaticali rispetto ai parser LR canonici. La maggior parte delle grammatiche può essere espressa opportunamente con una grammatica LALR pertanto questa tipologia di parser risulta molto efficiente ed adatta allo scopo.
Tale tipologia di parser effettua un'analisi sintattica di tipo *bottom-up*, dove l'albero di parsing della stringa di input viene generato a partire dalle foglie e risalendo fino alla radice.
I parser LALR(1) usano un solo simbolo di input come *lookahead* ad ogni step per prendere le decisioni, non necessitano di backtracking ed utilizzano una direzione di scansione del testo in input da sinistra verso destra.
Il software CUP in input richiede la specifica della grammatica costituita dai simboli terminali, da quelli non terminali e dalle regole di produzione. Ad ogni produzione è collegata un'azione costituita proprio da codice Java, motivo principale che ha determinato la scelta di tale strumento per lo sviluppo del simulatore di Ambient Intelligence. Questo approccio permette di operare direttamente sugli oggetti di simulazione nativi e quindi realizzare un interprete del linguaggio che a fine processo fornisca l'oggetto software contenente la struttura della simulazione configurata e pronta per essere elaborata dal simulatore.
L'azione associata ad una produzione viene eseguita quando il parser effettua una *riduzione* applicando la suddetta regola di produzione. Nell'implementazione realizzata per il simulatore di Ambient Intelligence, tali azioni vanno a creare e configurare passo per passo i vari elementi specificati nel linguaggio sorgente i quali vengono aggiunti all'oggetto che modella la simulazione.

4.4 Integrazione tra parser e simulatore

Lo sviluppo del parser progettato specificatamente per il simulatore di Ambient Intelligence ha permesso di introdurre diverse ottimizzazioni nel processo di configurazione ma principalmente ha permesso una perfetta integrazione mantenendo al contempo valido il principio di modularità che caratterizza la struttura dell'intero progetto. Infatti il parser è anch'esso uno dei moduli

software utilizzati dal simulatore di Ambient Intelligence e comunica con esso mediante l'interfaccia di interscambio dati alla stessa maniera degli altri moduli.
La particolarità dell'integrazione consiste nel modo con cui viene creata la simulazione e gli elementi che la compongono. Questa procedura avviene all'interno del parser direttamente durante l'analisi del linguaggio di configurazione. In questo modo il parser utilizza direttamente le classi proprie del simulatore per creare gli oggetti di simulazione e costruirne la struttura man mano che le regole di produzione vengono soddisfatte. Questo aspetto ha permesso inoltre di fermare la catena delle fasi necessarie alla traduzione del linguaggio già alla seconda analisi ovvero quella sintattica, come mostrato in Fig. 6, risparmiando ulteriore elaborazione in fase di traduzione.
Uno dei processi di traduzione non effettuati è l'analisi semantica, in quanto la logica di costruzione dell'oggetto di simulazione che viene effettuata durante il processo di interpretazione risulta già garantita e validata dalle azioni eseguite nell'analisi sintattica la quale costituisce il passo immediatamente precedente. Nell'eventualità che un elemento aggiunto alla simulazione oppure dei parametri impostati non risultino coerenti, viene sollevato un errore ed il controllo passa direttamente al meccanismo di gestione degli errori del simulatore il quale, in accordo con le regole logiche e se possibile, decide se si tratta di una condizione che è possibile recuperare oppure non include l'elemento nella simulazione notificando l'azione intrapresa ma senza interrompere tutto il processo.
Come ulteriore verifica, essendo dati ricevuti da un modulo, il simulatore effettua sempre una diagnosi dell'oggetto di simulazione ricevuto per determinare se risulta ben costruito e quindi se possiede gli elementi ed i parametri essenziali per essere consistente così da poter eseguire la procedura di simulazione.

4.5 Grammatica

4.5.1 Fondamenti teorici

La grammatica è l'insieme delle regole di produzione del linguaggio formale. Queste regole descrivono come devono essere formate le stringhe, ovvero i token che costituiscono le sequenze di simboli dell'alfabeto del linguaggio, per essere considerate valide dal punto di vista sintattico.
La tipologia di grammatica del linguaggio di configurazione del simulatore di Ambient Intelligence è quella *context-free*, il linguaggio ottenuto risulta non contestuale ed ogni token rappresenta un simbolo atomico del linguaggio.
La funzione del parser consiste nel risolvere il problema del riconoscimento nei linguaggi formali non contestuali, ovvero il problema di stabilire se data una stringa ed un linguaggio non contestuale, questa stringa appartiene al linguaggio.

Una grammatica formale (*Backus Naur Form* [15]) è definita da una quadrupla $\langle V_T, V_N, S, P \rangle$ dove:

- V_T è l'insieme dei simboli terminali;
- V_N è l'insieme dei simboli non terminali (con $V_T \cap V_N = 0$);
- S è un simbolo appartenente a V_N detto assioma ed è il simbolo iniziale della grammatica;
- P è l'insieme delle regole di produzione, dove ogni produzione è un'espressione del tipo $A \rightarrow \alpha$ dove $A \in V_N$ e $\alpha \in (V_T \cup V_N)^*$.

La grammatica è associata al linguaggio attraverso il concetto di *derivazione* di una stringa. La derivazione utilizza le regole di produzione della grammatica come regole di riscrittura:
Data la grammatica $G = \langle V_T, V_N, S, P \rangle$ si ha:

- $\beta\alpha\gamma$ deriva direttamente da $\beta \boldsymbol{A} \gamma$ in G (denotato da $\beta \boldsymbol{A} \gamma \Rightarrow \beta\alpha\gamma$) se esiste una regola $A \rightarrow \alpha$ in P
- β deriva da α in G (denotato da $\alpha \Rightarrow^* \beta$) se esistono $\gamma_1 \in (V_T \cup V_N)^*, \dots, \gamma_k \in (V_T \cup V_N)^*$ con $k \geq 0$ tali che $\alpha \Rightarrow \gamma_1 \Rightarrow \cdots \Rightarrow \gamma_k \Rightarrow \beta$

Gli elementi della grammatica context-free sono:

- *simboli terminali*, ovvero i simboli elementari stabiliti dall'alfabeto del linguaggio i quali vanno a comporre le stringhe di testo;

- *simboli non terminali*, sintatticamente rappresentano delle variabili le quali denotano un set di stringhe che a sua volta conterrà ulteriori simboli terminali, non terminali o una combinazione di questi. Suddette variabili hanno la funzione di facilitare la generazione del linguaggio da parte delle regole grammaticali e sono proprio i simboli non terminali che impongono la struttura gerarchica del linguaggio la quale è fondamentale per la fase di analisi sintattica e traduzione;

- un simbolo non terminale particolare è il *simbolo iniziale* in quanto il set di stringhe che va a denotare è il linguaggio stesso generato dalla grammatica formale;

- le *Produzioni* di una grammatica specificano il modo in cui i simboli terminali e non terminali possono essere combinati per formare le stringhe del linguaggio.

Assiomi del linguaggio:

- data una grammatica, solo alcune stringhe possono essere derivate a partire dal simbolo iniziale, mentre altre non possono esserlo;
- l'insieme di tutte le stringhe derivabili dal simbolo iniziale è il linguaggio definito dalla grammatica;
- nella sequenza di derivazione per arrivare ad una determinata stringa, compaiono alcune stringhe formate da simboli terminali ed altre formate da simboli non terminali. La stringa finale sarà formata solamene da simboli terminali.

Si definiscono:

- *forma frasale* di G, qualunque stringa sull'alfabeto formato dai simboli terminali e da quelli non terminali ($V_T \cup V_N$) derivata dall'assioma, ovvero il simbolo iniziale S;
- *frase* di G qualunque stringa sull'alfabeto dei soli simboli terminali V_T derivata dall'assioma S;
- *linguaggio generato dalla grammatica G*, l'insieme delle frasi di G il quale si indica con $\mathcal{L}(G)$.

4.5.2 Specifiche

Identificare una parola del linguaggio non è un'operazione semplice in quanto solitamente un linguaggio non banale è composto sia da parole statiche identificative di un determinato comando sia da stringhe non prevedibili in fase di progettazione del linguaggio ma che certamente devono soddisfare determinate regole affinché siano valide per la grammatica del linguaggio e quindi comprensibili dal relativo parser.
Queste regole che una stringa deve soddisfare per essere un lessema valido sono verificate mediante un potente strumento che si rivela perfetto per lo scopo, ovvero le *espressioni regolari*. Una espressione regolare è una sequenza di caratteri che definisce un pattern di ricerca, ovvero uno schema che deve avere una determinata combinazione di simboli per essere ritenuta una stringa valida.
Anche nel caso più semplice in cui un lessema è identificato da una specifica parola, questa è verificata da una espressione regolare la quale identifica solamente una sequenza valida formata dall'esatto ordine dei caratteri che vanno a formare la relativa parola del linguaggio.

Il lavoro del lexer consiste esattamente nel sottoporre il testo in input alle espressioni regolari che definiscono i lessemi al fine di identificarli e riportarne al parser l'ordine di rilevamento. Nella tabella 1 sono elencate le espressioni regolari del linguaggio di configurazione del simulatore di Ambient Intelligence necessarie ad identificare i lessemi riconosciuti dal lexer,

mentre nella tabella 2 sono riportate le relative descrizioni accompagnate da un esempio del pattern che identificano e dal simbolo grammaticale del linguaggio ad esse associato, il quale verrà comunicato al parser.

ID	Regex
LineTerminator	\r|\n|\r\n
WhiteSpace	{LineTerminator}|[\t\f]
Date	((31|30)-(0[13578]|1[02])|30-(0[469]|11)|(29|(2[0-8]|1\d|0[1-9]))-(0[1-9]|1[0-2]))-([123][0-9]{3})
Time	([01][0-9]|2[0-3]):([0-5][0-9])
Word	(\w|-)+
End	end|END
Number	-?[0-9]\d*
PhProp	temperature_(K|C|F)|humidity (Questa espressione regolare varia in base delle proprietà implementate nel simulatore)
PropType	internal|external
AModeID	AM_[A-Z0-9]*
Decimal	-?\d\d?\.\d\d

Tabella 1: Espressioni regolari implementate nel lexer

Regex	Esempio	Grammar Symbol	Descrizione
"#".*	#commento		Identifica i commenti nel codice i quali vengono ignorati.
WhiteSpace	‘ ‘		Rileva simboli di fine riga, spazi e tabulazioni che vengono ignorati.
End	end	END	Segnala il raggiungimento della fine del testo in input.
simulation	simulation	SIM	Indica la specifica dei parametri di simulazione.
ambient	ambient	AMBIENT	Indica la specifica di un nuovo ambiente di simulazione.
sensor	sensor	SENSOR	Indica la specifica di un elemento di tipo Sensore.
actuator	actuator	ACTUATOR	Indica la specifica di un elemento di tipo Attuatore.

From	From	FROM	Indica la data di inizio simulazione a seguire.
;	;	SP	Simbolo separatore.
Date	21-01-2017	DATE	Formato indicante un giorno del calendario.
Time	10:02	TIME	Formato orario.
to	to	TO	Indica la data di fine simulazione a seguire.
step	step	STEP	Indica la specifica del passo di simulazione (step size).
Number	12	NUMBER	Rileva i numeri interi anche negativi.
AModeID	AM_HEAT	AMODE_ID	Identifica i nomi delle modalità operative di un attuatore.
Decimal	-3.2	DECIMAL	Rileva numeri decimali anche negativi separati da punto.
minutes	minutes	MIN	Indica la specifica a seguire del valore numerico in min.
PhProp	temperature_C	PH_PROP	Specifica una delle proprietà fisiche supportate dai moduli del simulatore.
PropType	internal	PROP_TYPE	Definisce se la proprietà fisica è interna o esterna all'ambiente di simulazione.
Word	Ambiente01	ID	Qualunque altra parola non riconosciuta è un ID.

Tabella 2: Descrizione e simbolo grammaticale delle espressioni regolari del lexer

Le espressioni regolari specificate nel lexer vengono per forza di cose applicate sequenzialmente, pertanto in un'eventuale condizione in cui una stringa soddisfa più di una espressione regolare, solamente la prima in ordine cronologico di applicazione che viene soddisfatta identifica il lessema, oscurando le successive espressioni regolari che sarebbero state soddisfatte dalla stringa in analisi. Quella appena esposta è una condizione da evitare la quale però non è improbabile che si verifichi se non si dedica una dettagliata attenzione nel progetto delle espressioni regolari che andranno ad identificare i vari lessemi del linguaggio, rendendo di conseguenza controproducente la potenza di questo strumento. In alcuni casi, invece, la suddetta condizione se opportunamente controllata può essere sfruttata a vantaggio del processo di riconoscimento del lessico come è stato effettuato nel lexer che riconosce il linguaggio di configurazione del simulatore di Ambient Intelligence. In particolare è stata applicata volontariamente per il riconoscimento degli identificatori assegnati agli elementi della simulazione ed alla simulazione stessa.

Non essendo possibile definire un pattern di riconoscimento mediante espressioni regolari degli identificatori che l'operatore del simulatore può scegliere di assegnare ai singoli elementi di

simulazione, si è progettato il linguaggio in maniera tale che se nessuna delle espressioni regolari viene soddisfatta identificando un lessema valido, l'ultima di queste andrà a catturare il lessema analizzato considerandolo come una stringa identificativa di un elemento, ovvero un nome assegnato all'elemento che l'operatore specifica. Questo approccio non porta a condizioni errate in quanto la grammatica formale del linguaggio è composta da rigide regole che non lasciano spazio ad alcuna ambiguità.

4.5.3 Scope delle istruzioni

Il simulatore di Ambient Intelligence permette la specifica di ambienti multipli nella simulazione e tale proprietà si rispecchia necessariamente nelle istruzioni di configurazione della simulazione in quanto sorge appunto la necessità di configurare ambienti multipli, gestire l'associazione di nuove proprietà fisiche ed aggiungere ai singoli ambienti gli elementi della simulazione.
Dato che l'esplicazione delle istruzioni di configurazione viene effettuata sequenzialmente, si devono stabilire delle regole di associazione per far sì che venga correttamente interpretata l'appartenenza di ogni nuovo elemento della simulazione specificato.
La gestione degli ambienti è stata progettata con una struttura avente *scope* locale per singolo ambiente la quale permette di creare delle sessioni di visibilità per ogni ambiente presente. Nel momento in cui viene specificato un nuovo ambiente, il relativo oggetto software che lo modella viene creato con i parametri specificati nell'istruzione. Tale oggetto viene memorizzato temporaneamente per la fase di parsing in una struttura dati creata specificatamente per lo scopo, che risiede nel parser al momento della creazione in quanto viene continuamente aggiornata dalle istruzioni di configurazione.
La suddetta struttura dati che contiene gli ambienti creati consiste in una lista di oggetti che li modellano; la tipologia scelta per la lista permette l'accesso casuale diretto che viene sfruttato per indirizzare in maniera diretta ed immediata l'ambiente che possiede lo scope corrente e sul quale si sta lavorando. L'indice dell'ambiente che detiene lo scope corrente è mantenuto in un'apposita variabile incrementata nel momento in cui un nuovo ambiente viene specificato nella configurazione.
Questo approccio fa in modo che tutte le istruzioni seguenti la dichiarazione di un nuovo ambiente facciano riferimento ed agiscano su tale ambiente il quale risulta essere l'ultimo creato in ordine cronologico.
La presente implementazione potrebbe essere facilmente modificata per consentire l'accesso a qualsiasi ambiente dichiarato da parte delle istruzioni che quindi potrebbero essere disposte senza vincoli di ordinamento nel flusso della configurazione, ma si è scelto di non permettere tale funzionalità in quanto renderebbe le istruzioni più complesse dovendo per ognuna specificare a quale ambiente si riferiscono perdendo così la semplicità e brevità del linguaggio.

Ulteriore aspetto positivo della scelta effettuata è quello di poter vedere il codice di configurazione della simulazione a blocchi, dove ad esclusione del primissimo blocco il quale si occupa di configurare i parametri generali di simulazione, i successivi sono blocchi indipendenti che rappresentano gli ambienti dove in ognuno di essi vi sono tutti i parametri, le proprietà e gli elementi del singolo ambiente favorendo notevolmente la leggibilità della configurazione ed isolando visivamente le istruzioni che si riferiscono ad ambienti diversi.
Proprio per mantenere la predisposizione del linguaggio all'accesso casuale agli ambienti creati durante il processo di simulazione, i relativi oggetti che li modellano all'interno del simulatore, temporaneamente memorizzati nel parser, vengono aggiunti all'oggetto di simulazione solamente alla fine di tutto il processo di parsing e configurazione in modo tale che possano essere aggiornati in qualsiasi momento durante tale processo e solamente alla fine vengono definitivamente assegnati alla simulazione prima di consegnare l'oggetto software completo al simulatore, momento dal quale il parser non ha più alcun controllo sulla simulazione e termina il proprio lavoro.

4.5.4 Grammatica del linguaggio di configurazione

Le tabelle di seguito riportate contengono i simboli e le produzioni che regolano la grammatica del linguaggio di configurazione del simulatore di Ambient Intelligence seguite dalla descrizione delle produzioni.

SIMBOLI TERMINALI	
SIM	Creazione e specifica dei parametri di simulazione.
FROM	Data di inizio.
TO	Data di fine.
DATE	Formato data.
TIME	Formato ora.
SP	Separatore di istruzioni.
STEP	Specifica dello step size di simulazione.
PH_PROP	Creazione di una nuova proprietà fisica.
PROP_TYPE	Specifica della tipologia di proprietà fisica.
NUMBER	Numero intero.
MIN	Numero intero rappresentante minuti di tempo.
END	Fine delle istruzioni in input.
AMBIENT	Creazione di un nuovo ambiente nella simulazione.
SENSOR	Creazione di un nuovo sensore nell'ambiente corrente.

ID	Identificativo di un elemento assegnato dall'operatore.
ACTUATOR	Creazione di un nuovo attuatore nell'ambiente.
DECIMAL	Numero decimale anche negativo separato da punto.
AMODE_ID	Specifica una nuova modalità operativa dell'attuatore.

Tabella 3: Elenco e descrizione dei simboli terminali della grammatica

SIMBOLI NON TERMINALI	
simulation	Parametri di simulazione.
expression	Generica espressione del linguaggio e simbolo iniziale della grammatica.
ambient_new	Specifica un nuovo ambiente di simulazione.
ambient_cfg	Specifica i parametri dell'ambiente corrente.
new_sensor	Aggiunge un nuovo sensore all'ambiente corrente.
new_actuator	Aggiunge un nuovo attuatore all'ambiente corrente.
actuator_mode	Aggiunge una nuova modalità operativa all'attuatore specificato.

Tabella 4: Elenco e descrizione dei simboli non terminali della grammatica

PRODUZIONI			#
expression	→	simulation SP expression	1
	|	END	2
	|	ambient_new SP expression	3
	|	new_sensor SP expression	4
	|	new_actuator SP expression	5
	|	ambient_cfg SP expression	6
	|	actuator_mode SP expression	7
simulation	→	SIM FROM DATE TIME TO DATE TIME	8
	|	SIM STEP NUMBER MIN	9
	|	SIM ID	10
ambient_new	→	AMBIENT ID PH_PROP NUMBER	11

ambient_cfg	→	AMBIENT PH_PROP PROP_TYPE NUMBER	12
new_sensor	→	SENSOR ID PH_PROP	13
new_actuator	→	ACTUATOR ID	14
actuator_mode	→	ACTUATOR ID AMODE_ID DECIMAL DECIMAL PH_PROP	15

Tabella 5: Produzioni della grammatica formale

Facendo riferimento al numero progressivo riportato nella Tabella 5, segue la descrizione delle azioni che vengono eseguite dalle produzioni della grammatica saltando le istruzioni di creazione degli elementi (dal numero 3 al 7) le quali risultano auto esplicative. Alcune di queste non hanno un'azione correlata alla loro riduzione in quanto contengono solamente simboli non terminali i quali vengono utilizzati per specificare le regole grammaticali del linguaggio formale.

1. È una produzione della grammatica costituita da simboli non terminali e quindi non direttamente specificabile nella scrittura del codice di configurazione del simulatore di Ambient Intelligence. Questa produzione viene ridotta alla fine del processo ed essendo l'ultima ad essere eseguita, l'azione ad essa associata restituisce al simulatore l'oggetto di simulazione creato, il quale modella la simulazione e contiene tutti gli elementi ed i parametri specificati.

2. La produzione viene ridotta quando nel linguaggio è riscontrato uno dei simboli di chiusura che indicano la fine dell'elenco delle istruzioni. Questa produzione esegue le operazioni necessarie a concludere il processo di configurazione, aggiungendo all'oggetto di simulazione le versioni definitive di tutti gli oggetti che modellano gli ambienti creati durante il processo di parsing del codice, i quali a loro volta contengono tutti gli elementi configurati che afferiscono ad essi.
   Questo simbolo è stato utilizzato per altri due motivi. Fornisce un ulteriore ausilio di lettura della configurazione da parte dell'operatore e predispone il linguaggio alla specifica di simulazioni multiple sullo stesso input.
   Anche a questa produzione è stata associata l'azione di finalizzazione delle operazioni preliminari al termine del processo di parsing il quale si conclude precisamente con la produzione #1, quindi va a ricoprire una utilità pratica nel linguaggio e grazie alla sua presenza è possibile specificare un set di simboli di chiusura, come è stato effettuato, o una combinazione di essi.

8. Creazione della simulazione: la produzione crea l'oggetto di simulazione con i parametri relativi al tempo i quali stabiliscono data ed ora di inizio e fine della simulazione. Queste informazioni vengono specificate durante la creazione dell'oggetto che modella il tempo simulato il quale viene a sua volta assegnato alla simulazione in fase di creazione.
   Il costruttore dell'oggetto che modella la simulazione necessita anche di un ulteriore parametro non opzionale per poter eseguire la simulazione, il passo di simulazione il quale determina gli intervalli di campionamento delle variabili di simulazione e l'intervallo di stima dei calcoli effettuati, pertanto il parser in un primo momento specifica un valore di default.
   L'ultimo parametro richiesto è il logger che verrà utilizzato sia nella simulazione che in tutti gli elementi contenuti. Nel caso di utilizzo del parser mediante linea di comando viene utilizzata una istanza del logger il cui output viene effettuato sul terminale stesso, invece nel caso di utilizzo dell'interfaccia grafica viene utilizzato un logger il cui output viene riportato, sempre in forma testuale, nella finestra dedicata.
   La scelta del logger da utilizzare viene effettuata in maniera automatica rilevando se è in esecuzione il processo che gestisce l'interfaccia grafica oppure l'output verrà riportato sul terminale.

9. Imposta il valore del passo di simulazione specificato espresso in minuti.

10. Assegna alla simulazione l'identificativo specificato come argomento dell'istruzione. Tale valore verrà usato sia per l'identificazione della simulazione da parte dell'utente sia per presentare i risultati associandoli al nominativo assegnato.

11. L'azione associata alla produzione è quella di creare un nuovo ambiente da aggiungere alla simulazione. L'oggetto che modella l'ambiente non viene aggiunto direttamente alla simulazione ma è memorizzato in una struttura dati temporanea contenuta nel parser per essere aggiunto alla simulazione solamente alla fine del processo di configurazione, quando non può più avvenire alcuna modifica alla struttura già creata.
    Gli argomenti richiesti per la creazione di un nuovo ambiente sono il nome identificativo specificato nell'istruzione, il logger da usare, assegnato automaticamente dal parser così da essere della stessa tipologia scelta per la simulazione. Infine è necessario assegnare una proprietà fisica di base, che costituisce un'eccezione alla creazione diretta degli oggetti di simulazione da parte del parser essendo un'operazione complessa da svolgere in maniera automatica. Pertanto l'istruzione prevede solamente l'indicazione dell'identificativo della proprietà desiderata in maniera tale che il parser possa interrogare il simulatore di Ambient Intelligence per ottenere l'oggetto software che

modella quella specifica proprietà fisica. Pertanto nel caso in cui si voglia utilizzare una proprietà fisica nuova per il simulatore è necessario prima implementarla, altrimenti il parser fermerà il processo di configurazione segnalando l'errore non recuperabile.

12. La produzione assegna la proprietà fisica specificata nell'istruzione, se correttamente ottenuta dal simulatore, all'ambiente attivo ovvero quell'ambiente che possiede lo scope corrente. L'istruzione fornisce tutti i parametri necessari a configurare una proprietà fisica i quali prevedono il valore iniziale della proprietà nel momento in cui la simulazione viene avviata e la tipologia della stessa, ovvero se si tratta di una proprietà interna oppure esterna all'ambiente.

13. Creazione di un nuovo oggetto sensore ed aggiunta all'ambiente del corrente scope. Anche in questo caso, come per tutti gli elementi viene assegnato lo stesso logger della simulazione. L'identificativo è invece specificato nell'istruzione così come l'identificativo della proprietà fisica.

14. Analogamente alla precedente viene creato un oggetto che modella un attuatore e viene aggiunto all'ambiente che detiene lo scope corrente. Il logger assegnato è quello della simulazione e nell'istruzione è necessario specificare come parametro solamente un identificativo per l'attuatore. Anche questo caso costituisce un'eccezione alla modalità di creazione degli oggetti software da parte del parser essendo la configurazione di un attuatore molto complessa. I dettagli dell'implementazione del modulo dedicato alla modellazione della struttura di un attuatore sono esposti nella sezione 3.1.7. Il motivo principale è la specifica dei valori delle modalità operative che regolano il funzionamento dell'elemento nella simulazione e la loro assegnazione agli attuatori. La specifica di nuove modalità operative avviene mediante un'istruzione dedicata gestita dalla produzione numero 15 della Tabella 5.

15. L'azione associata a questa produzione effettua la creazione di una nuova modalità operativa per gli attuatori configurata secondo i parametri specificati nell'istruzione ed aggiunge tale oggetto software all'attuatore specificato.
    In questo caso, pur rimanendo invariato lo scope degli ambienti, si ha una visibilità globale per quanto riguarda gli attuatori nell'ambiente corrente in maniera tale da poter aggiornare un attuatore in qualsiasi punto della configurazione purché si rimanga all'interno dello scope che afferisce all'ambiente interessato. Questa restrizione è stata aggiunta per evitare ambiguità o errori da parte dell'operatore.
    Se l'attuatore con l'identificativo specificato nell'istruzione viene trovato nell'ambiente corrente allora a questo viene aggiunta la modalità operativa appena creata.

I parametri richiesti sono un identificativo, un valore nominale dell'impatto per unità di tempo che la modalità in oggetto produce, il valore di consumo energetico e l'identificativo della proprietà fisica influenzata dall'effetto che l'attuatore produce.

4.5.5 Validazione

Una grammatica formale, per essere valida, deve superare determinate verifiche volte a determinare principalmente se è realizzabile, funzionante ed adatta all'analisi sintattica in modo da poter essere tradotta mediante strumenti automatici. A seguire sono elencati i risultati delle verifiche alle quali è stata sottoposta la grammatica in fase di progettazione.

Left recursion
La grammatica del simulatore di Ambient Intelligence risulta essere *non left-recursive*.
Una grammatica si considera tale se presenta una produzione che abbia il primo elemento, ovvero quello più a sinistra, uguale all'head della produzione stessa. In caso di riduzione della produzione, il corpo chiamerebbe nuovamente la stessa produzione ricorsivamente dato che il simbolo di lookahead cambia solamente quando avviene il match di un simbolo terminale, entrando di conseguenza in un loop infinito.

Ambiguità
La grammatica non è ambigua.
Una grammatica risulta ambigua quando la struttura sintattica di una frase non è unica. Tale ambiguità non permette di eseguire l'analisi sintattica e la traduzione in modo deterministico risultando quindi non adatta all'utilizzo da parte di un'applicazione informatica. Inoltre l'ambiguità sintattica genera di conseguenza un'ambiguità semantica.

Cicli
La struttura della grammatica formale in oggetto non presenta cicli.
Una grammatica presenta un ciclo se da una produzione si può passare ad un'altra produzione la quale può ritornare alla chiamante. Questo porterebbe ad una condizione di loop infinito e quindi alla non risoluzione delle produzioni specificate.

Simboli non usati
Nessuno dei simboli terminali e non terminali definiti nella grammatica rimane inutilizzato.

Produzioni non ridotte
Tutte le produzioni che definiscono la grammatica del simulatore di Ambient Intelligence vengono ridotte.

4.5.6 Implementazione

Il processo di creazione del lexer e del parser mediante i rispettivi software di generazione garantiscono una struttura ottimizzata dei due strumenti software impiegati nel simulatore di Ambient Intelligence per il riconoscimento del linguaggio di programmazione. I risultati di tali processi sono i seguenti:

- la generazione del lexer viene effettuata dal Lexer Generator JFlex, il quale per la grammatica che descrive il linguaggio di configurazione del simulatore di Ambient Intelligence costruisce un NFA (Nondeterministic Finite Automaton) avente 308 stati il che viene trasformato in un DFA (Deterministic Finite Automaton) di 149 stati che diventano 122 dopo il processo di minimizzazione[5];
- la generazione del parser effettuata dal Parser Generator CUP non ha rilevato alcun errore presente nella grammatica specificata le cui regole hanno generato 49 stati di parsing unici. Il parser generator ha inoltre effettuato delle verifiche non riscontrando conflitti e non sono stati rilevati simboli terminali e non terminali inutilizzati o produzioni che non vengano mai ridotte.

4.6 Regole di utilizzo

L'intero linguaggio è definito dai simboli e dalle regole di produzione esposte nelle precedenti tabelle, ma per illustrarne le regole di utilizzo in maniera generale si affronta l'analisi di un esempio di simulazione configurata mediante il linguaggio.
La grammatica del linguaggio di configurazione non impone vincoli restrittivi ma, come ogni linguaggio, necessita di alcune regole per la corretta comprensione. Tali regole generali sono riportate nella seguente Tabella 6.

1. La specifica dei parametri di simulazione deve avvenire prima di creare ed aggiungere altri elementi in quanto rappresenta la radice dell'albero di inclusione (si veda la Fig. 1, gerarchia logica di inclusione degli elementi di simulazione).
2. Alla creazione di un ambiente tutte le istruzioni successive si riferiranno a tale ambiente in accordo con le regole definite. Lo scope dell'ambiente cambierà nel momento in cui si dichiara un nuovo ambiente e tutte le successive istruzioni afferiranno a quest'ultimo.

5. Minimizzazione di un DFA: procedimento che trasforma un dato automa a stati finiti deterministico nel DFA equivalente che riconosce lo stesso linguaggio formale ma avente il minimo numero di stati.

3. La fine della configurazione è specificata dal costrutto “end”.
4. Due istruzioni che specificano lo stesso parametro sono considerate come un aggiornamento dei valori, pertanto verrà utilizzato solamente l'ultimo valore specificato in ordine temporale nel flusso di configurazione della simulazione.
5. Le istruzioni devono terminare con il carattere ′;′ preceduto da uno spazio.

Tabella 6: Regole generali sulla scrittura della configurazione di una simulazione

L’esempio utilizzato per l’esposizione configura una simulazione multi-ambiente così da poter coinvolgere la maggior parte degli elementi di simulazione implementati nel simulatore di Ambient Intelligence.

	#AmiSimConfig
	#Simulazione di esempio
	#configurazione simulazione
1	simulation sim_esempio ;
2	simulation from 06-02-2020 10:00 to 07-02-2020 12:00 ;
3	simulation step 10 minutes ;
	#primo ambiente
4	ambient first_room temperature_C 21 ;
5	ambient humidity internal 30 ;
6	ambient humidity internal 22 ;
7	sensor S_101 temperature_C ;
	#secondo ambiente
8	ambient second_room temperature_C 12 ;
9	sensor S_201 temperature_C ;
10	actuator A_202 ;
11	actuator A_202 AM_COOL -0.71 0.57 temperature_C ;
12	ambient temperature_C external 11 ;
13	end

Tabella 7: Configurazione di una simulazione mediante il linguaggio formale

Nella specifica dell'elenco delle istruzioni non è necessario applicare una determinata indentazione in quanto eventuali spazi in esubero sono ignorati dalla grammatica implementata, l'unico vincolo al quale porre attenzione durante la scrittura è quello di scrivere ogni istruzione in una riga andando a capo ogni volta che si inserisce un punto e virgola.
Le righe il cui primo simbolo è il carattere cancelletto '#' identificano l'inizio di una riga di commento per l'operatore e saranno ignorate nel processo di parsing del linguaggio in quanto riconosciute dal lexer e non incluse nella fase di elaborazione successiva.
A questa regola vi è un'eccezione: per effettuare una verifica di validità del file al fine di stabilire se si tratta di un file di configurazione del simulatore di Ambient Intelligence viene effettuato il controllo esclusivamente della prima riga presente nel file la quale in una configurazione valida ci si aspetta che coincida con la stringa "#AmiSimConfig". Tale stringa viene usata come marcatore dei file di configurazione del simulatore di Ambient Intelligence a fini di filtraggio ed evitare che file non validi arrivino al lexer il quale solleverebbe svariati errori non potendo riconoscere la grammatica del linguaggio.
Questa tecnica permette di riconoscere immediatamente anche file di notevole dimensione con contenuto arbitrario che magari possono essere selezionati per sbaglio in quanto ne viene letto solamente l'inizio, contrariamente a quanto accadrebbe se invece fosse fornito direttamente al lexer il quale leggerebbe tutto il contenuto nella speranza di identificare lessemi validi del linguaggio. In realtà anche il lexer è stato progettato con un meccanismo di sicurezza che ferma il processo di analisi non appena viene rilevato un lessema non riconosciuto da nessuna delle espressioni regolari.

La seconda riga del file di configurazione rappresenta un nome assegnato alla configurazione. Questo valore di comodo non viene utilizzato dal simulatore e viene anch'esso ignorato dal lexer che non lo passa alla fase di analisi successiva. Da questo punto iniziano le istruzioni di configurazione della simulazione, in particolare la prima istruzione assegna il nome specificato, che nell'esempio è "sim_esempio".
A seguire, come si vede nell'istruzione numero 2 della Tabella 7, imposta la durata della simulazione indicando la data e l'ora di inizio nel formato "DD-MM-YYYY HH:MM" preceduta dal lessema "*from*" che denota la data di inizio e seguita dal lessema "*to*" il quale denota la data di fine simulazione, scritta con lo stesso formato della precedente.
Nel progettare il linguaggio si è scelto di garantire la leggibilità da parte dell'operatore, facendo in modo di dare alle istruzioni un formato che pur mantenendo la formalità e rigidità espressiva garantisse anche una struttura grammaticale simile al linguaggio naturale. Nello scegliere una lingua che fosse compatibile con tali obbiettivi, la scelta migliore è ricaduta sulla lingua inglese la quale oltre ad essere di uso e diffusione internazionale favorisce la compattezza delle istruzioni. Si veda ad esempio l'istruzione appena spiegata per l'assegnamento del tempo di simulazione.

L’istruzione numero 3 specifica la dimensione del passo di simulazione che deve essere usato durante lo svolgimento della stessa. Nell'esempio viene impostato un passo pari a 10 minuti. Questo comporta che tra lo stato della simulazione al punto n e lo stato al punto $n+1$ intercorrono esattamente 10 minuti di tempo simulato. Gli algoritmi di simulazione effettuano quindi i calcoli con un "campionamento" ad intervalli di 10 minuti di tempo simulato e gli algoritmi di previsione, come ad esempio il calcolo dell'impatto che un attuatore ha sullo stato attuale dell'ambiente, effettuano stime per lo stesso intervallo di 10 minuti successivi allo stato corrente del passo $n-1$. Questa istruzione conclude la parte di configurazione dei parametri della simulazione, l'ordine delle istruzioni da questo momento in poi può essere arbitrario, deve solamente rispettare le regole esposte nella Tabella 6.
Il motivo che ha portato all’introduzione della regola numero 5 (Tabella 6), risiede nel fatto che il lexer suddivide i lessemi in input utilizzando gli spazi contenuti nel testo, pertanto non sono ammessi spazi all’interno dei nomi degli elementi specificati in quanto verrebbero interpretati come due stringhe di testo separate che costituiscono due distinti lessemi del linguaggio, riconosciuti correttamente ma che non troverebbero nessun riscontro nelle produzioni della grammatica formale del linguaggio. L’introduzione della possibilità di non separare il carattere di fine istruzione con uno spazio richiederebbe l’analisi semantica, la fase più complessa dell’analisi di un linguaggio la quale è stata evitata nel presente progetto.

L'esempio della Tabella 7 continua con l’istruzione numero 4 la quale crea il primo ambiente della simulazione. Per la specifica di un nuovo ambiente sono necessari tre argomenti: il nome identificativo dell'ambiente, l'identificativo di una delle proprietà fisiche implementate nel simulatore di Ambient Intelligence ed il valore iniziale di tale proprietà. La proprietà fisica specificata costituirà la grandezza di base presente nell'ambiente in quanto un ambiente senza alcuna grandezza fisica non può esistere nella simulazione.
Le istruzioni successive alla numero 4 possono aggiungere un numero arbitrario di elementi e grandezze all'ambiente senza limiti. Nell'esempio viene creato l'ambiente con identificativo “first_room” al quale viene assegnata la grandezza fisica della temperatura con un valore iniziale pari a 21 gradi Celsius. Segue l'aggiunta all'ambiente di una seconda proprietà fisica, ovvero l’umidità, mediante l’istruzione numero 5. Le proprietà fisiche aggiunte all'ambiente dopo la sua creazione permettono di specificarne la tipologia, cioè se si tratta di una proprietà interna oppure esterna all'ambiente. Gli algoritmi di simulazione attualmente implementati nel simulatore di Ambient Intelligence non effettuano calcoli sulle proprietà esterne agli ambienti ma questi dati rimangono a disposizione per eventuali moduli che integrano una logica capace di sfruttarli.
Sfruttando la regola numero 4 (Tabella 6) riguardante l’aggiornamento dei valori degli elementi in caso di istruzioni afferenti allo stesso oggetto, l’istruzione numero 6 effettua proprio questo specificando nuovamente l’aggiunta della grandezza umidità in cui cambia solamente il valore

assegnato, 30 la prima volta e 22 la seconda. Tale condizione viene interpretata dal parser del linguaggio di configurazione come una richiesta di aggiornamento del dato, pertanto in fase di simulazione la grandezza umidità dell'ambiente “first_room” avrà un valore iniziale pari a 22.
Completa la configurazione del primo ambiente l’istruzione numero 7, che aggiunge un sensore di temperatura. L'elemento viene specificato mediante la relativa parola chiave “sensor” seguita dall'identificativo ed infine dalla proprietà fisica che monitorerà. Anche in questo caso la proprietà fisica viene specificata mediante la sua stringa identificativa.
L’istruzione 8 crea un secondo ambiente il quale acquisisce lo scope corrente e da adesso in poi le istruzioni afferiranno a questo con identificativo “second_room”. Viene assegnata la grandezza temperatura con valore iniziale di 12 gradi, un sensore di temperatura ed un attuatore.
L’istruzione 10 da sola non sarebbe in grado di produrre un attuatore funzionante in quanto oltre a specificare la creazione dell’elemento all’interno della simulazione ed assegnargli un identificativo, per funzionare l’elemento necessita della specifica delle modalità operative che lo caratterizzano e dei relativi parametri che ne modellano il comportamento. Per lo scopo è stata introdotta una istruzione dedicata che permette di aggiungere un numero arbitrario di modalità operative agli attuatori specificati. Questa istruzione è la numero 11 nell’esempio della Tabella 7 e determina l’elemento al quale aggiungere tali dati mediante l’identificativo ed a seguire richiede i seguenti parametri di configurazione:

- il codice identificativo della modalità operativa dell'attuatore;
- il valore di influenza dell'attuatore sull'ambiente per unità di tempo;
- il valore di consumo energetico della specifica modalità operativa;
- l'identificativo della proprietà fisica influenzata dall'azione dell'attuatore quando è attiva la modalità in oggetto.

Completa la configurazione del secondo ambiente l'istruzione numero 12 che aggiunge una grandezza esterna di temperatura con valore iniziale di 11 gradi Celsius.
Infine, il lessema “end” indica la conclusione dell’elenco di istruzioni di configurazione e permette di procedere alla fase successiva della procedura di parsing del linguaggio.

4.7 Istruzioni del linguaggio

Le istruzioni di configurazione che possono essere utilizzate nel linguaggio per la creazione della simulazione sono di seguito elencate e descritte. Al fine di identificare la tipologia delle variabili che rappresentano gli argomenti di tali istruzioni sono stati introdotti dei termini, riportati nella Tabella 8, i quali identificano una specifica tipologia di dato.

Termine	Tipologia di variabile
TEXT	stringa di testo.
INT	numero intero.
DATE	stringa di data e ora in formato DD-MM-YYYY HH:MM
SDNUM	numero decimale con segno opzionale.
PHPROP	identificativo di una proprietà fisica.
PTYPE	tipologia di proprietà fisica (interna o esterna all'ambiente).

Tabella 8: Identificatori dei tipi di dati utilizzati nelle istruzioni del linguaggio di configurazione

Nella Tabella 9 è riportato l'ordine di specifica delle istruzioni. Benché non vi siano rigide restrizioni sull'ordine in cui possono essere espresse le singole istruzioni bisogna comunque rispettare le regole descritte nella sezione 4.6.

Istruzione	Ordine	Descrizione
#Commenti	-	Righe di commento, ignorate dal lexer.
#AmiSimConfig	0	Identificativo di un file di configurazione.
#Nome_configurazione	0	Opzionale.
simulation TEXT ;	1	Assegna un nome alla simulazione.
simulation from DATE to DATE ;	1	Specifica l'intervallo temporale di simulazione.
simulation step INT minutes ;	1	Specifica il valore del passo di simulazione.
ambient TEXT PHPROP SDNUM ;	N	Crea un ambiente assegnandogli il nome TEXT ed avente la proprietà fisica di base PHPROP con il relativo valore iniziale SDNUM.
ambient PHPROP PTYPE SDNUM ;	N	aggiunge la proprietà PHPROP all'ambiente corrente la quale può essere interna od esterna (PTYPE) e possiede il valore iniziale indicato in SDNUM.
sensor TEXT PHPROP ;	N	aggiunge un sensore con l'identificativo indicato in TEXT il quale monitora la proprietà PHPROP dell'ambiente corrente.

actuator TEXT ;	N	aggiunge all'ambiente corrente un attuatore con l'identificativo specificato in TEXT.
actuator TEXT TEXT SDNUM SDNUM PHPROP ;	N	aggiunge una modalità operativa all'attuatore specificato. Gli argomenti indicano in ordine: l'ID dell'attuatore da aggiornare, il nome della modalità da aggiungere, il parametro di influenza sull'ambiente, il parametro di consumo energetico della modalità, la proprietà influenzata dalla modalità operativa.
end	(max N) +1	lessema di termine configurazione.

Tabella 9: Ordine di specifica istruzioni

Capitolo 5
Modello dell'utente virtuale

La simulazione di un utente virtuale costituisce una parte di notevole importanza in molti casi di studio ed in particolare negli scenari dove si analizzano gli effetti della presenza di un utente nell'ambiente pervasivo.
Emulare le azioni svolte da una persona è un compito molto difficile al quale diversi ambiti scientifici si dedicano. Nel presente progetto si è fatto ricorso a tecniche di intelligenza artificiale per generare dati sintetici mediante un modello comportamentale appreso dall'osservazione delle abitudini di vita quotidiana di una persona.
L'obiettivo è quello di generare dei set di lunghezza arbitraria di dati delle attività di vita quotidiana che svolgerebbe una persona i quali serviranno al simulatore di Ambient Intelligence per rendere dinamiche le azioni svolte dall'utente virtuale durante il processo di simulazione. Questo fattore inserisce, se richiesto, una componente di imprevedibilità all'interno della simulazione la quale va ad influenzare buona parte degli aspetti. Le azioni di un utente infatti variano lo stato degli ambienti coinvolti ed inoltre, in base all'evoluzione delle caratteristiche dell'ambiente durante la simulazione, l'utente virtuale è anche capace di intervenire direttamente sugli elementi presenti al fine di regolare le condizioni ambientali per soddisfare le proprie e variabili preferenze.
Il vantaggio principale derivato dall'introduzione di tale modulo nel simulatore di Ambient Intelligence consiste nella possibilità di effettuare simulazioni più realistiche e dinamiche in autonomia generando un intervallo temporale di dati di lunghezza arbitraria e con un arbitrario campionamento del tempo tale da rendere il software autonomo ed indipendente dalla necessità di disporre di un dataset. Tale indipendenza non pregiudica comunque la possibilità da parte dell'operatore di fornire il set di dati con le azioni che l'utente virtuale deve svolgere ed il simulatore di Ambient Intelligence offre anche la possibilità di fornire nuovi algoritmi comportamentali per l'utente mediante l'implementazione di moduli aggiuntivi.
La soluzione che ha permesso di raggiungere gli obiettivi prefissati ha coinvolto l'impiego di una rete neurale la quale è stata addestrata con un dataset di dati raccolti dal monitoraggio di una persona reale all'interno della propria abitazione domestica nell'arco di svariati mesi in modo da poter generare un modello realistico basato sulle azioni svolte quotidianamente.

5.1 Dati generati

I dati forniti dal modulo di intelligenza artificiale hanno come obiettivo quello di generare un comportamento dell'utente simulato che sia quanto più realistico possibile, così da poter effettuare delle simulazioni in cui poter valutare i risultati influenzati delle azioni di un utente pseudo realistico.
La posizione dell'utente nell'ambiente viene determinata dall'attività in corso di svolgimento in un determinato istante temporale della simulazione. Questa scelta è stata adottata in quanto la quasi totalità delle attività sono legate in maniera univoca ad un determinato ambiente: ad esempio, se l'utente sta preparando il pranzo si troverà in cucina, se sta facendo una doccia si troverà in bagno, etc. Solamente poche attività non sono legate ad un ambiente specifico: la lettura di un libro, ad esempio, può essere effettuata in camera da letto così come in soggiorno o anche in cucina. In fase di progetto si è scartata la possibilità di aggiungere questi ulteriori parametri ai dati elaborati dalla rete neurale per due motivi. Il primo è quello appena esposto della relazione tra attività ed ambiente in cui essa viene svolta in quanto l'informazione sulla posizione è nella trasportata dall'azione stessa. Il secondo motivo è rappresentato dal fatto che l'eventuale introduzione nel flusso dei dati della rete neurale avrebbe influenzato il comportamento andando potenzialmente a costituire un fattore di disturbo nella corretta determinazione delle varie attività svolte dall'utente.
Nei pochi casi particolari in cui si presenta un'attività non univocamente legata ad un ambiente di svolgimento si risolve facilmente andando a disambiguare il nome della stessa; nel caso dell'esempio della lettura di un libro si possono specificare le attività "lettura a letto" o "lettura sulla scrivania".
Questi dati relativi alle azioni che devono essere svolte all'interno del processo di simulazione da parte dell'utente virtuale devono essere elaborati in un momento temporale che precede l'inizio dei calcoli perché tali dati devono essere generati secondo determinati criteri e soprattutto devono risultare pronti per l'utilizzo fin dal primo passo di elaborazione. Nella fase di configurazione della simulazione l'operatore va a specificare la durata temporale della stessa in tempo simulato, e questo parametro risulta determinante per il modulo di intelligenza artificiale, in quanto va a determinare l'intervallo dei dati da produrre.
Nella fase di inizializzazione delle strutture software, il simulatore di Ambient Intelligence va ad interrogare il set di dati generato dal modulo di intelligenza artificiale per ottenere le attività dell'utente. Il set di dati viene sottoposto ad un processo di parsing per verificarne la correttezza della sintassi ed i dati contenuti vengono poi memorizzati in un'apposita struttura all'interno dell'oggetto software che modella l'utente. In questo modo risulta più agile fornire agli elementi della simulazione l'attività corrente in un qualsiasi momento temporale.

5.2 Workflow

Il flusso di lavoro per il funzionamento del modulo progettato si compone di due fasi. La prima, preparatoria e preliminare all'utilizzo del software è volta alla creazione del modello comportamentale a partire dal dataset di attività di vita quotidiana selezionato per lo scopo. L'altra costituisce l'utilizzo a regime del modulo dove il software prevede di impiegare il modello appreso nella fase di addestramento per la generazione dei dati richiesti dal simulatore di Ambient Intelligence, il che costituisce lo scopo principale della progettazione del presente lavoro.

La fase di preparazione del modulo di intelligenza artificiale si compone di tre elaborazioni distinte. La prima è costituita dalla preparazione dei dati di input che saranno utilizzati per addestrare la rete neurale. Dopo la lettura ed interpretazione si procede alla rimozione dei dati che non sono di interesse per il problema trattato o che rappresentano del rumore nel dataset ai fini dell'addestramento. Conclude questa fase di preparazione dei dati un algoritmo di adattamento che verifica e reimposta il formato in modo efficace per l'algoritmo di addestramento della rete neurale senza modificarne il contenuto ma solamente la forma.

La successiva è la fase cruciale che determinerà la qualità dei dati generati in quanto si effettua qui l'addestramento della rete neurale in cui viene costruito un modello derivato dall'analisi dei dati forniti. Solitamente una piccola parte di questi dati viene riservata alla successiva fase di validazione in cui si analizza la qualità dei risultati prodotti dalla rete neurale con il modello appreso mediante l'algoritmo di addestramento selezionato per lo scopo. Completata anche questa procedura la rete neurale risulta pronta per la generazione dei dati e quindi per essere impiegata nel processo di simulazione svolto dal simulatore di Ambient Intelligence.

La procedura appena esposta è stata pensata per essere effettuata solamente al fine della creazione del modulo di intelligenza artificiale, tuttavia risulta possibile ripeterla tutte le volte che si desidera cambiare il modello comportamentale della rete neurale sostituendo il dataset o l'algoritmo di addestramento impiegato.

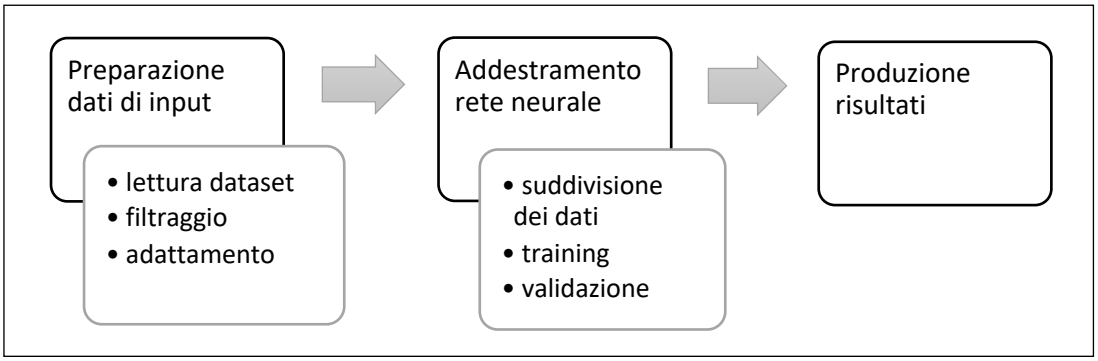

Figura 7: Processo di realizzazione del modulo di intelligenza artificiale

Quando il modulo risulta operativo, nel suo funzionamento a regime svolge il proprio compito effettuando tre fasi di elaborazione, schematizzate in Figura 8. La prima è presente solo se i dati forniti risultano in un formato diverso da quello progettato e necessitano quindi di una conversione, mentre le altre due sono la generazione dei dati e la relativa interpretazione per poterli fornire nel formato che l'interfaccia software si aspetta.

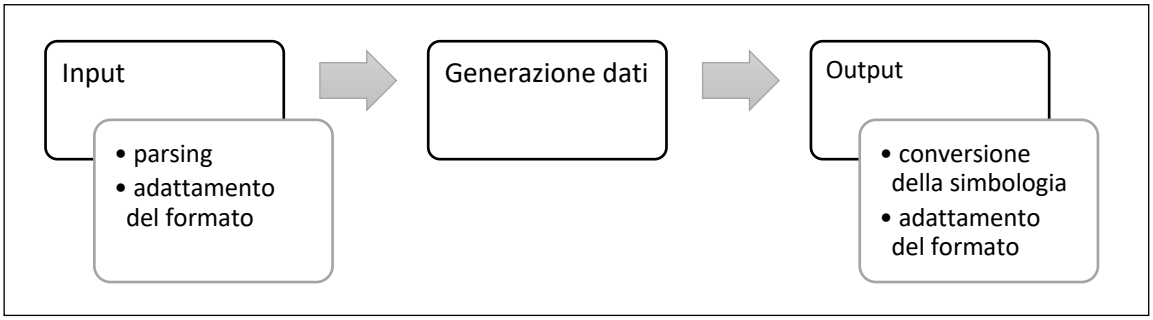

Figura 8: Fasi di elaborazione per la generazione dei dati

5.3 Strumenti utilizzati

Lo sviluppo del modulo di intelligenza artificiale è stato effettuato impiegando due diverse tecnologie. La rete neurale e tutte le operazioni che la riguardano sono programmate ed eseguite in Matlab[6] in quanto l'ambiente di sviluppo fornisce tutte le funzionalità necessarie alla realizzazione ed addestramento utilizzando le implementazioni allo stato dell'arte degli algoritmi.
Essendo il simulatore di Ambient Intelligence sviluppato con il linguaggio di programmazione Java, si è scelto di sviluppare il modulo logico con questo linguaggio per garantire una integrazione nativa la quale può all'occorrenza essere ampliata o modificata senza alcun vincolo.
Questo è stato possibile principalmente perché la fase di generazione dei dati e quella dell'effettivo utilizzo sono state progettate in maniera separata e completamente indipendente tra loro, in accordo con la modularità che caratterizza la struttura del simulatore di Ambient Intelligence. Ciò favorisce la sostituzione delle componenti con la massima facilità ed è infatti possibile sostituire la rete neurale con un'altra oppure un modulo differente che effettui un'altra tipologia di calcoli senza apportare alcuna modifica al software. Un tale modulo può addirittura fare a meno della componente di intelligenza artificiale purché gli vengano forniti i dati da utilizzare.

6. Ambiente per il calcolo scientifico sviluppato da MathWorks Inc. http://mathworks.com

Funzionamento runtime

Le modalità di utilizzo del modulo sono due e dipendono dalla modalità di esecuzione del codice Matlab. La prima lavora in runtime ed esegue la rete neurale nell'ambiente Matlab nativo che quindi deve essere in esecuzione localmente sulla macchina in cui è eseguito il simulatore. Questa modalità è la più dinamica possibile in quanto permette in tempo reale di modificare la rete neurale e generare nuovi dati, rendendo inoltre possibile intervenire nel processo di generazione ed inserire nuove routine che possano elaborare i dati.
Lo svantaggio di questo approccio è dovuto al requisito che l'operatore intento ad utilizzare il simulatore di Ambient Intelligence debba conoscere almeno in maniera basilare l'ambiente Matlab. Inoltre, è necessario possedere il software che deve essere in esecuzione per il funzionamento della simulazione.
Essendo questi aspetti di notevole impatto sull'usabilità del software, si è deciso di riservare questa modalità di esecuzione ad una utenza esperta che abbia l'effettiva necessità di modificare il processo che regola il funzionamento del modulo software di intelligenza artificiale o che voglia utilizzare magari una rete neurale differente (in tipologia o configurazione) per la generazione dei dati. In entrambi i casi lo sviluppo di un modulo di comunicazione tra i due software risulta abbastanza semplice, purché venga mantenuto il formato dati accettato dall'interfaccia software di comunicazione del simulatore di Ambient Intelligence.

Funzionamento autonomo

La seconda modalità di funzionamento risulta completamente autonoma in quanto il codice di esecuzione della rete neurale risultante dall'addestramento effettuato in Matlab viene esportato in una libreria Java utilizzata nel modulo software dedicato.
Il vantaggio di questo approccio è quello di non richiedere l'esecuzione contemporanea sulla stessa macchina dell'ambiente Matlab rendendo indipendente il modulo di intelligenza artificiale.
Lo svantaggio di questo funzionamento risiede nel fatto che la rete neurale non è più modificabile ma solamente eseguibile. Se è necessario apportare modifiche si deve procedere ad una nuova esportazione del codice Matlab aggiornato andando quindi a sostituire la libreria e di conseguenza ricompilare il simulatore di Ambient Intelligence con la versione aggiornata. Questo scenario tuttavia non è il caso di utilizzo più comune e sicuramente risulta molto più comodo eseguire in maniera indipendente la simulazione.

Software di elaborazione dati

Al fine di automatizzare gran parte delle procedure appena descritte è stato sviluppato per il presente progetto un software in linguaggio Java. Anch'esso utilizza nel suo piccolo una struttura modulare che permette quindi di scegliere in maniera molto agevole quali elaborazioni applicare ai dati.

Si compone delle funzioni di lettura e scrittura dei file di testo in quanto la maggior parte dei dataset sono forniti come semplice testo formattato per facilitare tutti i contesti di utilizzo ed integra le funzioni per la conversione dei dati nel formato CSV (Comma Separated Values) utile per una veloce manipolazione dei dati con software di gestione più avanzati, se necessario, ed anche per l'importazione in Matlab. Gli altri moduli che lo compongono forniscono le funzionalità di elaborazione dati esposte nella sezione 5.4.3, vi è un parser per la verifica della sintassi aggiornato per include anche la funzionalità di conversione della codifica numerica assegnata alle attività, un filler di dati e il modulo di creazione del file di output nel formato CSV. In tutte le procedure e le elaborazioni effettuate sui dati vi è sempre un controllo che permette di segnalare eventuali errori di elaborazioni volto a prevenire l'uso di un eventuale dataset non consistente generato da un errore nei dati.

5.4 Dataset

5.4.1 Origine dei dati

Il dataset utilizzato per l'addestramento della rete neurale necessaria alla generazione dei dati di simulazione è fornito dalla Washington State University, in particolare dal suo Center for Advanced Studies in Adaptive Systems (CASAS)[7] il quale si occupa di progetti relativi ad ambienti intelligenti in cui lo stato degli abitanti viene rilevato mediante appositi sensori ed influenzato da diverse tipologie di controller con l'obiettivo di migliorare il comfort dell'ambiente.
I dati sono stati raccolti in occasione del progetto WSU CASAS Smart Home Project [19] in diversi periodi e mediante il monitoraggio di utenti reali i quali hanno svolto normalmente le loro attività di vita quotidiana nella propria abitazione equipaggiata per rilevare i dati necessari alla determinazione della posizione dell'utente nei vari ambienti dell'abitazione e da questi determinare l'attività in corso di svolgimento. Tutti i dati rilevati sono accompagnati da una marcatura temporale che determina univocamente il momento in cui il dato è stato raccolto.

Selezione dati di addestramento

Anche se i dataset messi a disposizione dal centro CASAS sono svariati, la base di partenza dei dati è sempre costituita dalle letture grezze effettuate dalle diverse tipologie di sensori e le attività svolte dagli utenti devono pertanto essere ricavate analizzando tali dati. Essendo questa una attività di ricerca a sé stante, ed anche di notevole spessore, la selezione del dataset da utilizzare per l'addestramento della rete neurale del presente progetto è stata effettuata valutando solamente i dataset annotati, ovvero quelli in cui è riportata anche l'indicazione

7. Center for Advanced Studies in Adaptive Systems: http://casas.wsu.edu

dell'attività dell'utente rilevata dalle analisi dei sensori. Un ulteriore parametro di selezione è stato quello di scartare i dataset rilevati in ambienti dove fossero presenti animali domestici i quali costituiscono elemento di disturbo nella rilevazione delle attività producendo letture dei sensori non appartenenti ad una attività svolta dalla persona all'interno dell'abitazione.

5.4.2 Contenuto

Il dataset selezionato ricopre un periodo temporale che va dal 2010 al 2011. I dati sono stati raccolti in un appartamento abitato da una sola persona, una donna, senza animali domestici. Le attività presenti nel dataset sono le usuali attività di vita giornaliera di una persona e sono state classificate nella Tabella 10, dove è anche riportato il numero di occorrenze con le quali si presentano nei dati forniti. L'ultima attività elencata, "resperate", si riferisce all'utilizzo di un dispositivo per il monitoraggio della pressione sanguigna.

Attività	Occorrenze nel dataset
Meal preparation	3210
Relax	5833
Leave/enter home	1721
Sleeping	802
Eating	514
Work	341
Bed to toilet	314
Wash dishes	142
Housekeeping	66
Resperate	12

Tabella 10: Attività presenti nel dataset

Basandosi sulle informazioni disponibili e sulle occorrenze delle attività si può presumere che la donna monitorata sia una pensionata. Le poche attività etichettate come lavoro probabilmente si riferiscono ad attività svolte nella stanza dell'appartamento indicata come ufficio.
L'appartamento in cui sono stati raccolti i dati è composto da diversi ambienti. L'ingresso apre su un ambiente comune dove è presente la sala da pranzo e la cucina, comunicanti con il retro della casa. La zona notte è costituita da un ambiente separato in cui sono presenti la stanza da letto, un bagno e la cabina armadio. Infine un corridoio separa le due zone da un ulteriore ambiente dove vi è una camera per gli ospiti e da una stanza etichettata come ufficio. Il corridoio termina con una porta che conduce al garage dell'abitazione.

Nelle posizioni più opportune per ogni ambiente sono collocati diversi sensori e ogni porta ha un sensore che ne rileva l'apertura e la chiusura. Inoltre in ogni ambiente è presente un sensore di temperatura e diversi sensori di movimento, ad esclusione dei due bagni per ovvi motivi di privacy.
La struttura dell'ambiente e la disposizione dei sensori è rappresentata nella pianta architettonica della seguente Figura 9, dove sono indicate anche le posizioni dei sensori di movimento (icona circolare di colore blu) e i sensori di temperatura (icona rettangolare di colore verde). Tutte le porte sono da ritenersi equipaggiate con un sensore di rilevamento delle azioni di apertura e chiusura.

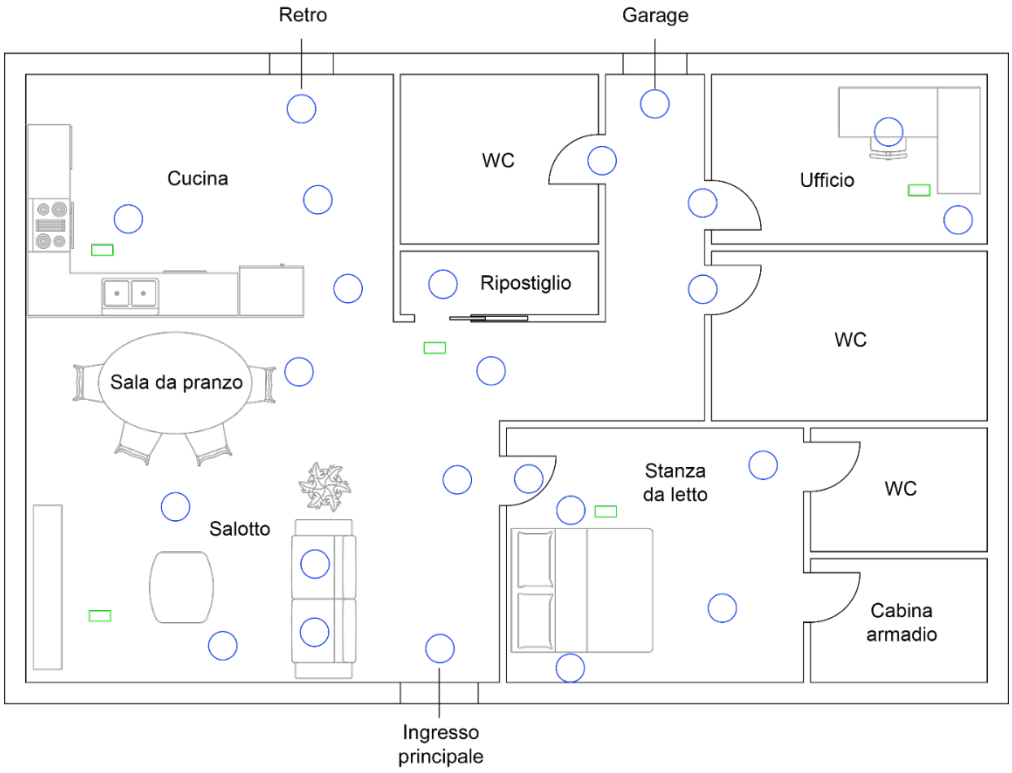

Figura 9: Pianta architettonica dell'ambiente monitorato

5.4.3 Preparazione dei dati

Il dataset, prima di poter essere utilizzato, deve necessariamente essere elaborato al fine di selezionare solamente i dati di interesse per i nostri scopi, eliminare dati superflui considerati non rilevanti ed infine verificare che i dati rimanenti abbiano la corretta struttura per essere utilizzati controllando che non vi siano errori o formati differenti.

Processo di filtraggio

Il dataset conta 1,72 milioni di entry e la prima operazione effettuata è volta alla rimozione delle letture grezze dei sensori in quanto non sono rilevanti e creerebbero disturbo nella fase di addestramento della rete neurale. Vengono quindi identificate innanzitutto le entry senza annotazioni le quali rappresentano proprio le letture grezze dei sensori e mediante l'applicazione di diverse espressioni regolari vengono localizzate e rimosse.
Le espressioni regolari tengono conto dei diversi sensori presenti con i rispettivi identificativi. Sono presenti tre tipologie: sensori di movimento, apertura/chiusura porta e temperatura, ognuno dei quali riporta una entry per l'evento di attivazione ed una per l'evento di disattivazione.
A terminare la fase di filtraggio dei dati vi è un processo di rimozione di eventuali spaziature doppie o non conformi con la formattazione generale dei dati mediante l'applicazione di elementari espressioni regolari.

Processo di adattamento

A seguire, nuovamente mediante l'applicazione di un'espressione regolare, vengono rimossi i microsecondi da ogni marcatura temporale in quanto superflui per gli scopi perché il simulatore di Ambient Intelligence usa una precisione che non scende al di sotto del secondo, pertanto non porta alcun vantaggio addestrare la rete neurale con una tale precisione e probabilmente sarebbe controproducente sulla qualità dei dati prodotti.
Il dataset netto a questo punto dell'elaborazione contiene solamente i dati di interesse correttamente formattati ed il numero di entry è diventato 12942.

Processo di parsing

Il dataset viene passato ad uno step di elaborazione in cui viene effettuato il parsing dei dati, ovvero le singole entry vengono lette per verificarne la correttezza del formato ed interpretate così che eventuali entry non conformi vengano scartate e non si verifichino errori nelle fasi successive.
Questo processo, inoltre, effettua una conversione dei nomi assegnati alle attività. Dopo aver valutato diversi approcci, si è constatato che la maniera più efficiente e soprattutto efficace per quanto riguarda l'addestramento a discriminare le diverse attività svolte dall'utente è quella di assegnare ad ognuna di esse un valore numerico appartenente ad una sequenza ordinata. Ad ogni attività viene quindi assegnato un numero intero univoco che la identifica, con l'accortezza di selezionare numeri interi consecutivi così che sia agevole determinare le attività risultanti dai dati prodotti dalla rete neurale in fase di simulazione.

Nella Tabella 11 sono riportate le assegnazioni effettuate per i dati trattati, elencando tutte le attività che risultano presenti almeno una volta nel dataset impiegato.

Attività	Codice assegnato
Sleeping	1
Bed_to_Toilet	2
Meal_Preparation	3
Relax	4
Housekeeping	5
Eating	6
Wash_Dishes	7
Leave_Home	8
Enter_Home	9
Work	10
Resperate	11

Tabella 11: Associazione numerica delle attività

Essendo delle stime probabilistiche, i dati prodotti avranno anche valori decimali i quali si andranno a collocare tra due interi consecutivi e sarà poi cura dell'algoritmo di interpretazione andare ad applicare il criterio più opportuno per la determinazione dell'attività. Solitamente si applica l'arrotondamento, considerando il valore intero più vicino.
I dati di output prodotti dalla rete neurale saranno quindi dei valori probabilistici che spaziano nell'intervallo numerico definito dalle attività presenti nel dataset di input. Tali valori numerici sono assegnati nella fase di lettura preliminare del dataset. Per semplicità nel presente progetto sono stati assegnati i numeri interi crescenti a partire da uno e seguendo l'ordine di occorrenza con cui si manifestano le diverse attività nella lettura ordinata del dataset.
La stessa funzione che stabilisce la corrispondenza tra la stringa contenente il nome testuale dell'attività ed il numero ad essa assegnata deve essere usata anche nel processo di presentazione dei dati prodotti dalla rete neurale così da poter riconvertire i valori numerici nel formato testuale facilmente comprensibile dagli esseri umani. Tale elaborazione non è necessaria ai fini funzionali del software ma risulta utile alla lettura e presentazione dei dati.

Processo di filling

Il dataset a questo punto sarebbe pronto per essere impiegato nell'addestramento della rete neurale, ma dopo aver effettuato diverse prove si è ritenuto di rilevo per la qualità dei dati finali prodotti introdurre una ulteriore fase di elaborazione.

Nella fase preliminare della progettazione si sono vagliati svariati approcci alla risoluzione del problema ed una sfida non indifferente era costituita dalla scelta della tipologia di dati con relativo formato che fornisse il miglior risultato possibile. Dopo aver effettuato diverse prove, l'analisi finale dei risultati ha fatto ricadere la scelta sull'approccio che impiega una rete neurale le cui caratteristiche verranno a breve descritte. La rete neurale ottiene un oggettivo miglioramento della precisione nei dati forniti in fase di simulazione se viene utilizzato un dataset "fitto" nella fase di addestramento.

Normalmente le entry che compongono il dataset riportano le informazioni in cui una determinata attività è stata rilevata e l'entry successiva si manifesta nel momento in cui una seconda diversa attività viene rilevata. Di conseguenza si ha un intervallo temporale che va a separare le due attività. Tale intervallo a livello logico indica che l'ultima attività rilevata è da considerarsi ancora in esecuzione finché non ne viene rilevata un'altra diversa e l'intervallo stesso funge da indicatore sulla durata della prima attività.

Il simulatore di Ambient Intelligence richiede al presente modulo di intelligenza artificiale l'attività stimata a intervalli regolari dettati dalla configurazione assegnata alla simulazione da eseguire. Nella fase di istruzione della simulazione e, quindi, di inizializzazione e popolazione delle strutture dati preliminare all'esecuzione, vengono calcolati tutti gli step temporali di simulazione basati sui parametri specificati e durante la creazione del comportamento da assegnare all'utente all'interno del processo di simulazione viene effettuato un numero di chiamate al presente modulo di intelligenza artificiale pari al numero di step di simulazione.

Quando la rete neurale si trova a rispondere ad una interrogazione, deve quindi soddisfare la richiesta e fornire l'azione che l'utente deve svolgere all'orario specificato senza la consapevolezza dei momenti precedenti o successivi l'istante temporale al quale si riferisce la richiesta. Dal punto di vista della rete neurale ogni interrogazione è indipendente dalle altre. Tale proprietà presenta sia dei lati positivi che dei lati negativi.

L'aspetto positivo risiede nel fatto che non c'è alcun vincolo nei dati di input, la rete neurale funziona sia che le si chieda l'attività dell'utente in un tempo spot sia che le si chiedano le attività di un intervallo temporale esteso e senza alcun vincolo sulla sua lunghezza. Proprio questo fattore ha determinato la scelta del presente approccio al problema nella fase di progettazione: in questo modo il simulatore di Ambient Intelligence può permettere all'operatore che lo utilizza la massima libertà di azione sulla simulazione, che non deve sottostare ad alcun vincolo temporale.

L'aspetto negativo risiede nel fatto che probabilmente una rete neurale che tenga conto degli stati passati dell'utente durante l'intervallo temporale di simulazione potrebbe fornire delle attività stimate migliori sotto qualche aspetto, ma per funzionare dovrebbe conoscere in ogni momento tutti gli stati della simulazione, cosa non praticabile nella condizione attuale in quanto i due processi di calcolo vengono effettuati in tempi differenti. Ciò non toglie che sarebbe possibile farlo senza alcun impedimento andando a sviluppare un modulo che intervenga nella

simulazione per stimare il comportamento dell'utente ad ogni singolo passo di calcolo. In questo scenario si avrebbe la possibilità di adattare tale comportamento all'evolversi della simulazione ma si dovrebbero valutare attentamente le prestazioni della rete, intese sia in termini di qualità dei dati prodotti sia delle prestazioni computazionali che tale rete avrebbe, in quanto si troverebbe a dover effettuare tutte le stime in realtime eseguendo un numero di interazioni pari al numero di step temporali di calcolo che compongono il processo di simulazione.

Quest'ultima fase di elaborazione del dataset consiste quindi nel colmare gli intervalli temporali che si presentano tra un'attività e la successiva, e ricoprono periodi temporali anche grandi in cui la rete neurale non ha alcun dato sul quale basarsi nella fase di addestramento. L'intervallo maggiore è senza dubbio costituito dai periodi notturni della giornata in cui ci si ritrova solamente una entry nel momento in cui viene rilevata l'attività di riposo della persona e la successiva attività si rileva svariate ore dopo, nella giornata successiva. Questi intervalli di vuoto vengono colmati andando a ripetere l'ultima attività rilevata, la quale si suppone sia ancora in esecuzione. L'operazione effettuata ha il solo scopo di specificare ciò che il dataset dà per sottinteso in maniera tale da farlo cogliere chiaramente alla rete neurale nella fase di addestramento.
Al processo descritto è stato assegnato il nome di "filling", ovvero riempimento, in quanto va a replicare il dato di attività dell'utente che si trova al limite inferiore di ogni intervallo temporale fino al raggiungimento del successivo intervallo determinato da una diversa attività svolta. Nessun dato viene alterato durante il processo. L'intervallo di campionamento scelto per questa operazione è quello del minuto, in quanto risulta lo stesso applicato dal simulatore di Ambient Intelligence ed è il minimo livello di precisione temporale al quale il processo di simulazione può arrivare. Questa è stata una scelta progettuale del simulatore di Ambient Intelligence in quanto il fatto di avere nei dati di simulazione una maggiore precisione non fornisce alcun beneficio o informazione aggiuntiva.

5.5 Rete neurale

5.5.1 Fondamenti teorici

Una *rete neurale* è una interconnessione di *unità logiche a soglia* (ULS) [21], che effettuano l'operazione di somma pesata degli ingressi ricevuti confrontandone il risultato con un valore di soglia, in modo da fornire in output un valore binario alto se il valore di soglia viene superato e basso in caso contrario.
Una ULS separa lo spazio dei vettori di ingresso che producono una risposta alta da quelli che producono una risposta bassa tramite un iperpiano, pertanto le funzioni booleane implementabili sono funzioni linearmente separabili.

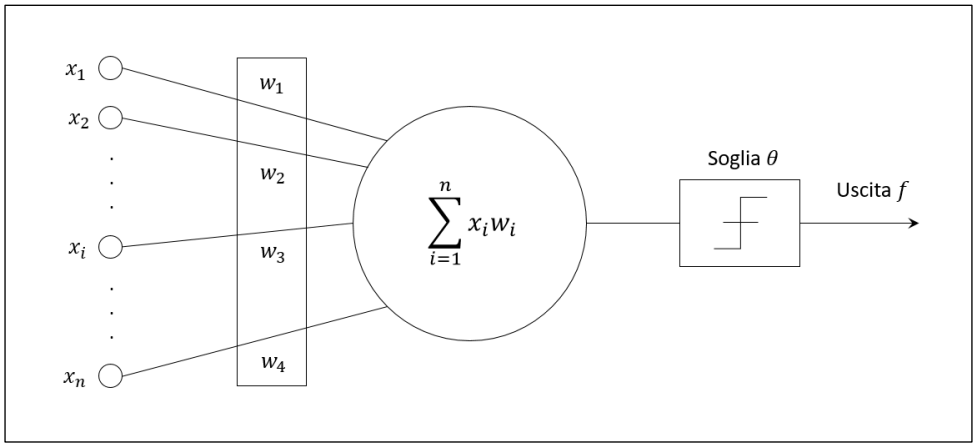

Figura 10: Rappresentazione di una ULS

$$f = \begin{cases} 1 \ \ se \sum_{i=1}^{n} x_i \cdot w_i \geq \theta \\ 0 \ \ altrimenti \end{cases}$$

In applicazioni più complesse, dove vi sono più di due sole azioni possibili, una ULS non è sufficiente per poter affrontare il problema e si deve ricorrere ad una rete di ULS la quale prende il nome di *rete neurale* in quanto le ULS sono considerate modelli semplificati dei neuroni biologici che si attivano o meno in base all'intensità e all'attivazione delle loro connessioni sinaptiche.
La funzione implementata da una rete neurale dipende sia dalla topologia che la caratterizza sia dai pesi assegnati alle singole ULS.

La tipologia di rete utilizzata in questo progetto è una rete con connessioni in avanti *feed foward*. Questo tipo di rete neurale non presenta cicli: ciò significa che nessun ingresso ad una ULS dipende dalla sua uscita, neanche attraverso ulteriori ULS intermedie. La rete impiegata è

inoltre multistrato, ovvero le ULS in essa contenute sono organizzate a strati i cui elementi ricevono ingressi solamente dallo stato immediatamente precedente.
L'ordine degli strati che compongono una rete neurale è sempre strutturato da almeno tre strati canonici. Il primo strato è chiamato di *input* e riceve i dati forniti in ingresso alla rete, mentre l'ultimo strato è chiamato di *output* e fornisce in uscita i dati prodotti nella codifica utilizzata dalla rete neurale per la loro trattazione. Collocati tra i due strati di input e di output vi sono uno o più strati intermedi i quali costituiscono il modello appreso dalla rete e prendono il nome di *strati nascosti* (hidden layers) in quanto non hanno un contatto diretto con l'ingresso o l'uscita della rete neurale. Ogni strato nascosto è composto da ULS e pertanto deve essere dimensionato; tale operazione non è di semplice risoluzione e verrà trattata nel dettaglio in seguito.

5.5.2 Retropropagazione e Resilient backpropagation

Il metodo più usato per l'addestramento supervisionato di una rete neurale multistrato con connessioni in avanti è quello di impiegare l'algoritmo di retropropagazione.
L'idea alla base dell'apprendimento mediante l'algoritmo di retropropagazione è quella dell'applicazione della regola del concatenamento per il calcolo dell'influenza che ogni peso esercita sull'intera rete nel rispetto della funzione errore utilizzata. Viene effettuato il calcolo della derivata parziale per ogni peso e la minimizzazione della funzione errore viene raggiunta applicando il metodo della discesa lungo il gradiente.

La tipologia di addestramento usata è quella supervisionata la cui metodologia di apprendimento si basa dall'osservazione, pertanto maggiore sarà la quantità di dati osservati e migliori saranno le prestazioni ovvero la qualità dei dati prodotti [23].

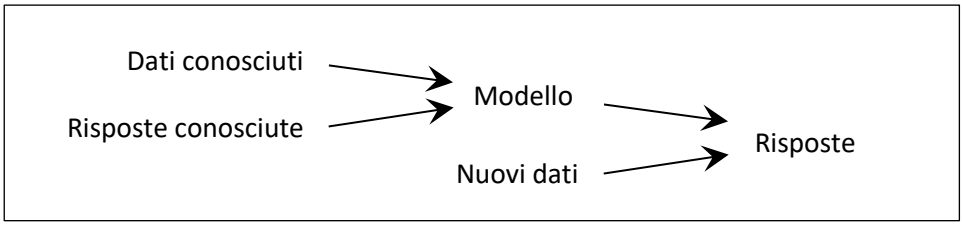

Figura 11: Flusso dei dati

Nella Figura 11 è rappresentato lo schema del flusso di dati nella fase di addestramento (la metà sinistra) e nella fase di simulazione (la metà destra) ovvero di esecuzione del modello appreso dalla rete nell'uso a regime alla quale essa è destinata.
Nella fase di addestramento vengono presentati i dati che la rete tratterà e le risposte corrette conosciute, in maniera tale che l'algoritmo di addestramento costruisca un modello matematico

che caratterizzerà la rete neurale. Nella seconda fase alla rete neurale viene presentato un set di dati alla quale essa applicherà il modello appreso per generare le relative risposte risultanti.

Durante la procedura risulta opportuno sostituire le funzioni di soglia delle ULS con delle funzioni sigmoide le quali possono essere riportate alla forma originale una volta completato l'addestramento della rete neurale.
Ogni strato della rete fornirà un'uscita la quale viene modellata come un vettore di componenti in analogia all'ingresso che lo strato riceve. Il vettore degli ingressi del primo strato è denotato con $X^{(0)}$, lo strato j-esimo della rete avrà quindi come uscita il vettore $X^{(j)}$ il quale costituisce il vettore degli ingressi dello strato (j+1)-esimo. L'uscita della rete viene infine indicata con f. Ogni sigmoide di ogni strato ha un vettore di pesi applicato sui suoi ingressi, denotato con $W_i^{(j)}$ per la sigmoide i-esima dello strato j-esimo.

L'attivazione, ovvero l'ingresso di una unità a sigmoide risulta data da:

$$s_i^{(j)} = X^{(j-1)} \cdot W_i^{(j)}$$

Algoritmo di retropropagazione

Il metodo della retropropagazione effettua il calcolo del gradiente della funzione di errore quadratico:

$$\varepsilon = (d - f)^2$$

Il vettore dei pesi rispetto al quale viene calcolato il gradiente dovrebbe contenere tutti i pesi della rete; risulta però più opportuno calcolare le derivate parziali di ε rispetto ai pesi in gruppi corrispondenti ai vettori dei pesi delle singole sigmoidi.

La derivata parziale di ε rispetto al vettore dei pesi $W_i^{(j)}$ è:

$$\frac{\partial \varepsilon}{\partial W_i^{(j)}} \stackrel{\text{def}}{=} \left[\frac{\partial \varepsilon}{\partial w_{1i}^{(j)}}, \dots, \frac{\partial \varepsilon}{\partial w_{li}^{(j)}}, \dots, \frac{\partial \varepsilon}{\partial w_{m_{j-1}+1,i}^{(j)}} \right]$$

dove m_{j-1} è il numero di unità a sigmoide presente nello strato (j-1)-esimo e $w_{li}^{(j)}$ è la componente l-esima del vettore $W_i^{(j)}$.

Applicando la regola del concatenamento si ottiene:

$$\frac{\partial \varepsilon}{\partial W_i^{(j)}} = \frac{\partial \varepsilon}{\partial s_i^{(j)}} \frac{\partial s_i^{(j)}}{\partial W_i^{(j)}}$$

effettuando le sostituzioni si arriva alla forma:

$$\frac{\partial \varepsilon}{\partial W_i^{(j)}} = -2(d - f)\frac{\partial f}{\partial s_i^{(j)}} X^{(j-1)}$$

La quantità $(d - f)\frac{\partial f}{\partial s_i^{(j)}}$ viene indicata con $\delta_i^{(j)}$ e indica quanto l'errore quadratico dell'uscita della rete sia sensibile a cambiamenti nell'ingresso alla corrispondente funzione sigmoide.

Il gradiente di ε può essere scritto usando i δ:

$$\frac{\partial \varepsilon}{\partial W_i^{(j)}} = -2\delta_i^{(j)} X^{(j-1)}$$

I vettori dei pesi vengono modificati secondo la direzione del gradiente negativo, quindi la regola per la modifica dei pesi di tutta la rete è la seguente:

$$W_i^{(j)} \leftarrow W_i^{(j)} + c_i^{(j)} \delta_i^{(j)} X^{(j-1)}$$

dove $c_i^{(j)}$ è la costante che determina la velocità di apprendimento per il vettore dei pesi, la quale solitamente è uguale per tutti i vettori dei pesi della rete.

Scelta dell'algoritmo

La selezione dell'algoritmo di addestramento della rete neurale non è un'operazione facile in quanto un determinato algoritmo può essere ottimo per la risoluzione di una tipologia di problemi ma non per un'altra. Le prestazioni, intese principalmente come qualità del modello appreso più che prestazioni computazionali della fase di addestramento, dipendono da svariati fattori compresa la complessità del problema, la quantità di dati presenti nel dataset utilizzato, la struttura della rete neurale impiegata e diversi altri.

La valutazione degli algoritmi che ha portato alla scelta di quello da impiegare nel presente progetto ha preso in considerazione la pubblicazione di uno studio effettuato dal dipartimento di Ingegneria Civile[8] della Erciyes University[9], dove la divisione di idraulica ha affrontato il problema dello studio di stato e comportamento del suolo legato al rischio idraulico [24].

8. Erciyes University Engineering Faculty: insaat.erciyes.edu.tr

9. Erciyes University: en.erciyes.edu.tr

Il suddetto studio, necessario per la pianificazione delle operazioni da parte del Sistema di Gestione Acque, è volto ad effettuare delle previsioni sullo stato di stress del suolo in base ai dati raccolti durante diversi anni di monitoraggio.
La tipologia di problema risulta molto simile al problema affrontato nel presente progetto relativo alle attività di un utente, visto che entrambi sono basati su dati storici per generare delle stime di dati temporalmente successivi a quelli presenti nel dataset di addestramento del modello della rete neurale.

In [24] il problema viene risolto con successo impiegando una rete neurale (Artificial Neural Network) della stessa tipologia della rete strutturata per il presente progetto, ovvero una rete neurale con connessioni in avanti (feed foward) multi-strato addestrata quindi con un algoritmo di retropropagazione. Il team di ricerca ha effettuato il confronto di tre algoritmi di retropropagazione (backpropagation): Levenberg-Marquardt (LM); Conjugate gradient (CGF); Resilient backpropagation (RB).

Nel confronto vengono valutate la velocità di convergenza in addestramento e le prestazioni nella fase di testing. I risultati finali sono riportati nella Tabella 12:

Table 7—Training results of each algorithm

Algorithm used for training	CPU time (s)	Number of epochs	MARE (kg/cm^2)	MSE (kg^2/cm^4)
LM	6	50	2.29	0.0033
CGF	28	554	2.98	0.0035
RB	35	2000	2.94	0.0037

Table 8—Testing results of each algorithm

Algorithm	Nodes in hidden layer	MARE (kg/cm^2)	MSE (kg^2/cm^4)
LM	7	4.13	0.0123
CGF	13	4.27	0.0120
RB	7	4.13	0.0118

Tabella 12: Risultati del confronto fra i tre algoritmi di retropropagazione [24]

Le conclusioni ricavate mostrano innanzitutto che la risoluzione della tipologia di problema trattata può essere efficacemente raggiunta con l'impiego di una rete neurale multistrato con connessioni in avanti e che risultano validi tutti e tre gli algoritmi di addestramento coinvolti nel confronto.
I risultati prestazionali evidenziano come l'algoritmo di *Levenberg-Marquardt* sia il più veloce richiedendo solamente una piccola frazione di tempo rispetto agli altri due algoritmi presi in considerazione per addestrare la rete neurale nella fase di training. Invece nella fase di testing

del modello appreso dalla rete neurale si hanno risultati migliori con l'algoritmo *Resilient backpropagation* il quale ha il minor valore MSE.

Resilient backpropagation

L'algoritmo *Resilient Backpropagation* (identificato con l'acronimo RPROP) propone una variante del metodo originale di retropropagazione andando ad effettuare un adattamento del valore di aggiornamento dei pesi in relazione al comportamento della funzione errore, in particolare andando a considerarne la topologia locale [25].
L'algoritmo agisce secondo le seguenti condizioni:

$$\Delta w_{ij} = \begin{cases} -\Delta_0\,, & se\ \dfrac{\partial \varepsilon}{\partial w_{ij}} > 0 \\ +\Delta_0\,, & se\ \dfrac{\partial \varepsilon}{\partial w_{ij}} < 0 \\ 0\ \ , altrimenti \end{cases}$$

dove Δ_0 è il *valore di aggiornamento* (una costante) e dipende dal problema.
Questo metodo di modifica dei pesi risulta abbastanza elementare e proprio a causa della sua semplicità non funziona in modo soddisfacente se applicato a problemi complessi dove risulta arduo trovare una soluzione accettabile.
L'idea alla base dei miglioramenti apportati dall'algoritmo RPROP è quella di ottenere delle informazioni sulla topologia della funzione errore così che i valori di aggiornamento dei pesi possano essere modificati in maniera appropriata. Per ogni peso viene introdotto il suo specifico valore di aggiornamento Δ_{ij} il quale si evolve durante la procedura di apprendimento in relazione al suo segno locale della funzione errore ε.
Si ottiene così una seconda regola di apprendimento proprio per i valori di apprendimento stessi:

$$\Delta_{ij}(t) = \begin{cases} \Delta_{ij}(t-1) * \eta^{+,} & se\ \dfrac{\partial \varepsilon}{\partial w_{ij}}(t-1) * \dfrac{\partial \varepsilon}{\partial w_{ij}}(t) > 0 \\ \Delta_{ij}(t-1) * \eta^{-}, & se\ \dfrac{\partial \varepsilon}{\partial w_{ij}}(t-1) * \dfrac{\partial \varepsilon}{\partial w_{ij}}(t) < 0 \\ \Delta_{ij}(t-1), & altrimenti \end{cases}$$

con $0 < \eta^- < 1 < \eta^+$.

Il valore di aggiornamento dei pesi viene influenzato solamente dal comportamento del segno delle due derivate. Ogni volta che la derivata parziale del corrispondente peso w_{ij} cambia di segno, il che indica che l'ultimo valore di aggiornamento è stato troppo alto e di conseguenza l'algoritmo ha superato un minimo locale (Fig. 13), il valore di aggiornamento $\Delta_{ij}(t)$ viene decrementato di un fattore η^-. Se la derivata mantiene il proprio segno allora il valore di

aggiornamento dei pesi è leggermente incrementato al fine di accelerare la convergenza nelle regioni superficiali (Fig. 12).

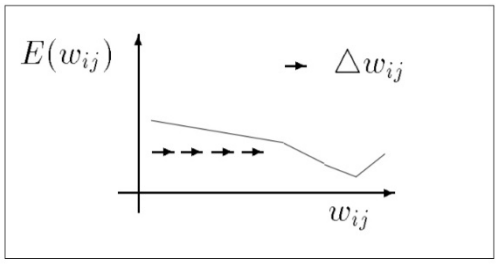

Figura 12: Tempo di convergenza

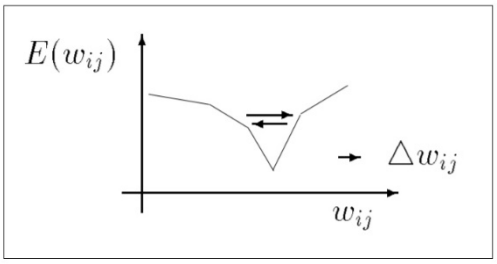

Figura 13: Minimo locale

La regola di aggiornamento dei pesi è la stessa esposta nella prima equazione con una sola eccezione: se la derivata parziale cambia di segno, il precedente step di aggiornamento ha superato un minimo, viene quindi ripristinato:

$$\Delta w_{ij}(t) = -\Delta w_{ij}(t-1), \quad se \frac{\partial \varepsilon}{\partial w_{ij}}(t) * \frac{\partial \varepsilon}{\partial w_{ij}}(t-1) < 0$$

Quando si verifica un cambiamento di segno, il processo di adattamento viene quindi "riavviato" il che significa che negli step successivi non viene effettuato nessun adattamento dei valori di aggiornamento.
I valori di aggiornamento ed i pesi cambiano ogni volta che l'intero pattern set viene presentato alla rete neurale, ovvero in ogni epoca di addestramento.

Conclusioni

L'algoritmo RPROP risulta quindi di facile implementazione e possiede uno schema di apprendimento semplice il quale riduce drasticamente il numero di epoche richieste rispetto all'algoritmo di retropropagazione originale mentre lo sforzo computazionale è leggermente incrementato rispetto agli altri algoritmi di addestramento.
Avendo mostrato ottimi valori in termini di convergenza e robustezza, l'algoritmo Resilient Backpropagation sembra essere una buona soluzione per l'addestramento della rete neurale nel presente progetto al fine di assolvere il compito della generazione delle attività di vita quotidiana degli utenti simulati all'interno del simulatore di Ambient Intelligence.

Implementazione

L'algoritmo risulta disponibile tra le funzioni fornite dai toolbox di Matlab, e pertanto si è scelto di utilizzare tale implementazione ritenuta sicuramente la più efficiente per il suddetto ambiente di sviluppo software.

La funzione che lo implementa si chiama *trainrp*[10] e viene specificata come parametro della rete neurale utilizzato poi successivamente dalla funzione *train* la quale effettua la procedura di addestramento.

5.5.3 Procedura di addestramento

In questa sezione viene descritta la procedura utilizzata per la creazione e l'addestramento della rete neurale. Tutte le relative operazioni vengono eseguite nell'ambiente di sviluppo di Matlab utilizzando le funzioni che questo fornisce per l'elaborazione dei dati e gli algoritmi necessari alla trattazione della rete neurale.
Completato il processo sarà possibile valutare le prestazioni della rete con i dati riservati specificatamente a tale scopo in maniera da poter decidere se mantenere lo stato della rete, procedere ad una nuova fase di addestramento oppure modificare la struttura della rete neurale ed i relativi parametri.

Preparazione dati

La prima operazione effettuata è l'importazione dei dati selezionati per l'addestramento dal dataset. Vengono create due variabili le quali modellano un vettore riga della dimensione pari alla lunghezza del dataset. Nel primo vengono inseriti i dati di input costituiti solamente dalle marcature temporali delle entry, mentre il secondo vettore viene invece popolato con i dati target costituiti dal valore delle attività svolte in corrispondenza dell'istante temporale nel vettore di input.
I dati di output del software di elaborazione del dataset sono organizzati in due colonne, una con i tempi e l'altra con gli identificatori numerici dell'attività. Questo formato risulta notevolmente più leggibile e modificabile da un essere umano rispetto all'accodamento classico delle entry previsto dal formato CSV.

Sia che l'importazione dei dati avvenga mediante una delle funzioni fornite da Matlab sia che si immettano mediante una semplice procedura di copia, per organizzarli in formato di vettore riga potrebbe essere necessario effettuare delle operazioni di trasposizione che risultano molto efficienti in ambiente Matlab nonostante la grande mole di dati trattati.
La procedura di addestramento viene effettuata utilizzando solo una parte del dataset il quale viene suddiviso come di seguito indicato:

- 70% dedicato al training;
- 15% per la validazione;
- 15% per il testing.

10. Funzione *trainrp* di Matlab: mathworks.com/help/deeplearning/ref/trainrp.html

Creazione della rete

Si procede alla creazione della rete neurale mediante l'utilizzo della funzione *fitnet*[11] (function fitting neural network) la quale genera una rete neurale orientata alla risoluzione di problemi di data fitting, che si collocano nella stessa categoria di appartenenza dei problemi di curve fitting. La funzione richiede come argomento obbligatorio il numero di stati nascosti e la loro dimensione, ovvero il numero di neuroni in essi contenuti.

Dimensionamento hidden layer

La quantità di ULS presenti nello strato nascosto incide in maniera decisiva sulla capacità di apprendimento e classificazione delle reti neurali: un numero troppo basso di unità causa una perdita di informazioni rispetto a quelle fornite dallo spazio degli elementi di input ed un numero di ULS superiore a quello necessario non apporta contributo ai dati di uscita.
Tipicamente si utilizza un numero di ULS che sia in grado di catturare una percentuale di varianza del dataset di input che sia compresa tra 70% e 90% [27].
Nel dataset impiegato per l'addestramento sono presenti 11 attività diverse e si è scelto di coprire il 90% di tale valore il che corrisponde ad un numero di 10 unità.

Addestramento

L'algoritmo utilizzato per la procedura di training è l'algoritmo *Resilient Backpropagation* e la valutazione delle performance del modello appreso viene calcolata mediante l'errore quadratico medio (Mean Squared Error).
La procedura di addestramento termina all'epoca numero 478 la quale risulta essere quella con il più basso valore dell'errore di validazione. L'addestramento continua comunque per ulteriori 6 iterazioni prima di arrestare la procedura.
Una iterazione viene chiamata epoca ed è definita come una singola presentazione di tutti i vettori di input alla rete neurale, la quale si aggiorna di conseguenza ai risultati di tutte le presentazioni.

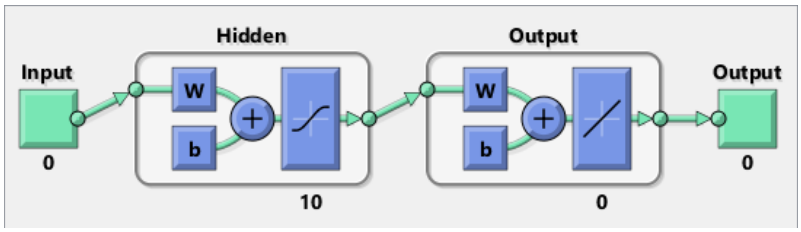

Figura 14: Struttura della rete neurale

11. Funzione *fitnet* di Matlab: mathworks.com/help/deeplearning/ref/fitnet.html

5.6 Validazione dei risultati

In questa sezione verrà affrontata l'analisi dei risultati prodotti dalla rete neurale appena addestrata al fine di valutare la qualità del modello appreso. L'analisi è stata eseguita mediante l'applicazione di una metrica analitica così da poter valutare in maniera oggettiva la qualità dei dati valutando la similitudine dei risultati generati dalla rete neurale con il dataset di attività di vita quotidiana ottenute dal monitoraggio di una persona reale.

5.6.1 Valutazione analitica

Verranno adesso calcolati gli indici di valutazione analitica, e quindi oggettiva, dei dati prodotti dalla rete neurale posti in comparazione con i dati originali del dataset impiegato nell'addestramento della stessa.
La comparazione dei dati è effettuata tra l'insieme costituito dal dataset completo e l'insieme dei risultati generati dalla rete neurale avente la stessa lunghezza temporale in termini di giornate calcolate. In questo modo è possibile valutare in maniera completa le prestazioni della rete ed il pattern comportamentale che la caratterizza nelle varie fasi del giorno coinvolgendo tutte le tipologie di dati trattati.

Uniformazione dei set di dati

Entrambi i set di dati riportano le attività in orari che non possono coincidere, pertanto si è effettuata l'elaborazione di *filling* mediante il software di elaborazione che si occupa di preparare il dataset per l'utilizzo nella fase di addestramento. L'algoritmo, descritto nella sezione 5.4.3, va a replicare l'attività riportata nei dati minuto per minuto fino alla successiva in maniera tale da uniformare i due set di dati e rimuovere tutti gli intervalli vuoti.
In questo modo ogni minuto di una giornata di un set trova un valore corrispettivo nell'altro set di dati ed oltre a permetterne il confronto questa elaborazione facilita la valutazione della durata delle singole attività in quanto adesso basta semplicemente contare le occorrenze che corrispondono ai minuti di svolgimento esatti nell'intervallo preso in considerazione. La lunghezza dei due set di dati adesso risulta uguale e sono costituiti da 285055 entry che corrispondono a 198 giornate, circa 7 mesi.

Metrica di valutazione

La tipologia di problema non si presta bene alla valutazione mediante i classici metodi statistici in quanto le attività di vita giornaliera di una persona sono regolate da un modello matematico alquanto complesso e per la natura dei dati che le rappresentano non vi è uno specifico schema di ripetizione ma sono caratterizzati da un pattern comportamentale il quale presenta alcuni elementi fondamentali ripetibili ma è contraddistinto da un importante grado di variazione tra i

cicli giornalieri. In questo caso una ripetizione delle attività in maniera molto simile non sarebbe un buon risultato.
In letteratura non ci sono molte trattazioni che affrontano l'argomento, situazione che si ribalta se si riesce tramite delle elaborazioni conservative dei dati a trasformare i risultati in modo da potersi portare nel campo di applicazione dei metodi di analisi statistica. Tale approccio però sarebbe esposto potenzialmente al rischio di influenzare involontariamente i dati se non si procede con una rigorosa dimostrazione matematica del funzionamento della procedura, cosa che va oltre gli scopi del presente progetto. Dopo aver approfondito svariati studi in diversi settori affini che potrebbero fornire un supporto allo scopo, si è trovata una metrica adatta ed applicabile al presente lavoro senza effettuare alcuna trasformazione dei dati.
L'idea chiave è quella di valutare il grado di similitudine che presentano i dati generati dalla rete neurale rispetto a quelli del dataset impiegato per l'addestramento della stessa, aprendo le porte a diversi approcci di valutazione. Le principali tecniche di analisi statistica possono adesso essere applicate ai due insiemi di dati che si vogliono confrontare, diversi approcci in tal senso sono presentati in [29].
Tuttavia, invece di intraprendere questa strada si è scelto di applicare la metodologia proposta da un altro studio [31] che si è rivelato essere particolarmente adatto alla valutazione della similitudine tra i due insiemi di dati e che tratta la stessa tipologia di dati impiegati nel presente progetto in tutte le sue parti, dall'addestramento della rete neurale ai risultati prodotti.

Time Serie

La tipologia di dati appena citata è la *Time Serie*, ovvero una sequenza ordinata di valori disposti secondo un ordine temporale. Ciò corrisponde alla definizione di segnale a tempo discreto [32], il quale è costituito da una successione di valori, chiamati campioni, definiti su un insieme discreto di valori del tempo. Nel presente progetto i campioni sono uniformemente distanziati nel tempo ad intervalli di un minuto.
Nelle elaborazioni che hanno preceduto l'analisi, si è normalizzato l'intervallo temporale che intercorre tra due letture nel dataset di origine affinché rispecchiasse lo stesso intervallo dei dati generati dalla rete neurale, pertanto entrambi i set risultano essere già pronti per essere trattati come segnali numerici.

Metrica selezionata

La metrica di valutazione analitica selezionata per il presente progetto è stata proposta in uno studio sull'efficienza della ricerca in similarità di sequenze contenute in un database [31] effettuato dal Centro di Ricerca *IBM Almaden* (IBM Almaden Research Center).
Lo studio propone un metodo di indicizzazione per elaborare le query di similarità di sequenze temporali memorizzate in un database. La valutazione della similarità tra due sequenze viene calcolata estraendone le caratteristiche mediante la Trasformata Discreta di Fourier, così da

mappare le sequenze nel dominio delle frequenze dove bastano le prime armoniche per caratterizzare un segnale. Una importante osservazione è costituita dall'impiego del teorema di Parseval, il quale garantisce che la distanza Euclidea viene preservata sia nel dominio del tempo che nel dominio della frequenza.
Lo studio continua poi con la costruzione di un albero di ricerca contenente solamente i primi coefficienti di Fourier (in particolare fa riferimento al primo coefficiente il quale costituisce l'armonica fondamentale della sequenza) per ogni segnale in maniera tale da poter effettuare una ricerca per similarità efficiente ma questa parte non è di interesse per lo scopo del presente progetto di intelligenza artificiale.

Teoria dei Segnali

Trasformata di Fourier discreta

La Trasformata di Fourier è definita per sequenze periodiche, ma è comunque possibile calcolare lo sviluppo di una sequenza di durata finita impiegando la trasformata di Fourier discreta (DFT, Discrete Fourier Transform) [32].
Si consideri la sequenza $x(n)$ di lunghezza finita N tale che $x(n) = 0$ eccetto che nell'intervallo $0 \leq n \leq (N-1)$.
La trasformata di Fourier discreta è definita [32] come:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \qquad per\ 0 \leq k \leq N-1$$

dove $W_N = e^{-j(2\pi/N)}$

Fast Fourier Transform

Il calcolo diretto della Trasformata di Fourier discreta $X(k)$ impiegando la definizione appena riportata richiede $4N$ moltiplicazioni e $(4N-2)$ addizioni per ogni valore di k e poiché occorre valutare $X(k)$ per N diversi valori di k, il calcolo richiede in totale $4N^2$ moltiplicazioni e $N(4N-2)$ addizioni rendendo il costo computazionale di tale operazione esponenziale ($\approx N^2$).
Fortunatamente vi sono diversi algoritmi efficienti per il calcolo della Trasformata di Fourier che effettuano la Trasformata veloce di Fourier (FFT, Fast Fourier Transform).
Il principio fondamentale su cui si basano questi algoritmi è la scomposizione del calcolo della trasformata di Fourier discreta di una sequenza lunga N in trasformate di Fourier discrete di dimensioni sempre più piccole. Il modo in cui questo principio è applicato dà luogo alle diverse tipologie di algoritmi, tutti caratterizzati da un miglioramento della velocità di calcolo pressoché equiparabile [32].

La complessità computazione degli algoritmi per il calcolo della Trasformata veloce di Fourier è proporzionale ad $NlogN$ il che costituisce un enorme guadagno prestazionale rispetto al calcolo diretto della trasformata.

Teorema di Parseval

La tecnica proposta in [31] sfrutta il Teorema di Parseval, il quale asserisce che l'energia di una sequenza nel dominio del tempo risulta uguale all'energia nel dominio delle frequenze.

Sia X la DFT della sequenza x, si ha che:

$$\sum_{t=0}^{n-1} |x|^2 = \sum_{f=0}^{n-1} |X|^2$$

La Trasformata Discreta di Fourier eredita dalla Trasformata di Fourier nel tempo continuo le proprietà, in particolare la proprietà dell'additività, dell'omogeneità e del ritardo.

Siano X_f e Y_f rispettivamente le DFT delle sequenze x_t e y_t:

$$[x_t] \Leftrightarrow [X_f] \quad \text{e} \qquad [y_t] \Leftrightarrow [Y_f]$$

Additività:

$$[x_t + y_t] \Leftrightarrow [X_f + Y_f]$$

Omogeneità:

$$[ax_t] \Leftrightarrow [aX_f]$$

Ritardo:

$$[x_{t-t_0}] \Leftrightarrow [X_f e^{j\omega t_0/n}]$$

dove la pulsazione $\omega = 2\pi f$.

Quest'ultima evidenzia come un ritardo nel dominio del tempo modifica solamente la fase e lascia inalterato il modulo.
Le proprietà appena esposte implicano che la Trasformata Discreta di Fourier è una trasformazione lineare ed il teorema di Parseval asserisce quanto segue:

$$\|\vec{x} - \vec{y}\|^2 \equiv \left\|\vec{X} - \vec{Y}\right\|^2$$

Ciò implica che la distanza euclidea tra due sequenze $\vec{x}$ e $\vec{y}$ nel dominio del tempo è la stessa distanza euclidea nel dominio della frequenza.

Distanza Euclidea

Il calcolo della distanza tra due sequenze viene effettuato usando la distanza euclidea, che è definita come segue:

$$D(\vec{x}, \vec{y}) = \sqrt{\sum_{t=0}^{n-1} |x_t - y_t|^2}$$

Soglia di tolleranza

Il valore della distanza euclidea fornisce solamente un'indicazione riguardo il grado di similarità tra le due sequenze coinvolte nel calcolo. Serve pertanto un indice di paragone, rappresentato dalla *soglia di tolleranza*.

Si tratta di un valore di riferimento che stabilisce un limite numerico entro il quale le due sequenze possono essere considerate simili ed oltre il quale sono sicuramente differenti.

Il dimensionamento della soglia di tolleranza viene effettuato in [31] seguendo la relazione $\sqrt{1000 \cdot n}$ dove n è la lunghezza della sequenza.

Procedura di calcolo

I passi di calcolo per poter valutare i due set di dati mediante l'applicazione della metrica analitica esposta sono i seguenti:

1. importazione dei due set di dati;
2. calcolo della Trasformata Discreta di Fourier mediante l'algoritmo FFT;
3. calcolo della distanza Euclidea tra le due FFT trovate;
4. calcolo del valore di soglia per la similarità;
5. confronto del valore di distanza Euclidea trovato con la soglia di similarità. Le due sequenze di input risultano simili se l'indice di similarità ottenuto mediante il calcolo della distanza Euclidea delle DFT risulta inferiore al valore della soglia di tolleranza.

5.6.2 Calcolo della metrica di valutazione

Tutti i calcoli per la valutazione dei risultati sono eseguiti in ambiente MATLAB, sfruttando l'implementazione fornita per l'algoritmo necessario al calcolo della Trasformata Veloce di Fourier.

Il valore della distanza Euclidea tra le due FFT è pari a $3{,}6925e^{-10}$.

La soglia di similarità si calcola con $\sqrt{1000 * 285055} = 1{,}1668e^{4}$.

Si ha quindi che $3{,}6925e^{-10} < 1{,}1668e^{4}$, pertanto i due set di dati, cioè il dataset reale e l'insieme dei risultati generati dalla rete neurale, risultano simili secondo la metrica analitica usata. Questa valutazione conferma la bontà dei risultati ottenuti e del lavoro svolto per la realizzazione del progetto.

5.6.3 Valutazione della durata delle attività

In questa parte conclusiva del capitolo si effettua l'analisi di similarità tra il dataset dei dati reali ed i dati generati dalla rete neurale con l'attenzione rivolta alla durata delle attività svolte dall'utente. Un buon indicatore relativo all'apprendimento del modello da parte della rete neurale è dato dalla similitudine, entro certi criteri, dei pattern che caratterizzano i due insiemi di dati.

Dopo aver condotto l'analisi analitica dei risultati si effettua adesso la valutazione di similarità dell'andamento statistico tra i dati generati ed il dataset reale. A tal fine si sono sfruttate le proprietà che contraddistinguono la tipologia di dati trattati e cioè quelle relative ad una sequenza temporale. In ogni giornata dei set vengono svolte le attività di vita quotidiana, ognuna di queste attività è caratterizzata da una durata e da una specifica collocazione temporale nella sequenza.

La procedura per riassumere ed evidenziare il comportamento di tutte le giornate dei due insiemi di dati è quella di calcolare la media statistica della durata di ogni attività all'interno delle giornate, effettuando la media di entrambi i set di dati si ottengono due sequenze temporali che rappresentano una il comportamento medio del set di dati reali e l'altra quello del set di dati sintetici generati dalla rete neurale.

Si è effettuato il calcolo della durata di ogni attività in accordo con i timestamp riportati nei dati e dopodiché è stato possibile effettuare il conteggio dei minuti totali delle attività i cui risultati sono riportati nella seguente Tabella 13. Le etichette relative alle attività sono le stesse usate in precedenza e riportate nella Tabella 11. Nei risultati seguenti, oltre al conteggio dei tempi di svolgimento di ogni attività, è calcolata su questo dato anche la media giornaliera la quale non ha un ruolo diretto nel confronto dei risultati ma risulta un ausilio all'interpretazione.

Attività	Minuti totali	Media giornata
1	66060	344
2	915	5
3	26311	137
4	128828	671
5	1031	5
6	7341	38
7	1562	8
8	46048	240
9	6986	36
10	5509	29
11	119	1

Tabella 13: Media della durata delle attività presenti nel dataset

Attività	Minuti totali	Media giornata
1	56070	292
2	20100	105
3	15620	81
4	70600	368
5	56440	294
6	63220	329
7	1420	7
8	1280	7
9	1080	5
10	800	4
11	710	4

Tabella 14: Media della durata delle attività generate dalla rete neurale

Disponendo adesso delle due sequenze di dati, si può procedere al confronto. Applicando la stessa metrica analitica usata in precedenza si ottiene un valore della distanza euclidea delle DFT delle due sequenze inferiore alla soglia di tolleranza e pertanto risultano simili. In questo caso di sequenze brevi risulta più adatto alla valutazione rappresentare i risultati in un grafico che ne esalti l'andamento nel tempo così da poter cogliere il pattern che caratterizza ciascun set di dati.

Il grafico più adatto è sicuramente quello a linee in pila perché viene impiegato in statistica per portare all'attenzione il confronto del comportamento tra serie di dati ed è esattamente lo scopo di questa valutazione.

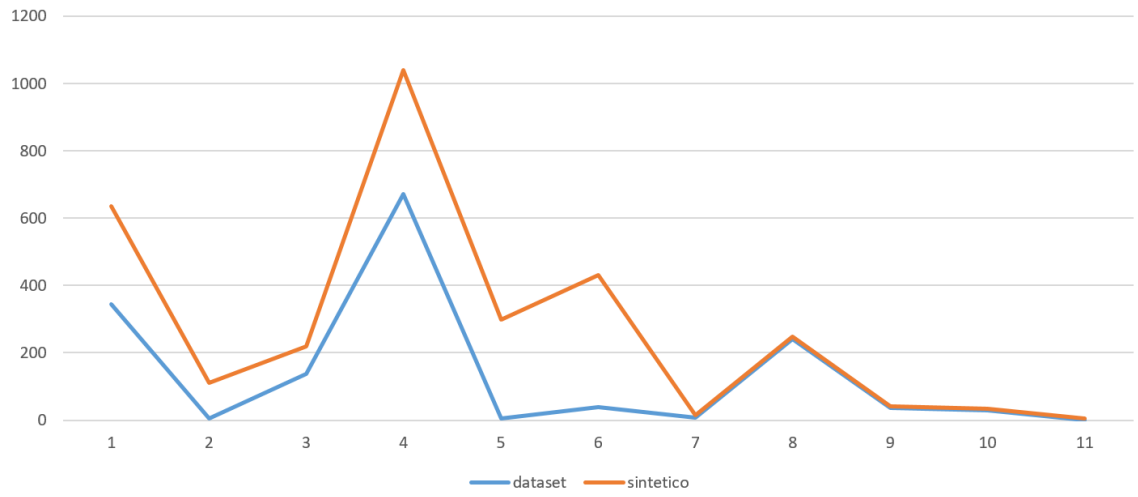

Figura 15: Grafico a linee sovrapposte delle sequenze di media temporale delle attività

Le conclusioni che si possono trarre dal confronto effettuato sono che i due insiemi di dati, quelli reali e quelli generati dalla rete neurale su un vasto numero di iterazioni, rispettano lo stesso pattern comportamentale e si classificano pertanto come simili. Vi sono alcune differenze, evidenziate dalla prima parte delle attività sintetiche che, nonostante presentino un andamento simile ai dati reali, sono diverse come durata di esecuzione totale. Tra queste vi sono le attività di dormire, preparare la colazione e di relax. La parte centrale del grafico presenta delle marcate differenze comportamentali, che coinvolgono la durata delle attività di svolgimento delle faccende domestiche e di mangiare. Infine l'ultima parte del grafico mostra un andamento particolarmente simile tra le due sequenze aventi valori diversi in quanto la caratteristica di questa tipologia di grafico è quella di rappresentare le linee sulla stessa scala ma traslate lungo l'asse delle ordinate per effettuare la sovrapposizione e valutarne la similitudine. Le attività finali sono quelle relative al tempo trascorso fuori casa, all'attività di lavoro e di misurazione della pressione sanguigna le quali sono tutto sommato attività caratterizzate da una certa regolarità.

Capitolo 6
Risultati sperimentali

Il presente capitolo conclude la trattazione esponendo i risultati e le metodologie impiegate per la sperimentazione completa di tutto il lavoro svolto con lo scopo di dimostrare oltre all'effettivo funzionamento del progetto anche la qualità dei risultati ottenuti.
In [4] sono proposti gli obiettivi che la sperimentazione di un simulatore dovrebbe generalmente approfondire. Il principale aspetto chiave che riguarda la sperimentazione è quello di assicurare che il modello simulativo fornisca dei risultati accurati rispetto al modello dello scenario reale utilizzato.
Nei precedenti capitoli sono state già svolte delle sperimentazioni focalizzate su alcuni aspetti del simulatore, caratterizzati da una complessità tale da giustificare uno studio dedicato. Ci si riferisce alla parte riguardante il linguaggio di configurazione della simulazione affrontata nel capitolo 4 dove si è eseguita una simulazione completamente automatizzata mediante un esempio riportato nella Tabella 7 ed alla seconda sperimentazione svolta nel capitolo 5 focalizzata sull'analisi del comportamento dell'utente simulato regolato dalla rete neurale.
Adesso si affronta la sperimentazione del simulatore nella sua completezza coinvolgendo tutte le principali componenti del sistema. Lo scopo è quello di valutare il comportamento e l'interazione tra i vari aspetti analizzati in precedenza in un contesto di applicazione reale, impiegando i casi più comuni in modo da replicare l'utilizzo da parte dell'operatore finale e quindi analizzarne le prestazioni in termini di qualità dei risultati ed usabilità.
Questo processo viene affrontato con un approccio incrementale il quale inizia con una simulazione basilare che viene via via arricchita da nuovi componenti ed aspetti fino al raggiungimento della versione più completa analizzata poi nel dettaglio alla fine del capitolo.

OMISSIS

Conclusioni

Il presente lavoro di tesi ha proposto un simulatore di Ambient Intelligence modulare volto alla valutazione di scenari costituiti da ambienti pervasivi dove vengono osservate o modificate le condizioni ambientali con lo scopo di adattarle al livello di comfort degli occupanti. La valutazione delle soluzioni presenti in letteratura ha evidenziato come queste siano focalizzate su aspetti specifici del settore e pertanto non risultano adatte per adempiere agli scopi preposti.

La progettazione del simulatore di Ambient Intelligence è stata effettuata con l'obiettivo di realizzare una struttura software modulare e scalabile permettendo l'intercambiabilità dei componenti e l'adattabilità a qualsiasi scenario. Queste caratteristiche hanno anche permesso di realizzare un algoritmo di simulazione general purpose che si occupa di gestire le operazioni e che non intraprende azioni dirette sui componenti, in quanto tutte le elaborazioni e i calcoli vengono effettuati in maniera decentralizzata dai singoli moduli, ognuno per quanto riguarda le proprie competenze. Il simulatore è supportato da un linguaggio formale di configurazione sviluppato specificatamente per essere conciso, espressivo ed efficiente. Si rivela estremamente utile per l'utilizzo da riga di comando e per poter eseguire simulazioni salvate.

Le caratteristiche della struttura software non pongono limiti all'intervallo temporale della simulazione, al numero di ambienti virtuali o alla quantità di elementi contenuti. Nuovi elementi possono essere facilmente introdotti nel flusso di simulazione ed è anche prevista l'integrazione di sistemi autonomi che possono interagire con tutti gli elementi presenti; questa funzionalità è stata pensata principalmente per i sistemi di Ambient Intelligence.

Il simulatore dispone di un modulo di intelligenza artificiale per permettere di introdurre nel processo di simulazione un utente virtuale che si comporta in maniera pseudo realistica seguendo i pattern comportamentali di una persona reale, così da permettere di valutare l'impatto che le azioni dell'utente hanno sull'ambiente e sugli elementi contenuti. La determinazione delle azioni di vita quotidiana viene effettuata da una rete neurale creata per lo scopo ed addestrata sulla base di dati reali. Questa scelta ha permesso di avere nella simulazione attività sempre diverse per ordine e durata. Un ulteriore vantaggio fornito da questo modulo è quello di poter eseguire simulazioni senza vincoli temporali in quanto la rete neurale è in grado di generare set di dati comportamentali di lunghezza arbitraria, rendendo così indipendente il simulatore dai relativi dataset, i quali possono comunque essere usati.

I risultati sperimentali hanno confermato la bontà della simulazione ed il raggiungimento degli obbiettivi anche per quanto riguarda le funzionalità del software. I dati generati dalla rete neurale sono stati valutati mediante l'utilizzo di una metrica analitica che ha evidenziato un buon grado di similarità con i dati reali, indice del buon apprendimento del modello che caratterizza tale tipologia di dati. Le simulazioni eseguite hanno prodotto risultati coerenti e realistici, emulando con le dovute semplificazioni gli scenari del mondo reale.

Elenco delle figure

Elenco delle tabelle

Bibliografia

[1] P. Remagnino, G. L. Foresti. Ambient intelligence: A new multidisciplinary paradigm. 2004. IEEE Transactions on systems, man, and cybernetics-Part A: Systems and humans, 35(1), 1-6

[2] K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten, J.-C. Burgelman. Scenarios for Ambient Intelligence in 2010, IST Advisory Group Final Rep., Eur. Comm., Feb. 2001.

[3] S. D. T. Kelly, N. K. Suryadevara, S. C. Mukhopadhyay. Towards the implementation of IoT for environmental condition monitoring in homes. 2013. IEEE sensors journal, 13(10), 3846-3853.

[4] S. Robinson. Simulation: the practice of model development and use. Vol. 50. Chichester: Wiley, 2004.

[5] N. Matloff. Introduction to discrete-event simulation and the simpy language. Davis, CA. Dept of Computer Science. University of California at Davis. Retrieved on August 2.2009 (2008): 1-33.

[6] N. Shirehjini, A. Ali, F. Klar. 3DSim: rapid prototyping ambient intelligence. Proceedings of the 2005 joint conference on Smart objects and ambient intelligence: innovative context-aware services: usages and technologies. ACM, 2005.

[7] S. Gaglio, G. Lo Re, M. Morana. Human Activity Recognition Process Using 3-D Posture Data. Human-Machine Systems, IEEE Transactions on, vol.45, no.5, pp.586-597.

[8] S. Helal, et al. Persim-Simulator for human activities in pervasive spaces. 2011 Seventh International Conference on Intelligent Environments. IEEE, 2011.

[9] M. Buchmayr, W. Kurschl, J. Küng. A simulator for generating and visualizing sensor data for ambient intelligence environments. Procedia Computer Science 5 (2011): 90-97.

[10] T. Van Nguyen, K. Jin Gook, C. Deokjai. ISS: The Interactive Smart home Simulator. Paper presented at: In Proc. of the 11th International Conference on Advanced Communication Technology, 2009; Phoenix Park, Korea.

[11] J. Park, M. Moon, S. Hwang. CASS: A Context-Aware Simulation System for Smart Home. Paper presented at: In Proc. of the 5th ACIS International Conference on

Software Engineering Research, Management & Applications, 2007; Washington DC, USA.

[12] K. Lyytinen, Y. Youngjin. Ubiquitous computing. Communications of the ACM 45.12 (2002): 63-96.

[13] A. De Paola, P. Ferraro, S. Gaglio, G. Lo Re, S. K. Dad. An adaptive bayesian system for context-aware data fusion in smart environments. IEEE Transactions on Mobile Computing, 16(6), 1502-1515.

[14] V. Agate, A. De Paola, G. Lo Re, M. Morana. A platform for the evaluation of distributed reputation algorithms. Proceedings on the 2018 IEEE/ACM 22nd International Symposium on Distributed Simulation and Real Time Applications (DS-RT) (pp. 1-8). IEEE.

[15] A. Aho, M. Lam, R. Sethi, J. D. Ullman. Compilers: Principles, techniques, and tools. Pearson Education. 2006.

[16] A. De Paola, P. Ferraro, G. Lo Re, M. Morana, M. Ortolani. A fog-based hybrid intelligent system for energy saving in smart buildings. Journal of Ambient Intelligence and Humanized Computing, 11, 2793-2807.

[17] JFlex, lexical analyzer generator. www.jflex.de

[18] Construction of Useful Parsers. www2.cs.tum.edu/projects/cup/

[19] D. Cook. Learning setting-generalized activity models for smart spaces. IEEE Intelligent Systems, 2011.

[20] A. De Paola, P. Ferraro, S. Gaglio, G. Lo Re. Autonomic Behaviors in an Ambient Intelligence System. Proceedings of the 2014 IEEE Symposium on Computational Intelligence for Human-like Intelligence (IEEE SSCI 2014).

[21] N. J. Nilsson. Intelligenza Artificiale. 2002.

[22] V. Agate, P. Ferraro, S. Gaglio. A Cognitive Architecture for Ambient Intelligence Systems. Proceedings of the 6th International Workshop on Artificial Intelligence and Cognition (AIC 2018).

[23] MathWorks. Supervised Learning Workflow and Algorithms. mathworks.com/help/stats/supervised-learning-machine-learning-workflow-and-algorithms.html

[24] Ö. Kişi, E. Uncuoğlu. Comparison of three back-propagation training algorithms for two case studies, 2005.

[25] M. Riedmiller, H. Braun. Rprop - A Fast Adaptive Learning Algorithm. Proceedings of the International Symposium on Computer and Information Science VII, 1992.

[26] A. De Paola, P. Ferraro, S. Gaglio, G. Lo Re, M. Morana, M. Ortolani, D. Peri. A Context-aware System for Ambient Assisted Living. Proceedings of the 11th International Conference on Ubiquitous Computing and Ambient Intelligence (UCAmI 2017).

[27] Z. Boger, H. Guterman. Knowledge extraction from artificial neural network models. IEEE Systems, Man, and Cybernetics Conference, Orlando, FL, USA, 1997.

[28] A. De Paola, P. Ferraro, S. Gaglio, G. Lo Re, M. Morana, M. Ortolani, D. Peri. An Ambient Intelligence System for Assisted Living. Proceedings of the International Annual Conference of AEIT (2017).

[29] D. Voas, P. Williamson. Evaluating goodness-of-fit measures for synthetic microdata. Geographical and Environmental Modelling, 5(2), 177-200, 2001.

[30] A. De Paola, G. Lo Re, M. Morana, M. Ortolani. SmartBuildings: an AmI System for Energy Efficiency. Proceedings of the 4th International Conference on Sustainable Internet and ICT for Sustainability, 2015.

[31] R. Agrawal, C. Faloutsos, A. Swami. Efficient similarity search in sequence databases. International conference on foundations of data organization and algorithms, pp. 69-84. Springer, Berlin, Heidelberg, 1993.

[32] A.V. Oppenheim, R.W. Schafer. Elaborazione numerica dei segnali. 1994.

[33] A. De Paola, P. Ferraro, S. Gaglio, G. Lo Re. Context-Awareness for Multi-Sensor Data Fusion in Smart Environments. Proceedings of the 15th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2016).

[34] H. Zhang, E. Arens, W. Pasut. Air temperature thresholds for indoor comfort and perceived air quality. Building Research & Information, 39:2, 134-144, 2001.