

RELAZIONE DI PROGETTO DI "SISTEMI AUTONOMI"

Bombero

Alessio Addimando 0000796304

Edoardo Antonini 0000797351



BOMBERO

Indice

1	Introduzione	4
1.1	Idea generale	4
1.2	Moduli principali	5
1.3	Prototipo e demo	6
2	Aspetti fondamentali e richiami teorici	7
2.1	Sistemi Autonomi	7
2.2	Agenti autonomi e sistemi intenzionali	8
2.2.1	Reasoning	10
2.2.2	Agenti BDI	10
2.3	Coordinazione e sistemi auto-organizzativi	12
2.4	Approccio oggettivo e soggettivo alla coordinazione:	13
3	Stato dell'arte	14
4	Analisi dei requisiti	15
4.1	Intervista e acquisizione requisiti	15
4.2	Requisiti funzionali	15
4.2.1	Servizi human-supervisioning (Caserma VV.FF. e/o proprietario di un terreno):	15
4.2.2	Servizi self-organizing:	16
4.3	Requisiti non funzionali	16
4.4	Componenti del sistema Bombero	17
4.4.1	Supervisor	17
4.4.2	Patrol	17
4.5	Possibile Topologia	18
4.6	Considerazioni pre-sviluppo	20
5	Progettazione	21
5.1	Il paradigma utilizzato	21
5.2	Infrastruttura di test	22
5.3	Definizione prototipo	22
5.4	Gestione sensori	23

5.5	Definizione comportamento dell'agente	24
5.6	Gestione allarme	26
5.7	Comunicazione Agente/Sensori	28
5.8	Comunicazione tra agenti	28
6	Implementazione	30
6.1	Realizzazione hardware	30
6.2	Realizzazione software	30
6.3	Tecnologie utilizzate	31
6.3.1	AOP con Jason	31
6.3.2	Coordinazione in TuCSoN	33
6.3.3	Distribuzione in JADE	34
6.4	Dettaglio algoritmi	34
6.4.1	Calcolo rischio	34
6.4.2	Triangolazione del fuoco	35
6.4.3	Scelta del leader	35
6.5	Problemi riguardanti le librerie	37
7	Testing e performance	38
7.1	Stress & recovery Test	38
8	Divisione del lavoro	39
9	Resilienza dell'infrastruttura e continuità operativa	40
9.1	Availability	40
9.2	Scalability	41
10	Conclusioni	42
	Riferimenti bibliografici	43

1 Introduzione

1.1 Idea generale

Lo scopo del progetto, come il nome lascia facilmente intuire, incentra la sua attenzione sulla realizzazione di un sistema autonomo e distribuito real-time per far fronte al pericolo di incendi boschivi.

Soprattutto nel periodo estivo i terreni coltivati, o maggiormente quelli lasciati incolti, acquistano una certa pericolosità dato l'incremento significativo del rischio di incendio che viene generato.

Il progetto, a tal proposito, gestisce a fondo il problema del monitoraggio di aree nella maggior parte dei casi isolate e distanti dal centro abitato, quindi difficilmente controllabili.

Ad oggi è possibile individuare quattro tipologie di incendio ben distinte:

- *Incendio radente*: si propaga nel sottobosco della foresta, bruciando la lettiera, i cespugli, le erbe ed i detriti morti; si può diffondere anche in formazioni cespugliose, nei prati e nei pascoli.
- *Incendio di chioma*: si propaga dalla cima di un albero all'altra. Può avere una relazione minore o maggiore con l'incendio radente ed il più delle volte è questo che scatena l'incendio di chioma.
- *Incendio di barriera*: incendio di chioma + incendio radente, che formano un unico fronte di fiamma.
- *Incendio sotterraneo*: si propaga all'interno della lettiera e dell'humus, nello strato che copre il suolo minerale. Avanza molto lentamente, ma può essere molto grave perché danneggia l'apparato radicale delle piante.

Il progetto gestisce, a tal proposito, la prima tipologia, anche intesa come prima fase critica d'innescio.

Rispetto ad un fattore di "gravità" crescente, il sistema, rigorosamente in autonomia, coordina i tre processi successivamente elencati:

- *Prevenzione(e monitoraggio)*: fase di precisa rilevazione, tramite una rete di sensori uniformemente distribuiti, delle grandezze fisiche significative per il

calcolo del rischio e per la determinazione sempre aggiornata dello status del terreno in questione.

- *Alerting*: attraverso un processo di triangolazione della posizione della zona ad alto rischio di infiammabilità viene effettuata una comunicazione della situazione di pericolo a caserme VV.FF. e/o al proprietario del terreno.
- *Intervento (possibile estensione)*: al fine di rallentare la combustione può essere introdotta un'attività di umidificazione della zona a rischio in attesa dell'arrivo dei VV.FF., o qualsivoglia ente incaricato.

1.2 Moduli principali

A livello prettamente teorico, sulla falsa riga di quanto elencato in precedenza, si possono individuare, nell'idea generale, quattro moduli cardine:

- *Intelligenza*: l'attività computazionale, e quindi tutta la logica, è incentrata sulla fase di prevenzione, ossia il continuo monitoraggio della situazione up-to-date del sito tramite lo scambio di dati inerenti alle varie grandezze rilevate dai sensori.
In aggiunta, il sistema presenta come task complementare la capacità di effettuare, allo scoppiare di un incendio (spontaneo o doloso), una triangolazione atta alla rilevazione della posizione e successiva comunicazione d'emergenza.
- *Connettività*: si distinguono due classi di nodi; i **supervisor** e le **patrol**.
I supervisor, gerarchicamente superiori, ricevono dati dalle patrol, copiosamente replicate nel sito, elaborandone informazioni sostanziali per l'attuazione dei meccanismi sopra citati.
- *Sensing*: in seguito a un attento studio, e con la gentile consulenza degli enti di competenza, è stato individuato un set di grandezze fisiche sensato per monitorare al meglio un'area prestabilita:

- Temperatura;
 - Umidità relativa (percentuale);
 - Monossido di Carbonio (CO);
 - Valori anemometrici (velocità del vento).
- *Attuatori*: Possibile uso di una pompa collegata ad un irrigatore per far fronte alla fase di intervento, precedentemente citata.

1.3 Prototipo e demo

L'elaborato per una prima fase di simulazione e demo, è stato sviluppato, e assemblato, con l'ausilio di 3 single-board computer (Raspberry Pi 2 Model B).

Uno di questi, in particolare, è parte integrante di una stazione ignifuga ed interconnessa con gli altri nodi, e rappresenta il primo prototipo reale.

Per questo primo Proof of Concept, inoltre, il gruppo applicativo *1 supervisor + N patrol* è aggregato (e simulato) in un unico nodo predefinito (singolo Raspberry).

2 Aspetti fondamentali e richiami teorici

2.1 Sistemi Autonomi

Un sistema autonomo è definito come un complesso di funzioni autoorganizzate e dinamiche autocontrollate che determinano il dominio in cui questo opera. Fondamentalmente un sistema autonomo possiede un set di input e output che, presi da soli, non regolano e determinano il sistema. Per cui, ciò che ne controlla e regola l'interazione con l'ambiente a cui appartiene, sono solo proprio i suoi processi *self-production*. Più in generale, i sistemi autonomi:

- generano, preservano e regolano una rete interna interdipendente;
- stabiliscono un rapporto attivo con l'ambiente in cui si trovano;
- stabiliscono un'autonomia temporale;
- specificano le proprie regole interne e reagiscono agli stimoli esterni (input/output) secondo uno schema autodeterminato in relazione alle disposizioni e alle condizioni interne.
- posseggono un elevato indice di robustezza, che ne diventa pre-requisito per l'autonomia stessa, producendo un determinato output a fronte di una determinata perturbazione;
- stabiliscono una interdipendenza tra il sistema stesso e le sue parti che tende alla differenziazione funzionale in sottosistemi distinti.

IN MERITO AL PROGETTO: Bombero può definirsi un sistema autonomo in quanto rispetta al meglio i punti sopracitati. In particolare, definisce un ordine indipendente interno dove vengono differenziati due sottotipi di componenti che interagiscono con l'ambiente circostante reagendo alle mutazioni continue a cui è soggetto.

2.2 Agenti autonomi e sistemi intenzionali

L'aumento esponenziale della complessità del software ha portato i sistemi computazionali ad assumere una natura pervasiva e autonoma. Queste caratteristiche mettono in crisi i tradizionali metodi di Software Engineering portando verso un cambio di paradigma. La programmazione ad agenti va oltre quella ad oggetti ed incapsula, oltre allo stato e al comportamento, il flusso di controllo. Gli agenti quindi sono entità attive, autonome e che non possono essere invocate. Negli agenti stessi, a tal merito, è incapsulato il criterio per auto-governare il proprio flusso di controllo. Nel paradigma agent-oriented, durante lo scambio di messaggi, si trasmettono solo i dati e non il controllo. Il data flow va considerato indipendentemente dal controllo che, invece, è decentralizzato. Gli agenti quindi comunicano scambiandosi informazioni; per esempio, tramite l'Agent Communication Language (ACL). Questi possono essere considerati come singoli individui di un sistema multi agente (MAS), dove rappresentano un singolo flusso di controllo tra una molteplicità; inoltre, possono essere distribuiti nell'ambiente. Nei sistemi multi-agente l'interazione è una dimensione cardine. Infatti, gli agenti devono comunicare tra loro e con l'ambiente in modo da ottenere le informazioni per portare a termine i loro compiti.

In accordo con la definizione canonica, quindi, un agente "intelligente" è un sistema computazionale capace di svolgere *azioni* autonome a fronte di specifiche *percezioni* in un ambiente.

Le azioni cambiano l'ambiente al fine di soddisfare gli obiettivi dell'agente. Un'azione rappresenta fundamentalmente una decisione dell'agente stesso. Il suo risultato è, a sua volta, un'altra azione che mappa sequenze di percezioni alle azioni.

La percezione, invece, è un processo che permette all'agente di riconoscere lo stato up-to-date dell'ambiente e, di conseguenza, riadattare il propri comportamenti. Il risultato di una percezione è un "percelto".

Nei MAS, inoltre, sono presenti sia obiettivi individuali che globali e l'unico modo per gestirli entrambi, senza farli collidere, è usare le interazioni. Riassumendo, un agente presenta le seguenti caratteristiche:

- **Autonomia:** incapsula il controllo, non ha interfacce e non può essere invocato.

- Interattività: in quanto componente di un MAS, deve interagire con il sistema scambiando informazione e conoscenza.
- Proattività: in quanto parte attiva autonoma di un sistema rende possibile l'esecuzione di eventi piuttosto che aspettare passivamente riadattandosi in base ai cambiamenti del proprio ambiente.
- Reattività: il cambiamento dell'ambiente dove è situato deve provocarne reazioni tempestive. Un agente puramente reattivo basa le sue "reazioni" interamente sul presente non riferendosi a stati passati trasformando uno stimolo in una risposta (azione)
- Socialità: vive e interagisce interdipendentemente nella sua "società di agenti" e nel MAS, unico "contenitore legittimo". L'agente si coordina con i suoi simili cooperando, collaborando, ma anche entrando, se necessario, in competizione con quest'ultimi.
- Situatedness: è strettamente collegato con l'ambiente in cui si trova e con il quale interagisce in maniera esclusiva.

Nasce da questo l'idea di definire un agente come un sistema "intenzionale", ossia prevedere uno strato astratto di computazione grazie al quale sia capace di spiegare, comprendere e modellare a suo favore l'ambiente circostante. A questo proposito gli agenti posseggono in memoria un'esplicita, e simbolica, rappresentazione del modello del loro mondo grazie alla quale possono, a fronte di un simbolico *ragionamento*, prendere decisioni su quali azioni intraprendere per completare i propri goal. Chiaramente ad ogni istante di tempo ci sono molti scenari in cui l'ambiente può evolversi rendendolo essenzialmente non-deterministico.

Di conseguenza, anche l'agente data la molteplicità di azioni che può eseguire in ogni istante di tempo è assolutamente non-deterministico.

Un agente con "stato mentale" comprende, a tal proposito:

- stati epistemici: rappresentano la conoscenza del mondo (i.e. *percepts, beliefs*)
- stati motivazionali: rappresentano gli obiettivi che essi raggiungono (i.e. *goals, desires*)

Il processo di selezione di un'azione da eseguire tra le molte disponibili in base ad un attuale "stato mentale" è chiamato *practical reasoning*.

2.2.1 Reasoning

Si definiscono di seguito le pratiche di reasoning di un agente:

- Practical reasoning, ragionamento indirizzato alle azioni, il processo mediante il quale si decide cosa fare per raggiungere lo scopo desiderato
- Epistemic reasoning, ragionamento indirizzato alla conoscenza, include l'aggiornamento delle informazioni e la cancellazione di quelle non più valide sostituendole con altre.

Il practical reasoning stesso si divide in due attività principali:

- Deliberation
- Means-end reasoning

La *deliberation* indica il processo nel quale l'agente decide quale obiettivo vuole raggiungere, mentre il *means-end reasoning* specifica in che modo farlo. In pratica la prima attività definisce le *intentions*, cioè cosa l'agente "intende" raggiungere/fare, mentre la seconda rappresenta la selezione di un certo insieme di azioni specifiche per quell'*intention*.

2.2.2 Agenti BDI

Gli agenti BDI derivano il loro nome dalle tre caratteristiche che li contraddistinguono, cioè:

- I *Beliefs*, che rappresentano la visione del mondo dal punto di vista dell'agente, comprendendo quindi, lo stato interno, lo stato dell'ambiente e i messaggi dagli altri agenti. Sono perciò unità informative che riflettono l'attuale conoscenza che l'agente ha, per cui sono soggetti spesso a cambiamento.
- I *Desires*, cioè ciò che l'agente desidera raggiungere, possono essere intesi come sinonimo di *goal*

- Le *Intentions*, sono i modi in cui un agente cerca di raggiungere un certo scopo e sono solitamente implementati come dei *plans*. L'agente può avere molti *desires*, ma solo un certo numero di *intentions* attive in un certo momento, che, spesso, vengono eseguite concorrentemente

Ci sono inoltre delle caratteristiche secondarie, in particolare:

- La *plan library* che contiene l'insieme dei *plans* utilizzabili dall'agente, solitamente i *plans* sono delle routine aventi un corpo che definisce l'insieme delle attività da svolgere.
- L'*event queue* che rappresenta una coda dove vengono immagazzinati gli eventi (interni ed esterni)

Un generico control loop di un agente BDI è quindi definito dalla seguente sequenza di azioni:

- Percezione dell'ambiente
- Aggiornamento *Belifs*
- Calcolo dei possibili *Desires* (fase di *option-generation*)
- Scelta dell'*intentions* da applicare tra tutte le possibili alternative (fase di *deliberation*)
- Scelta del piano per eseguire l'*intention*
- Esecuzione del piano

IN MERITO AL PROGETTO: In Bombero i vari *bomberini* che compongono la "società di agenti" aderiscono al meglio alle caratteristiche elencate precedentemente rispetto alla definizione di agente. Durante la progettazione si è deciso di creare un MAS basato su agenti BDI per definire la logica dell'applicazione e, successivamente, in fase di implementazione si è scelto *Jason* come framework per lo sviluppo. Quest'ultimo, infatti, definisce degli agenti che hanno un reasoning cycle molto simile a quello sopra descritto, con delle leggere modifiche atte a rendere più semplice un'implementazione concreta. Inoltre, come sarà meglio definito in seguito, Jason utilizza il *means-end reasoning* durante il *BDI planning*

2.3 Coordinazione e sistemi auto-organizzativi

Per capacità di *self-organisation* si intende una combinazione delle seguenti caratteristiche:

- autonomia, dove l'interazione con il mondo esterno è permessa a patto che il controllo non sia delegato;
- adattività, che permette al sistema di rispondere ai cambiamenti esterni opportunamente;
- dinamicità, non limitata alla presenza di uno stato finale ma un processo logico continuo.

Si è in precedenza parlato della necessità per un sistema autonomo di godere della capacità di riorganizzarsi, in maniera quindi da presentarsi adattivo e riuscire a rispondere agli stimoli esterni. Modellare un sistema auto-organizzativo tramite un MAS e quindi utilizzare un'astrazione ad agenti può essere la giusta scelta; a patto che si rispettino i seguenti requisiti:

- autonomia e incapsulamento dell'ambiente.
- azioni locali e percezioni.
- ambiente distribuito che supporti *interazione e coordinazione*.
- supporto per la organizzazione e la cooperazione.

La coordinazione, inoltre, è definita come la capacità di gestire lo spazio di interazione. In un sistema autonomo questa è fondamentale ed a tal scopo è possibile utilizzare coordination model di due tipi:

- control-oriented, cioè basati sull'atto di comunicazione
- data-oriented, cioè basati sull'informazione scambiata durante la comunicazione. Un particolare esempio è la coordinazione Tuple-based (LINDA / TuCSoN).

Un sistema di coordinazione Tuple-based, in particolare, presenta i seguenti elementi:

- spazi di tuple, intesi come set di dati organizzati in strutture chiamate *tuple*
- linguaggio di comunicazione, basato su template e meccanismi di tuple matching (i.e. Prolog e unification per TuCSoN)
- linguaggio di coordinazione, inteso come set di operazioni disponibile per gestire le tuple nello spazio (i.e. *out,in,rd* in TuCSoN).

IN MERITO AL PROGETTO: Bombero può definirsi un SOS in quanto gli agenti Jason permettono autonomia e incapsulano l'ambiente, agiscono in base a proprie percezioni, si coordinano e riorganizzano grazie ad un supporto tuple-based (TuCSoN) e interagiscono in un sistema puramente distribuito (JADE).

2.4 Approccio oggettivo e soggettivo alla coordinazione:

L'approccio oggettivo si ha quando i concetti relativi alla coordinazione vengono estratti dagli agenti e definiti in astrazioni esterne (es. coordination artifacts). In quello soggettivo, duale al precedente, i problemi di coordinazione sono direttamente affrontati dagli agenti al fine di raggiungere nel modo migliore il proprio obiettivo. Quest'ultimo può non bastare a definire modelli di coordinazione complessi o a gestire le interazioni agente-ambiente. È perciò buona pratica, in sistemi più estesi, unire i due approcci facendo attenzione a non minare l'autonomia degli agenti.

IN MERITO AL PROGETTO: In Bombero è stato utilizzato un approccio ibrido che unisce la soggettività con lo scambio di messaggi in una certa fase dell'esecuzione, e l'oggettività per la coordinazione, utilizzando dei *coordination artifacts* come *TuCSoN*, in un'altra.

3 Stato dell'arte

La letteratura riguardante le reti di sensori è molto vasta, per cui ci si è concentrati sulle applicazioni inerenti ai problemi da noi affrontati, cioè la prevention e early detection degli incendi forestali.

Come si può vedere in [3] già dal 2008 si iniziano a studiare reti di sensori low-power e low-cost per l'analisi del rischio d'incendio. In quest'articolo, come nel più moderno [1] si utilizza ZigBee come protocollo di comunicazione all'interno della WSN gerarchica; in [2] invece si parla di XBee, un protocollo comunque molto simile. Tutti gli articoli esaminati si concentrano molto nel fatto di mandare in sleep i propri dispositivi quando non utili in modo da diminuire il consumo energetico. Essendo articoli accademici di un certo livello tutti gli approcci utilizzano sensori non-consumer al contrario di quello che il nostro gruppo ha potuto fare. Tuttavia si è cercato di prendere le idee più rilevanti dei vari articoli e cercare di applicarle al meglio.

Le soluzioni degli articoli sopra citati non si occupano del problema dell'ingegneria del sistema distribuito in questione, perciò è stato ritenuto opportuno utilizzare degli strumenti particolari riguardanti questo aspetto. In particolare ci si è concentrati sullo sviluppo di un sistema *multi-agente* (MAS) dove gli agenti si coordinano tramite uno *spazio di tuple*. In letteratura abbiamo trovato solo un'applicazione della logica ad agenti riguardante il nostro caso di studio, cioè quella mostrata in [4]. Tuttavia in questo caso gli agenti vengono usati per coordinare diversi sistemi di supporto alle decisioni (DSS) e quindi con un approccio totalmente diverso da quello proposto da Bombero. La logica di sviluppo del progetto, infatti, si basa sull'uso del paradigma ad agenti per conseguire la buona ingegnerizzazione del sistema, consentendo modularità e riusabilità, ma anche caratteristiche fondamentali nei sistemi distribuiti come, per esempio, la *failure recovery*. Per fare ciò si è ricorso all'uso di framework noti a livello di ricerca, come *Jason*[5], *JADE*[6] e *TuCSon*[7]

Come ultima nota, viene segnalato che in [1] e [3] per il calcolo del rischio si cita il *Canadian Fire Weather Index*; un sistema abbastanza complesso che fa riferimento a molti parametri. Per semplicità, in Bombero, si è utilizzato uno schema più semplice, come il *SPC Fire Weather Forecast Criteria*¹

¹www.spc.noaa.gov/misc/Critical.Criteria_for_web.pdf

4 Analisi dei requisiti

4.1 Intervista e acquisizione requisiti

Prima di iniziare a svolgere questo progetto è stato deciso che fosse necessario un modello fisico reale del caso di studio per poter sviluppare un prototipo realistico. Per fare ciò sono state acquisite informazioni da varie fonti.

Oltre a cercare informazioni in letteratura, si è ritenuto necessario chiedere a chi si occupa attualmente della problematica analizzata, cioè i *Vigili del Fuoco*. Molti dei requisiti sotto-elencati quindi sono stati estratti da un'intervista fatta con il direttore regionale del Corpo VV.FF. G.B. Pordon, della sede di Bologna, in collaborazione con ARPA Emilia-Romagna. Altri, invece, sono arrivati dai vigili, seppur non specializzati, della sede di Cesena. Quest'analisi è, ovviamente, da definirsi adatta allo sviluppo di un primo sistema concreto, ma da affinare notevolmente per eventuali applicazioni commerciali future.

4.2 Requisiti funzionali

Dall'analisi del caso di studio sono stati estratti i seguenti requisiti funzionali.

4.2.1 Servizi human-supervisioning (Caserma VV.FF. e/o proprietario di un terreno):

- Visualizzazione status area d'interesse tramite interfaccia web semplice ed esplicativa.
- Visualizzazione eventuali aumenti significativi dell'indice di rischio incendi in relazione all'area individuata.
- Visualizzazione tempestiva di eventuali situazioni di emergenza date dalla presenza di un incendio.

4.2.2 Servizi self-organizing:

- Calcolo del rischio di incendio in una zona.
- Rilevazione dati continua da sensori di grandezze utili per il monitoraggio dell'area.
- Possibilità di segnalare una situazione di emergenza e procedere alla triangolazione della posizione individuata.
- Controllo dei parametri delle sotto-zone in cui i dispositivi del sistema sono dislocati (i.e. temperatura/livelli di monossido di carbonio/altri sensori)
- Procedere con l'irrigazione preventiva della zona al fine di umidificare e prevenire l'espansione dell'incendio.

4.3 Requisiti non funzionali

Il sistema in studio, data la sua natura potenzialmente distribuita, autonoma e real-time cercherà, ove possibile, di soddisfare i seguenti requisiti non funzionali:

- Autonomia e auto-organizzazione, il sistema creerà una "società di membri" in cui, interagendo con il mondo esterno si riorganizzerà rispetto ai propri bisogni e allo stato dell'ambiente.
- Trasparenza, in relazione a meccanismi ed operazioni eseguite in back- ground, che permettono un utilizzo fluido e ininterrotto dei servizi offerti dal sistema.
- Scalabilità, il sistema deve supportare la presenza di N supervisor e M patrol, con M due o più volte maggiore rispetto ad N.
- Tolleranza ai malfunzionamenti e all'inaffidabilità della rete.

Il sistema deve, ove possibile, continuare ad operare e a facilitare il ripristino a fronte di failure di alcune delle sue parti.

4.4 Componenti del sistema Bombero

In risposta ai requisiti sopracitati, in fase di progettazione è stato ritenuto opportuno modellare il sistema secondo le seguenti sotto parti:

- Servizio lato web, destinato agli utenti per il monitoraggio della situazione up-to-date della zona di riferimento.
- Core del sistema, composto da due classi di componenti gerarchicamente collegate (*Supervisor* e *Patrol*)

Il core del sistema, a tal proposito, deve rispondere ai seguenti requisiti:

4.4.1 Supervisor

Si presenteranno come entità attive, interconnessi con i loro simili, implementeranno meccanismi di gestione della scalabilità e di fault tolerance oltre, naturalmente, all'intera logica applicativa del sistema.

4.4.2 Patrol

Ogni *Supervisor* individuerà, gerarchicamente più in basso, l'entità *Patrol*, o "sentinella". La sentinella, replicata in una rete di sensori, sarà un'entità pressoché passiva adibita alla raccolta dati dei vari sensori, che vengono comunicati, in formato raw, al proprio controllore di riferimento.

Il sistema, seguendo la logica di questa divisione dovrà quindi, in autonomia:

- Effettuare il continuo monitoraggio dello status dell'area di interesse analizzando i dati provenienti dalla comunicazione real-time di ogni supervisor con le sue rispettive patrol.
- Effettuare il calcolo del rischio di incombenza di un possibile incendio in base alla rivelazione di valori standard e anomali.
- In caso di emergenza, in presenza di un possibile incendio, determinare tramite algoritmi di triangolazione della posizione il probabile punto di innesco dell'incendio comunicandolo tramite l'apposito servizio web insieme ad un alert (i.e. dashboard descrittiva).
- Effettuare l'umidificazione del terreno "a rischio".

4.5 Possibile Topologia

Come si evince dalla figura, la rete presenterà le *patrol* (in marrone), in gran numero visto anche il costo ridotto, che ricoprono uniformemente l'intero ambiente di monitoraggio e i *supervisor* (in verde) che formeranno delle sottoreti autonome comunicando con le proprie *patrol*.

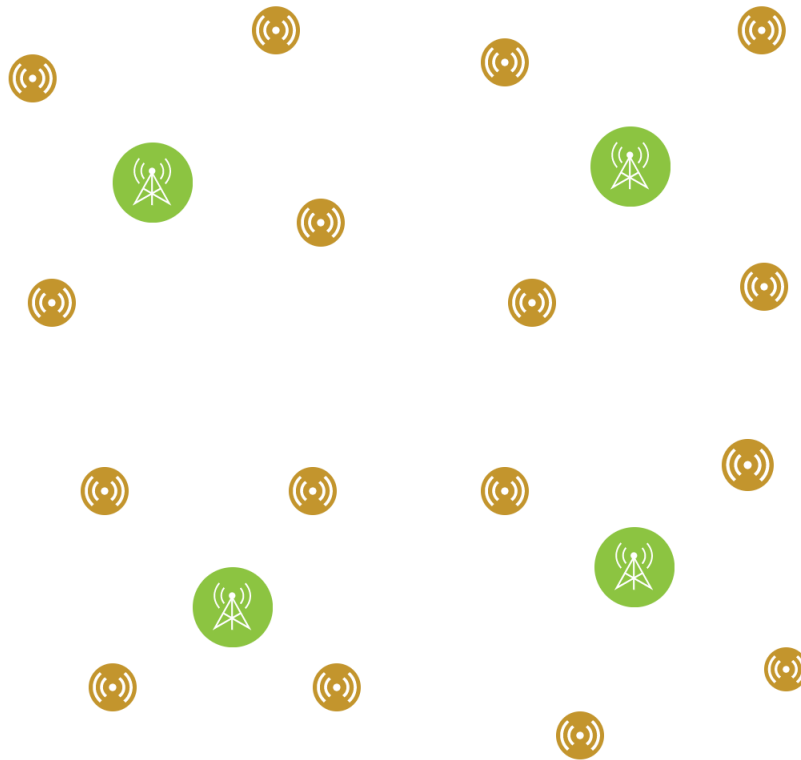


Figura 1: Possibile topologia Bombero in un'area medio-piccola

In merito a quanto scritto finora è riportato di seguito, in figura 2 il corrispondente diagramma dei casi d'uso.

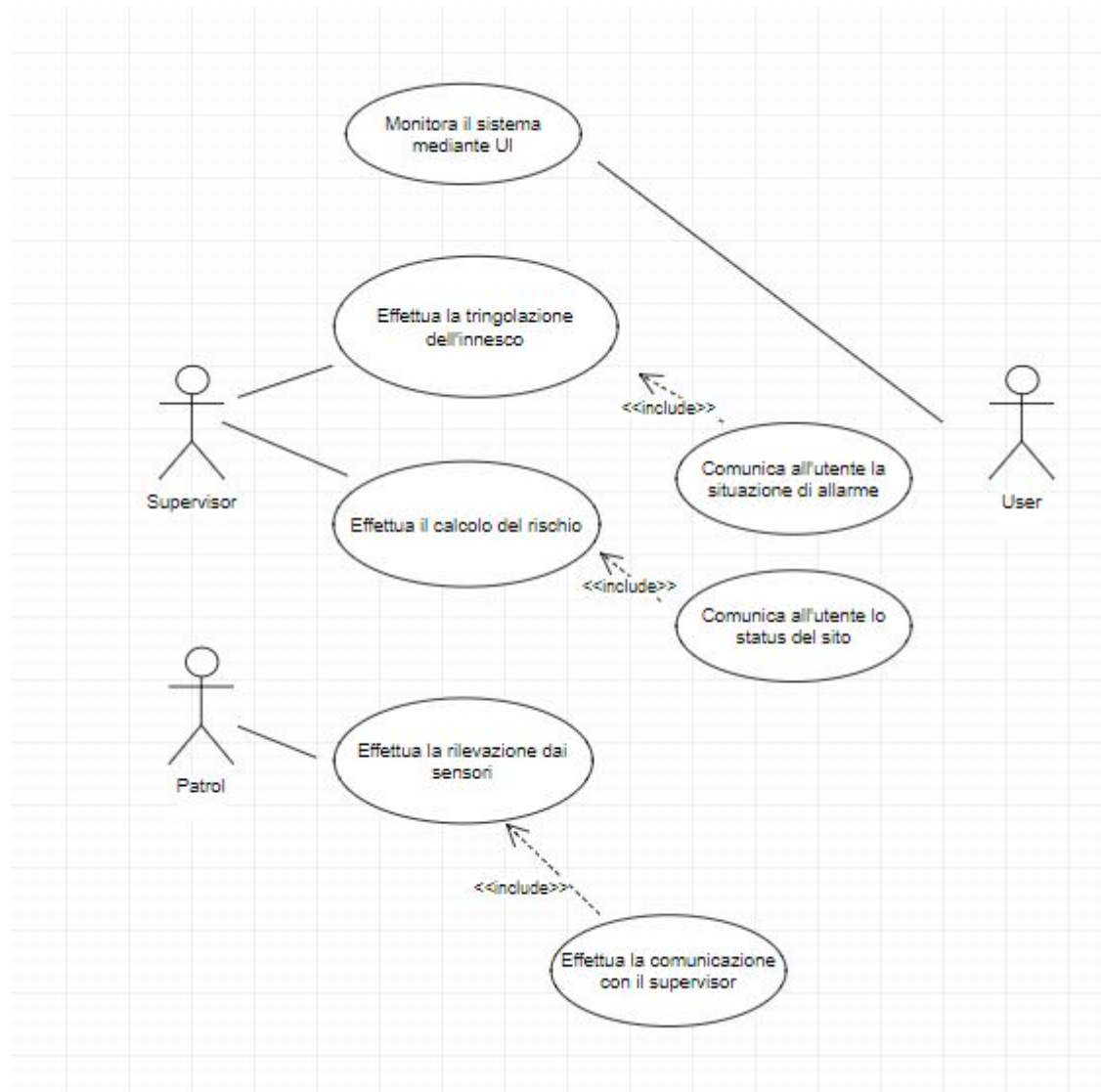


Figura 2: Diagramma dei casi d'uso Bombero

È da notare come, essendo un sistema autonomo, l'utente abbia un ruolo molto limitato e per lo più passivo, anche se molto importante, mentre il resto dei compiti venga svolto automaticamente dagli altri "attori" del diagramma

4.6 Considerazioni pre-sviluppo

Consapevoli che l'implementazione di un sistema come questo che sia completo in ogni sua parte non potrà probabilmente essere realizzato nelle tempistiche determinate dal carico di lavoro dell'esame, ci si è posti come obiettivo quello di sviluppare man mano un prototipo, che rispondesse il più possibile ai requisiti elencati. Per tale motivo, fin da subito, si è pensato che alcune parti del sistema dovranno necessariamente essere simulate o aggregate ad altri componenti.

Infine, sempre per le medesime motivazioni, si è deciso di considerare il modulo di *Intervento* solo una possibile estensione, viste anche le scarse possibilità di effettuare, attualmente, un processo di testing sensato. A tal proposito segue la fase di progettazione dove è dettagliatamente esplicitato il processo di impostazione dell'infrastruttura di test dell'elaborato.

5 Progettazione

5.1 Il paradigma utilizzato

Durante la progettazione ci è posto il problema di decidere quale sarebbe stato il paradigma più adatto allo svolgimento del progetto. La scelta è ricaduta in un misto tra Object-Oriented Programming (OOP) e Agent-Oriented Programming (AOP). In pratica, il flusso di controllo principale, cioè la parte core dell'applicazione è progettata tramite un agente, cioè un'entità proattiva/reattiva e autonoma, che, appunto, "agisce" sull'ambiente e percepisce i cambiamenti dello stesso.

Ovviamente, per mantenere una buona struttura del software è stato diviso il flusso di controllo dalla gestione dell'ambiente; quest'ultimo viene, infatti, gestito con un classico paradigma ad oggetti. Per concludere è opportuno specificare che la progettazione della View si è limitata a una semplice dashboard web per la notifica degli eventi, essendo il progetto da considerarsi poco correlato con gli utenti e con la user experience. Ciò tuttavia andrà rivalutato in futuro.

La scelta dell' AOP è stata fatta in base alle caratteristiche del progetto in sé, infatti i sistemi distribuiti hanno delle proprietà particolari che motivano questa decisione:

- La mancanza di clock centralizzati
- L'openess, cioè la possibilità di modifiche alla struttura e dimensione del sistema
- La situatedness, cioè il fatto che la computazione debba essere "aware" del luogo in cui avviene
- La failure recovery
- La necessità di coordinazione tra le varie entità

L'AOP nasce proprio per risolvere questi problemi tramite le proprietà degli agenti, tra cui le già citate proattività, reattività e, naturalmente, autonomia. Inoltre è importante evidenziare che gli agenti hanno un proprio flusso di controllo completamente separato dal flusso dati e indipendente da quest'ultimo. Infatti, il data flow tra agenti normalmente avviene tramite scambi asincroni di messaggi,

in questo modo né il mittente né il destinatario dovranno bloccare il loro flusso di controllo per ottenere le informazioni.

Ovviamente la contaminazione dell'AOP con l'OOP può portare ad alcuni problemi riguardanti la gestione del flusso di controllo stesso, perciò l'impegno del team si è concentrato nel cercare di sfruttare gli aspetti positivi dei due approcci senza snaturarli eccessivamente.

Sempre nello spirito dell'AOP è stato inoltre deciso di utilizzare uno spazio di tuple per la coordinazione tra gli agenti.

5.2 Infrastruttura di test

Date le disponibilità di strumenti, per testare il prototipo si è deciso di utilizzare un'architettura molto semplice ma che comunque, a fronte di un'attenta valutazione, presentava molte delle criticità da affrontare, cioè quelle tipiche di un sistema distribuito in rete. In pratica si è deciso di collegare tre *Raspberry Pi 2* in rete per simulare tre Supervisor, cioè coloro che hanno i compiti più importanti e complessi computazionalmente. Per ottenere i dati su cui lavorare vengono collegati agli stessi un sensore di temperatura e umidità e un sensore di gas/fumo. Di seguito è illustrata la struttura del sistema di protezione fisica dei supervisor.

5.3 Definizione prototipo

Uno dei problemi emersi fin da subito è stato quello di progettare un sistema per salvaguardare i supervisor sia dagli incendi che dagli agenti atmosferici. Per cui, dopo aver raccolto alcune informazioni dai VV.FF. è stata ideata la stazione in figura 3

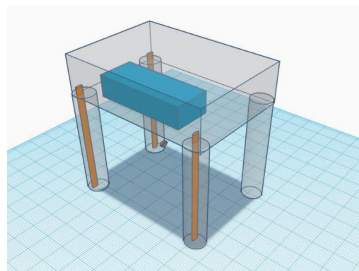


Figura 3: Modello 3D della stazione di protezione per il supervisor

5.4 Gestione sensori

A livello software una delle parti più importanti ha riguardato la gestione dei sensori. Ovviamente si è cercato di disegnare il sistema in modo da renderlo il più possibile "sensor-independent" e quindi cercando di astrarre il più possibile dall'effettiva implementazione, inoltre si è fatto in modo di rendere semplice l'aggiunta di nuovi tipi di sensori. Ciò ha portato a progettare una gerarchia di interfacce come definita qui sotto nella figura 4

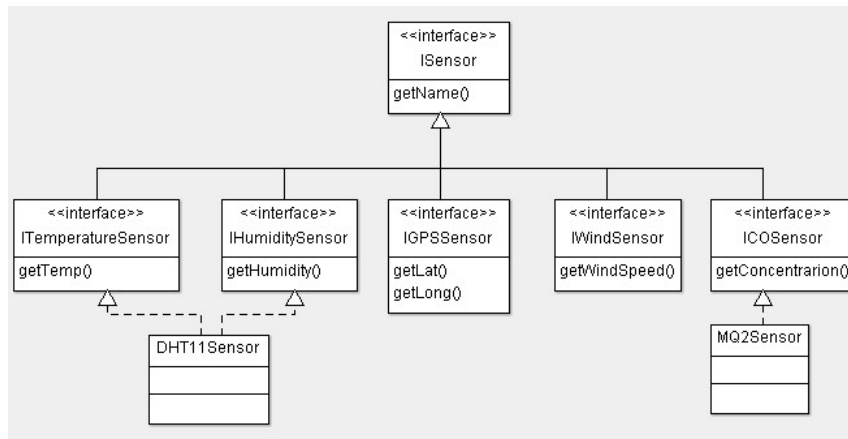


Figura 4: Diagramma di classe riguardante i sensori

Come visibile sono stati progettati i 5 tipi di sensore più comuni in un'applicazione riguardante il caso di studio cioè:

- Temperatura
- Umidità
- GPS
- Valori anemometrici (velocità del vento)
- Monossido di Carbonio (CO)

Come già detto, la gerarchia è facilmente estendibile ad altri tipi meno comuni di sensori. Si può vedere, inoltre, come vengano supportati anche sensori particolari, come il DHT11, che restituisce dati per più di una grandezza.

5.5 Definizione comportamento dell'agente

Il flusso di controllo di un singolo supervisor è stato progettato tramite il comportamento di un agente seguendo la logica BDI [11].

Ad alto livello possiamo definire come *desires* dell'agente:

- Eleggere un leader
- Avviare il tuple space
- Analizzare i dati provenienti dai sensori e eventualmente notificarli

A design-time sono definiti i vari *plans* che l'agente può utilizzare per raggiungere i propri *desires*, di seguito ne elenchiamo alcuni:

- La creazione di un anello logico per la scelta del leader.
- L'avvio dello spazio di tuple.
- Il recupero dei dati dai sensori.
- La scrittura dei dati nello spazio di tuple.
- Il calcolo del rischio d'incendio.
- Il check della presenza di un incendio e l'eventuale tentativo di triangolazione del punto di innesco.
- L'avvio di un servizio web per la notifica dello status della zona.

Ovviamente il numero dei *plans* applicabili per una *intention* sarà dipendente dai *belifs* attivi in un certo momento, perciò non tutti i task elencati verranno svolti da tutti gli agenti del sistema. Per esempio, solo l'agente che avrà nella sua *belifs base* un *literal* che lo identifica come leader potrà svolgere certi compiti. In figura 5 si mostra un diagramma di attività che dettaglia la struttura delle azioni sopra citate in un run considerato "normale" dell'applicazione, in pratica si mostra una possibile scelta dei *plans* relativi alle *intentions*.

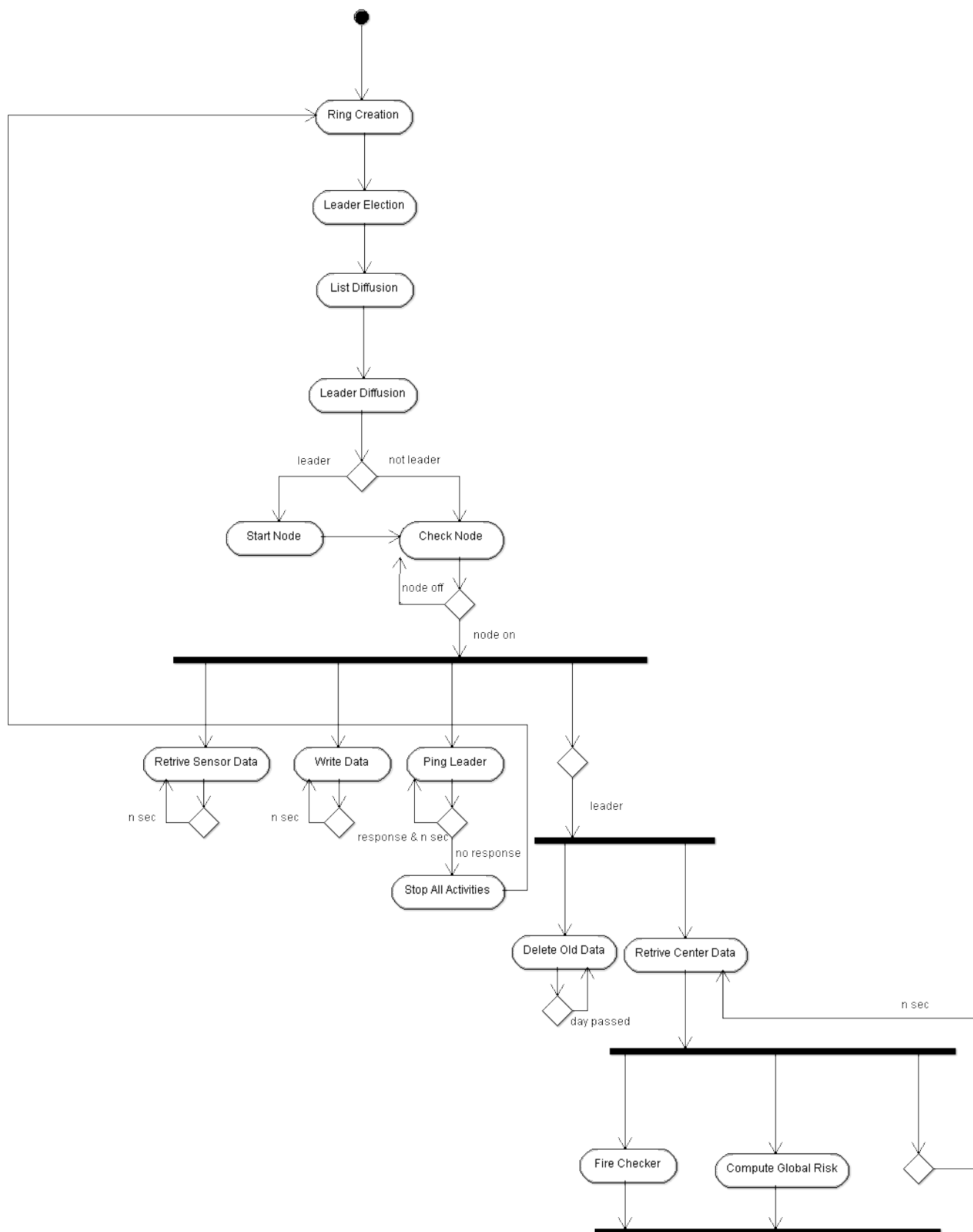


Figura 5: Diagramma di attività dell'agente

I passaggi iniziali, da "Ring Creation" a "Leader Diffusion", mostrano l'applicazione dell'algoritmo di scelta di leader ad anello, come spiegato in [9]; tramite questo procedimento si ottiene un leader all'interno dei supervisor che si occuperà di alcune funzioni più importanti e computazionalmente pesanti. Già da appena eletto, il leader avvierà il nodo su cui far girare lo spazio di tuple per la coordinazione, mentre gli altri si sospenderanno attendendo l'attivazione dello stesso, fondamentale al funzionamento del programma.

A tuple space avviato i vari supervisor iniziano i loro vari compiti in parallelo, tra questi si vede la lettura dei dati dai sensori e la scrittura dei dati nello spazio di tuple.

Un altro task molto importante è quello del ping al leader per controllare il suo corretto funzionamento. Infatti, in caso di malfunzionamenti al nodo centrale, il primo agente ad accorgersene sospenderà tutte le attività e inizierà un'altra leader election. Ciò avviene perché l'agente cerca di raggiungere nuovamente il *desires* di elezione del leader; è da notare che in una situazione del genere nessun altro *desire* appartiene quello citato è un'opzione, per cui l'agente non può attivare *intentions* che riguardino il raggiungimento di altri *desires*.

Come già accennato, ci sono delle funzioni particolari svolte solo dal leader, come la cancellazione dal tuple space dei dati non più utili (*semel per diem*) e il recupero delle grandezze misurate dallo stesso per analizzarle. Una volta ottenuti i dati, quindi, sarà possibile esaminarli per calcolare il rischio d'incendio generale e l'eventuale presenza di un incendio già avviato, in quest'ultimo caso il sistema provvederà a effettuare una triangolazione del possibile punto di innesco del fuoco in base ai dati dei sensori gps incrociati con le quantità di CO presenti nell'aria. Il diagramma dei attività mostrato non è sufficientemente espressivo per mostrare nella sua interezza la logica di un agente BDI, tuttavia si è pensato che potesse essere utile a intuire ad alto livello il suo comportamento, serve inoltre per mostrare come le varie *intentions* procedano in parallelo.

5.6 Gestione allarme

Quando il leader rileva la presenza di situazioni anomale, come un elevato rischio d'incendio (*calcolo del rischio*) o la presenza di un possibile incendio (*triangolazione*), questo genera un allarme che è visibile ai diretti interessati (VV.FF. e/o

proprietario del terreno) tramite un servizio web dedicato. Per effettuare ciò in fase di progettazione si è deciso di fare in modo che il supervisor possa mandare dati ad un client in ascolto segnalando i dati d'interesse mediante una dashboard semplice e intuitiva.

Di seguito è riportato il diagramma di sequenza che esplica nel dettaglio il servizio sopra descritto.

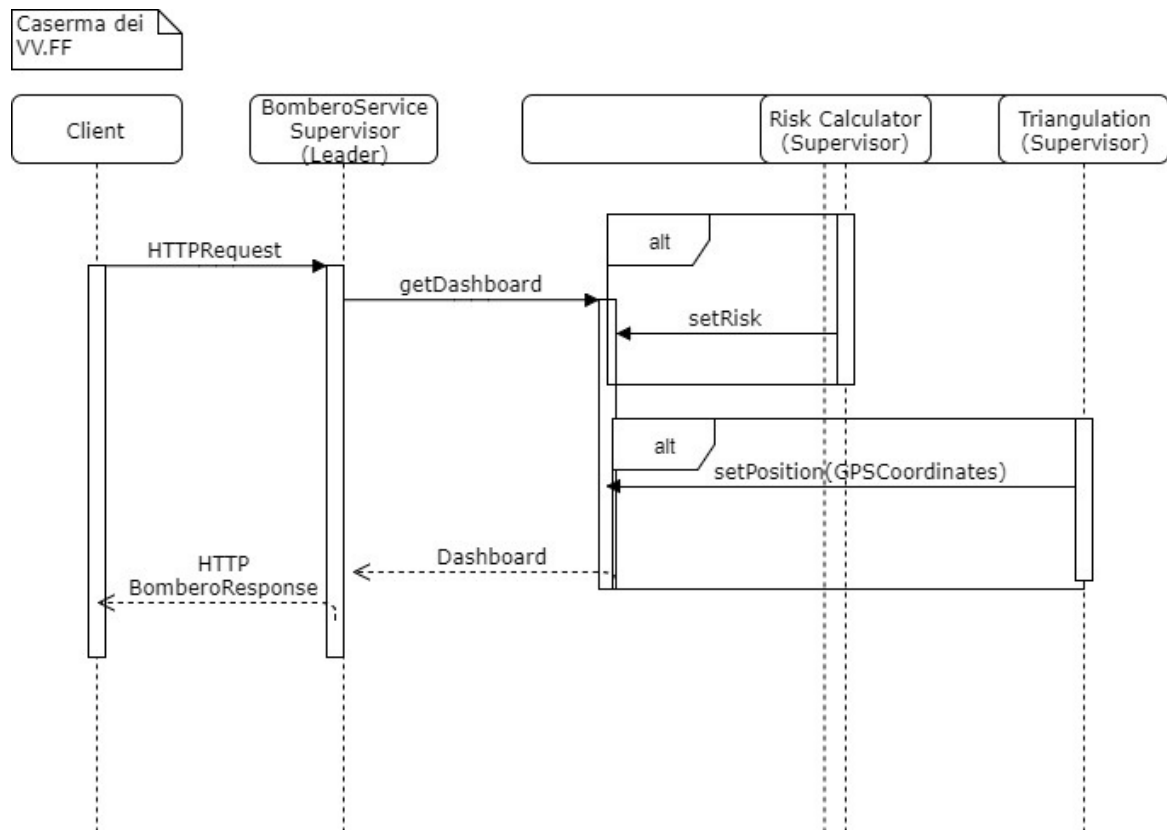


Figura 6: Diagramma di sequenza servizio di allarme remoto

Come è possibile evincere dalla figura 6 il client effettua una richiesta in remoto, mediante protocollo applicativo HTTP (*HyperText Transfer Protocol*), al nodo che è leader in quel momento. A questo punto il servizio effettua una richiesta interna della struttura della dashboard da restituire. La dashboard è modellata rispetto alla situazione che in quel momento è presente nell'area tenuta sotto controllo dall'agente.

Se il rischio supera una certa soglia, a fronte del calcolo precedentemente effettua-

to, questo viene settato e ne determina la modellazione di alcuni elementi grafici appositi che ne dimostrano la veridicità (i.e. messaggio di errore, videata d'allarme, etc).

Analogamente, se è stata anche rilevata la presenza di un incendio attivo, a fronte del processo di triangolazione, nella dashboard vengono settate anche le coordinate del punto più vicino all'incendio.

A questo punto la dashboard viene generata tramite linguaggio di Markup (*i.e. HTML*) e mostrata a video su browser lato utente. In caso contrario la UI restituita presenta una struttura semplice in cui viene soltanto notificata la presenza di una situazione standard priva di anomalie.

5.7 Comunicazione Agente/Sensori

La comunicazione tra gli agenti e i sensori, siano essi nei supervisor o nelle patrol, avviene nello stesso modo. Il metodo utilizzato è simile a un request-response tra client e server, infatti l'agente chiede i dati ai vari sensori che rispondono con le grandezze misurate. Un metodo alternativo potrebbe comportare una situazione in cui i dati dei sensori vengono inviati all'agente tramite un altro flusso di controllo generando un interrupt; tuttavia ciò risulterebbe come una deviazione rispetto al tradizionale metodo ad agenti, che prevede che quest'ultimi non ricevano chiamate.

5.8 Comunicazione tra agenti

In questa sezione si cercherà di specificare più in dettaglio come avviene la comunicazione tra i vari agenti facenti parte del sistema. I modelli di comunicazione adottati sono due:

- Lo scambio di messaggi asincrono;
- La coordinazione mediante uno spazio di tuple.

Il primo modello di comunicazione viene usato quando lo spazio di tuple non è presente o quando si assume che non funzioni come dovrebbe; in quel caso, per esempio, le funzioni di leader election e quella di detection della caduta di un nodo utilizzano questa modalità perché totalmente indipendente da qualsiasi altra

entità. Sebbene ottima per le fasi di setup della rete fisica e logica, questa modalità diventa scomoda e verbosa per quanto riguarda la comunicazione durante l'esecuzione del sistema. Il secondo modello di comunicazione, invece, è molto più comodo per compiti a sistema già "in piedi" e quindi viene usato per la coordinazione degli agenti nei loro task. I due metodi di comunicazione, essendo usati in momenti diversi non interferiscono e non generano problemi tra loro.

6 Implementazione

6.1 Realizzazione hardware

A livello hardware sono state realizzati più moduli:

- Il collegamento dei sensori con i vari Raspberry
- La realizzazione del prototipo per la protezione dei supervisor

Seguendo il modello 3D preparato in fase di progettazione si sono saldate quattro gambe cave a un parallelepipedo anch'esso vuoto e senza la faccia superiore, in modo da poter inserire al suo interno i vari strumenti necessari, tra cui la coperta isotermica, il Raspberry e un power bank per l'alimentazione. I cavi per i sensori sono stati inseriti all'interno delle gambe

6.2 Realizzazione software

Come già accennato la realizzazione del progetto è avvenuta usando Java come linguaggio di programmazione. Ciò non ha permesso di ottimizzare le performance, ma ha garantito di poter strutturare il sistema in maniera modulare e riusabile. Come primo step nell'illustrazione dell'implementazione mostriamo la divisione in package

- **env**; rappresenta l'ambiente del sistema, contiene la rappresentazione dell'environment agli occhi dell'agente
- **sensors**; riguarda tutto ciò che ha a che fare con i sensori, dalla gerarchia di interfacce ad alto livello ai dettagli implementativi per l'esecuzione su Raspberry
- **tucson4jason.***; contiene le classi che permettono la comunicazione tra TuCSoN e Jason²
- **utils.***; package con al suo interno funzioni di utilità generiche
- **utils.internalActions**; sub-package di utils da considerare a parte poiché contenente le classi che permettono di estendere le funzionalità di Jason

²più dettagli qui: <https://apice.unibo.it/xwiki/bin/view/TuCSoN/4Jason>

- **webInterface**; rappresenta l'insieme di classi per la gestione dell'interfaccia utente per la notifica di allarmi

Di seguito si elencano le tecnologie utilizzate e le loro caratteristiche.

6.3 Tecnologie utilizzate

Come premessa è necessario specificare che molte delle tecnologie citate sono sviluppate in ambito di ricerca e quindi non sono effettivamente stabili. A titolo di esempio si noti che per quanto riguarda Jason è stato necessario inviare varie mail agli sviluppatori per chiedere la risoluzione di alcuni bug.

6.3.1 AOP con Jason

Jason[5] è un framework per sistemi multi-agente che estende l'architettura *AgentSpeak*[10] per agenti BDI (Beliefs, desires, intentions) aggiungendo alcune funzionalità agli stessi e fornendo una modalità più pratica per la programmazione. Il linguaggio sviluppato è ad alto livello, Prolog-like e viene usato per definire i goal dell'agente. Per le funzioni di basso livello Jason si basa su Java, in questo modo si possono aggiungere varie estensioni (*Internal Actions*) in un linguaggio a oggetti. Il reasoning cycle di Jason è formato dalle seguenti passi:

- Analizzare l'environment
- Aggiornare la *belief base*
- Ricevere le comunicazione dagli altri agenti
- Scegliere i messaggi accettabili
- Scegliere un evento
- Recuperare tutti i *Plans*
- Determinare quelli applicabili
- Sceglierne uno
- Scegliere un *intentions* per l'esecuzione e eseguirla

Com'è possibile vedere è semplicemente una visione più volta all'implementazione di una classica architettura BDI, inoltre specifichiamo che nel nostro progetto la comunicazione tra agenti via messaggi è limitata alla leader election, mentre per tutto il resto viene utilizzato lo spazio di tuple.

Jason ha alcune caratteristiche che permettono una programmazione più agevole di un MAS come per esempio:

- il costrutto $!!\Pi$, con Π *plan* per eseguire il *plan* in una nuova *intention* slegata dalla precedente, ciò permette di far eseguire routine concorrenti in maniera molto semplice
- il costrutto $-+\Phi$, con Φ *belif*, per aggiornare un *belif* in modo atomico
- il costrutto $-!g$, con g *goal*, che permette di gestire la failure di un *plan*, è incredibilmente utile per fare in modo piuttosto semplice l'handling dei vari errori possibili
- *Plan* e *Belifs annotations* che specificano informazioni riguardo agli stessi e che possono essere utilizzate durante la programmazione. Alcune *annotations* sono presenti di default mentre altre possono essere create. Questo strumento può essere usato per rendere ancora più potente l'error handling
- Le internal actions, che, come già accennato, sono un meccanismo per estendere le funzionalità del software, anche in questo caso ne sono già presenti un buon numero, ma se ne possono definire altre a piacere. Alcune di queste sono utili anche per il message passing.
- L'Environment, per definire l'ambiente dove saranno situati gli agenti, questo meccanismo tuttavia ha dei problemi in alcuni casi, come verrà spiegato in seguito.

Per concludere viene specificato come avviene il planning in Jason.

Il meccanismo è chiamato BDI planning e utilizza un approccio dove i piani sono specificati a design-time, ma la loro esecuzione dipende dal *means-end reasoning* in base alle condizioni real-time del contesto.

6.3.2 Coordinazione in TuCSoN

TuCSoN[7] è un modello di coordinazione basato su spazio di tuple in particolare pensato per la sincronizzazione, cooperazione e coordinazione tra agenti autonomi. La sintassi delle sue operazioni è particolarmente semplice e anch'esso utilizza dei predicati Prolog-like. A grandi linee un sistema TuCSoN è una collezione di agenti e spazi di tuple coordinati in un set di nodi distribuiti. Questo, più formalmente, è composto dalle seguenti entità:

- Un *coordination media* in ReSpecT per i *tuple center*
- Dei *TuCSoN node* che rappresentano un'astrazione topologica e che ospitano i tuple center.
- *Agenti TuCSoN*, da coordinare.

Per modellare i requisiti di Bombero nella coordinazione il set limitato di primitive TuCSoN è stato più che sufficiente. Per scelte tecnologiche il nodo TuCSoN viene posto in ascolto su una porta non privilegiata (di default *20504*) nell'agente al momento leader e, in caso di eventuale caduta di quest'ultimo, viene fatto ripartire in quello che sarà poi identificato come nuovo leader; ciò al fine di garantire continuità nel servizio. Mediante la definizione di un determinato *tuple template* è possibile individuare, e gestirle in base alla primitiva, il set di tuple che effettuano *matching* con quest'ultimo e sono quindi di interesse per la computazione. Le primitive utilizzate sono state:

- `in(p)`, per prelevare i valori attuali delle grandezze fisiche che descrivono in quel momento lo stato dell'ambiente rispetto ad ogni agente (i.e. `in(agentIP,data(temp(_),hum(_),monoxide(_)...))`);
- `rd(All)`, per leggere, e non rimuovere, i valori aggiornati dei valori delle grandezze fisiche ricercati, presenti nel tuple center in un preciso istante.
- `out`, analogamente alle precedenti esegue l'inserimento nello spazio di tuple dei valori provenienti dalle interazioni di ogni agente con l'ambiente tramite il modulo di sensoristica.

Rispetto agli altri framework per spazi di tuple, questa soluzione è stata scelta per la presenza di una integrazione già sviluppata con Jason, cioè Tucson4Jason³.

6.3.3 Distribuzione in JADE

JADE[6], come noto, è anch'esso un framework ad agenti, ma in Bombero è utilizzato solo come piattaforma di supporto alla distribuzione. Infatti Jason non ha meccanismi di default per spostare gli agenti nella rete e si appoggia all'infrastruttura di JADE per farlo. Inoltre è possibile utilizzare la JADE infrastructure per far interoperare agenti Jason con agenti JADE, per farlo viene usato il linguaggio standard di comunicazione fra agenti, l'ACL (Agent-Communication Language). Tuttavia nel progetto non si è fatto uso di quest'ultima modalità.

6.4 Dettaglio algoritmi

Di seguito esponiamo in dettaglio alcuni algoritmi utilizzati nel progetto

6.4.1 Calcolo rischio

Come già accennato, per il calcolo del rischio è stata utilizzata una variazione del SPC Fire Weather Forecast Criteria⁴. Nell'algoritmo, per ogni grandezza si calcola il valore medio relativo alla zona di misurazione, e da questo si deriva il valore di rischio relativo come indicato nel documento citato. Successivamente, in base ai vari valori di rischio di tutte le grandezze si calcola il rischio globale, al momento questa operazione è stata implementata come una semplice media tra i rischi relativi. In seguito, si effettua il mapping da valori numerici alla scala seguente:

- Extremely Critical
- Critical
- Elevated
- None

³<https://apice.unibo.it/xwiki/bin/view/TuCSon/4Jason>

⁴www.spc.noaa.gov/misc/Critical.Criteria_for_web.pdf

6.4.2 Triangolazione del fuoco

Per la triangolazione del fuoco ci si è affidati all'algoritmo del *Geographical Midpoint* descritto in dettaglio qui: www.geomidpoint.com/calculation.html.

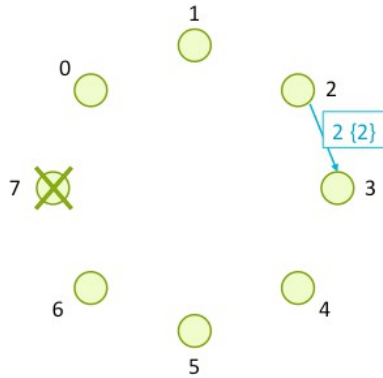
Nel caso proposto dal sito, l'algoritmo viene utilizzato per trovare il centro esatto tra i più posti dove una persona ha vissuto, perciò in questo caso vengono usati i giorni per pesare le varie località espresse con coordinate GPS. L'approccio nel caso di studio si è limitato alla sostituzione dei giorni con il valore rappresentante quanto la concentrazione di CO sia andata sopra soglia. Si è quindi convertito l'algoritmo per stimare la probabile posizione di una fonte di fumo e, di conseguenza, di un possibile incendio.

6.4.3 Scelta del leader

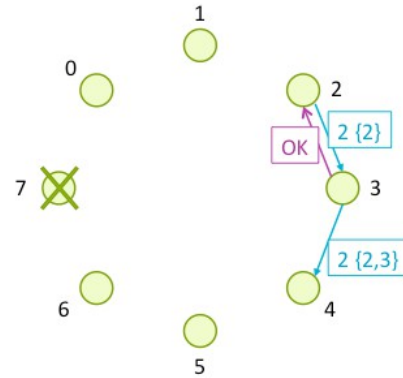
Per la scelta del leader si è utilizzato l'algoritmo di scelta del leader ad anello[9]; quest'ultimo considera i partecipanti al sistema in un anello logico e trasmette i messaggi lungo lo stesso. Nella versione base, quando un nodo si accorge che il leader è caduto, questo manda al nodo successivo nell'anello un messaggio ELECTION contenente due campi:

- Initiator
- List

A cui assegna il proprio identificatore (figura 7a)⁵. Il nodo che riceve il messaggio aggiunge il suo identificatore alla lista, inoltra il messaggio lungo l'anello e risponde al nodo precedente con un messaggio OK (figura 7b).

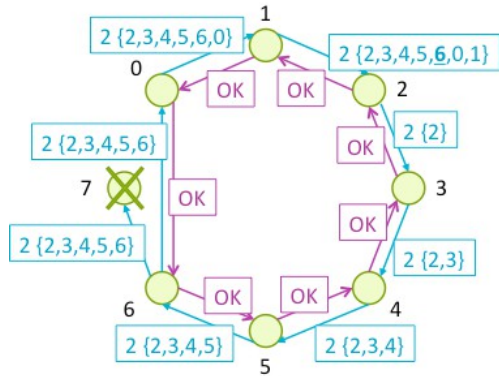


(a) Passo 1 - Leader Election

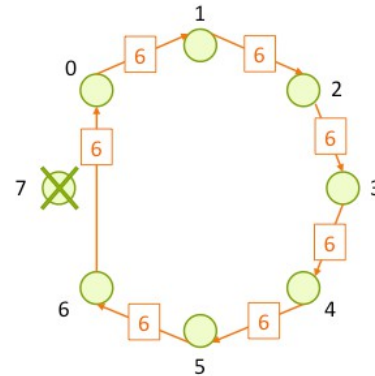


(b) Passo 2 - Leader Election

Quando il messaggio election ritorna all'initiator (figura 8a) esso sceglie il nodo con identificatore massimo e invia un messaggio COORDINATOR contenente il nuovo leader (figura 8b)



(a) Passo 3 - Leader Election



(b) Passo 4 - Leader Election

L'algoritmo implementato è molto simile. Fanno eccezione alcune variazioni necessarie per gestire messaggi che non raggiungono la destinazione e nodi che cadono durante la creazione dell'anello.

⁵Tutte le immagini citate per questo algoritmo provengono dalle slide del corso di Concorrenza y Sistemas Distribuidos della prof. Estefania Argente UPV Valencia A.A. 2016/17

6.5 Problemi riguardanti le librerie

Durante l'utilizzo delle librerie citate sono stati riscontrati alcuni problemi riconducibili alla loro implementazione.

Per quanto riguarda JADE si è notato che l'invio di messaggi tra un agente e l'altro non veniva sempre effettuato, e ciò sembrava indipendente dal codice scritto.

In Jason alcune *DefaultInternalActions* come *abolish* o *all_names* non hanno funzionato come avrebbero dovuto, per cui si è dovuto ricorrere ad alcuni workaround che hanno sporcato il codice. La seconda funzione citata, in particolare, avrebbe dovuto dare come output solo gli agenti *alive* in un certo momento, invece, in ogni caso, restituiva la lista di agenti presenti all'avvio del sistema. Un altro problema è stato riscontrato quando si è iniziato a testare la distribuzione degli agenti, in pratica, con JADE, gli agenti Jason non riuscivano più a contattare efficacemente l'environment, per cui è stato necessario creare delle internal actions che permettessero di effettuare il binding con l'ambiente.

In TuCSoN abbiamo notato solo che, sporadicamente, il nodo non riusciva ad avviarsi nel modo corretto. Ciò a dimostrazione che questi framework non sono per ora completamente stabili.

7 Testing e performance

7.1 Stress & recovery Test

Il sistema è stato testato tramite una serie di long-run durante i quali sono state monitorate tutte le varie funzioni, dal calcolo del rischio alla triangolazione. Inoltre, durante l'esecuzione sono stati simulati appositamente degli eventi di failure (i.e. caduta di un Supervisor) al fine di testare la capacità del sistema di riorganizzarsi di conseguenza.

A parte alcuni errori particolari insorti in un piccolo numero di sessioni, la rete è sempre riuscita a rilevare la failure ed eleggere il nuovo leader, ritornando velocemente all'operatività. Inoltre, per verificare il funzionamento della fire detection e del calcolo del rischio abbiamo stimolato i sensori, per esempio generando fumo o aumentando la temperatura, in modo da osservare la reazione del sistema. Le prove effettuate non possono considerarsi esaustive perché non svolte in un contesto reale, ma comunque hanno tutte rispettato le aspettative. Il grafico seguente mostra come, in seguito a due eventi di fail (*linea nera*), il sistema, sebbene i tempi minimi di recovery, mantenga l'operabilità in tutta la durata del test di servizio.

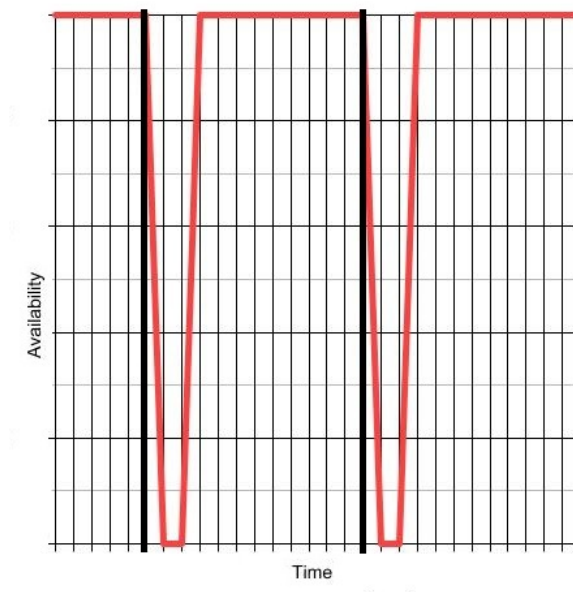


Figura 9: Variazione dell'operatività del sistema durante il test

8 Divisione del lavoro

Lo sviluppo del progetto è stato svolto dividendo i compiti come di seguito:

Alessio:

- Studio delle tecnologie di TuCSoN[7] e TuCSoN4Jason[8]
- Inizializzazione Raspberry (supervisor) e configurazione rete distribuita
- Connettori per i sensori e interfacciamento con Jason
- Interfaccia e servizio web per notifica d'allarme

Edoardo:

- Studio delle tecnologie di Jason[5] e JADE[6]
- Implementazione algoritmi di calcolo del rischio e triangolazione
- Gestione di failure recovery con leader election

Parte comune:

- Implementazione Mock
- Analisi dei requisiti e design di alto livello
- Raccolta informazioni dai VV.FF.
- Definizione architettura per test
- Design prototipo fisico

9 Resilienza dell'infrastruttura e continuità operativa

Quando si progetta un'applicazione è importante compiere le adeguate scelte architettureali per garantire il giusto grado di fruibilità del servizio al netto della presenza di eventuali guasti e problematiche legate alla sicurezza del sistema stesso.

Il sistema proposto, quindi, deve soddisfare al meglio tutti i requisiti che un'applicazione di questa natura possiede. Quest'ultimo infatti si presenta come un sistema *distribuito, efficiente, robusto, e scalabile*. Il sistema, a tal proposito, soprattutto in un'ottica di deploy su larga scala, dovrà assicurare *availability* e *scalability*, parametri indispensabili per un buon indice di resilienza.

9.1 Availability

Indipendentemente da guasti interni o altri eventi esterni (*i.e. furto di un supervisor, atto vandalico, danneggiamento a fronte di una situazione meteorologica avversa*) Bombero deve continuare ad erogare servizi in modo stabile e continuativo.

Per questo motivo devono essere definiti meccanismi per la gestione attiva della *fault tolerance* e, chiaramente, processi di *disaster recovery*.

Gia nella prima versione, il prototipo implementa un meccanismo di gestione dei guasti (vedi *Implementazione/Dettaglio Algoritmi/Leader election*) ed è in grado di far fronte a eventuali malfunzionamenti interni di un qualunque nodo garantendo il servizio in modo trasparente nei confronti dell'utente.

Non sono stati implementati, per motivi di tempo, meccanismi per il recovery di nodi soggetti a un qualsiasi genere di guasto. Cosa che, in una soluzione su larga scala è indispensabile. Per esempio, comunicare all'ente di competenza il problema; magari fornendo un report dettagliato sull'entità del danno, etc.

E' in più necessario garantire la persistenza dei dati con dei meccanismi di backup, cosa che nel prototipo non è presente, anche se lo spazio di tuple a cui si è fatto ricorso supporta funzioni per la gestione del problema.

9.2 Scalability

In un deploy su larga scala, per questo particolare dominio applicativo, una buona scalabilità del sistema è necessaria in quanto la dimensione delle aree è variabile e può andare da qualche km quadrato fino a centinaia di ettari. Per questo motivo il sistema deve permettere l'aumento dei nodi di calcolo (supervisor + rispettive patrol) aumentando la propria *scalabilità orizzontale*.

Inoltre, si può pensare ad una **"community solution"**, per coprire più aree possibili al fine di dividere il sistema in più sottogruppi indipendenti e gestire anche la presenza di più pericoli geograficamente distanti tra loro.

10 Conclusioni

Il progetto, nella sua prima forma necessaria alla consegna dell'elaborato, può definirsi soddisfacente. I requisiti del sistema, seppur a livello prettamente simulato, sono stati rispettati sia durante la progettazione più ad alto livello che nell'implementazione stessa. Tutta la fase di analisi dei requisiti è stata delineata non solo secondo un brain-storming libero dei componenti del gruppo, ma anche integrando ricerche sullo stato dell'arte della problematica e le attuali soluzioni già documentate e rese disponibili in rete.

Inoltre, non si può non citare la consulenza del Corpo dei Vigili del Fuoco, utile per l'individuazione del modello fisico a cui ispirarsi e per un'analisi dei requisiti aderente all'idea generale. Il sistema rispetta al meglio i principi fondamentali per la progettazione e lo sviluppo di un sistema distribuito e autonomo sfruttando le tecnologie del corso. I vincoli principali, oltre a quelli temporali, sono stati riscontrati nella configurazione iniziale dell'infrastruttura di rete, nella progettazione del comportamento dell'agente e nel design del prototipo fisico.

La soluzione attualmente presentata è sufficiente, ma ha sicuramente la necessità di essere migliorata a fronte di un possibile processo di reingegnerizzazione del progetto.

I test annessi dimostrano un fattore di resilienza del sistema apprezzabile, sebbene le dimensioni ridotte dell'ambiente di testing non possano definire al meglio l'effettiva efficienza del prodotto. Sicuramente, a fronte di questa prima soluzione il team ritiene che un progetto ben impostato, realizzato in termini di scalabilità e deployment in ambiente reale, necessiti di più tempo e, naturalmente, più risorse.

Riferimenti bibliografici

- [1] Molina-Pico, Antonio, et al., *Forest monitoring and wildland early fire detection by a hierarchical wireless sensor network*. Journal of Sensors 2016, 2016.
- [2] Bayo, A., et al., *Early detection and monitoring of forest fire with a wireless sensor network system*. Procedia Engineering 5, 2010: 248-251.
- [3] Zhang, Junguo, et al., *Forest fire detection system based on a ZigBee wireless sensor network*. *Frontiers of Forestry in China* 3.3, 2008: 369-374.
- [4] Jaber, Ali, Franck Guarnieri, and Jean-Luc Wybo. *Intelligent software agents for forest fire prevention and fighting*. Safety Science 39.1, 2001: 3-17.
- [5] Bordini, Rafael H., Jomi Fred Hübner, and Michael Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*. Vol. 8. John Wiley & Sons, 2007.
- [6] Bellifemine, Fabio Luigi, Giovanni Caire, and Dominic Greenwood. *Developing multi-agent systems with JADE*. Vol. 7. John Wiley & Sons, 2007.
- [7] Omicini, Andrea, and Franco Zambonelli. *TuCSoN: a coordination model for mobile information agents*. Proceedings of the 1st Workshop on Innovative Internet Information Systems. Vol. 138. 1998.
- [8] Mariani, Stefano, and Andrea Omicini. *Multi-paradigm Coordination for MAS: Integrating Heterogeneous Coordination Approaches in MAS Technologies*. WOA, 2016.
- [9] E. Chang, R. Roberts. *An improved algorithm for decentralized extremafinding in circular configurations of processes*. Communications of the ACM, 22(5): 281-283, ACM, 1979.
- [10] Rao, Anand S. *AgentSpeak (L): BDI agents speak out in a logical computable language*. European Workshop on Modelling Autonomous Agents in a Multi-Agent World. Springer, Berlin, Heidelberg, 1996.

- [11] Rao, Anand S., and Michael P. Georgeff. *An abstract architecture for rational agents*. KR 92 (1992): 439-449.