ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Ingegneria e Architettura
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

UN'INFRASTRUTTURA BASATA SU WEB PER SISTEMI DI REALTÀ AUMENTATA INTEGRATI CON INTERNET OF THINGS - PROGETTAZIONE E SVILUPPO PROTOTIPALE

Elaborato in
INGEGNERIA DEI SISTEMI SOFTWARE ADATTATIVI COMPLESSI

Relatore
Prof. ALESSANDRO RICCI

Co-relatore
Ing. ANGELO CROATTI

Presentata da
MARCO NOBILE

Terza Sessione di Laurea
Anno Accademico 2016 – 2017

PAROLE CHIAVE

Internet of Things

Web of Things

Augmented Reality

Augmented Worlds

Web of Augmented Things

Indice

Introduzione

Internet of Things e Realtà Aumentata sono due tra i topics più gettonati dall'odierno mondo IT, numerosi progetti di ricerca sono attualmente attivi in questi ambiti, e si prospetta che diventino tra i protagonisti del futuro (o anche del presente, nel caso dell'IoT) panorama tecnologico.

Lo scenario IoT prevede di costellare l'ambiente di oggetti "intelligenti" connessi in rete dotati di capacità di calcolo, sensori ed attuatori, al fine di migliorare la percezione che l'ambiente stesso ha di se stesso e di chi lo popola, permettendo di sviluppare tutta una serie di applicazioni per poterlo rendere reattivo e auto-organizzante.

La visione che propone la Realtà Aumentata (AR) può per certi aspetti risultare simile, in quanto considera degli osservatori, nel mondo reale, dotati di speciali dispositivi che permettono di *sovraimporre* ad esso uno strato virtuale di informazioni, aumentandolo.

Lo scopo di questa tesi è quello di riuscire ad integrare in un unico modello (visione) questi due grandi ambiti, e ingegnerizzare una base infrastrutturale che permetta di concepire uno scenario allargato in cui le Things pervasive con cui construire i sistemi non debbano più essere soltanto fisiche, ma possano anche avere una natura totalmente virtuale/aumentata, sensibili dall'uomo attraverso i meccanismi di AR. Per fare ciò si andrà prima alla ricerca di un valido modello che possa integrare lo strato reale e lo strato virtuale mascherando la natura delle Things (Augmented Worlds), dunque si procederà a renderle interoperabili, attraverso un protocollo di livello applicazione analogo allo stack WoT (che è nato proprio per risolvere questa problematica nell'IoT), dando forma al **Web of Augmented Things**.

Per raggiungere l'obiettivo prefissato, nei primi capitoli si trova una panoramica su WoT anche con riferimenti ingegneristici (buone pratiche per la

progettazione di sistemi WoT) , seguita da cenni riguardanti definizione, applicazioni, ruolo e tecniche utilizzate in ambito AR. Nel terzo capitolo viene trattato l'iter di scelta del modello di riferimento per WoAT, cruciale per partire con la progettazione dell'infrastruttura(quarto capitolo). Le ultime due parti sono dedicate all'implementazione e alla validazione di quanto sviluppato.

Capitolo 1

Web of Things

1.1 Transizione da IoT a WoT

Internet of Things può essere considerato, negli ultimi anni, uno dei topic più discussi e popolari in ambito IT. Nonostante trovare l'esatto confine tra cosa IoT sia e cosa non sia risulti non banale, si può descrivere la sua visione in modo semplice: Internet of Things è un sistema di oggetti fisici dotati di dispositivi elettronici abilitanti la comunicazione su diverse interfacce di rete (Internet compreso) che possono essere scoperti, monitorati, controllati e con i quali è possibile interagire.

Questi oggetti fisici a cui si fa riferimento sono direttamente associabili al concetto di *Smart Things* (che per semplicità verranno anche chiamate Things), oggetti di uso quotidiano che entrano a far parte di un sistema pervasivo attraverso la rete. Una Smart Thing, di norma, viene "aumentata" attraverso **sensori** (di temperatura, luce, etc.), **attuatori** (display, motori, etc.), **capacità di calcolo** (su cui è possibile installare del software), **interfacce di comunicazione** (wired o wireless).

Estendere i sistemi attraverso l'integrazione di Things abilità un importante set di nuove applicazioni possibili, ma al tempo stesso fa emergere una problematica importante: la nativa non *interoperabilità globale* tra le Things (data dalla diversità di vendors, vincoli hardware, avanzamento tecnologico, etc.). Succede spesso che questa problematica venga risolta utilizzando/sviluppando un protocollo di livello applicazione custom per il sistema che si vuole realizzare, senza mai però raggiungere un eterogeneità globale, obiettivo invece dell'emergente concetto di **Web of Things**.

WoT si prefigge di applicare le tecniche e gli strumenti, al livello Applicazione, dell'ormai consolidato World Wide Web per lo sviluppo di sistemi IoT, unificando le interfacce di ogni singola Thing in un unico modello, per una interoperabilità completa e globale. Così come il web è diventata la piattaforma di riferimento per l'integrazione di applicazioni distribuite in internet, il Web of Things è pensato per diventare il riferimento per l'integrazione di dispositivi (pervasivi) nelle applicazioni che desiderano interagire con essi.

Questa integrazione avviene, nella pratica, poichè in WoT i dispositivi (e i servizi che forniscono) possono essere acceduti come un qualsiasi sito web tradizionale, nelle sezioni seguenti si fornisce una descrizione dettagliata di cosa compone l'intero stack WoT e come si possono ingegnerizzare sistemi di questo tipo.

1.2 Architettura di WoT

L'architettura di Web of Things si compone di 4 livelli, ognuno dei quali permette alle Things di essere accessibili reciprocamente e integrate. I livelli sono riportati graficamente in figura 1.1.

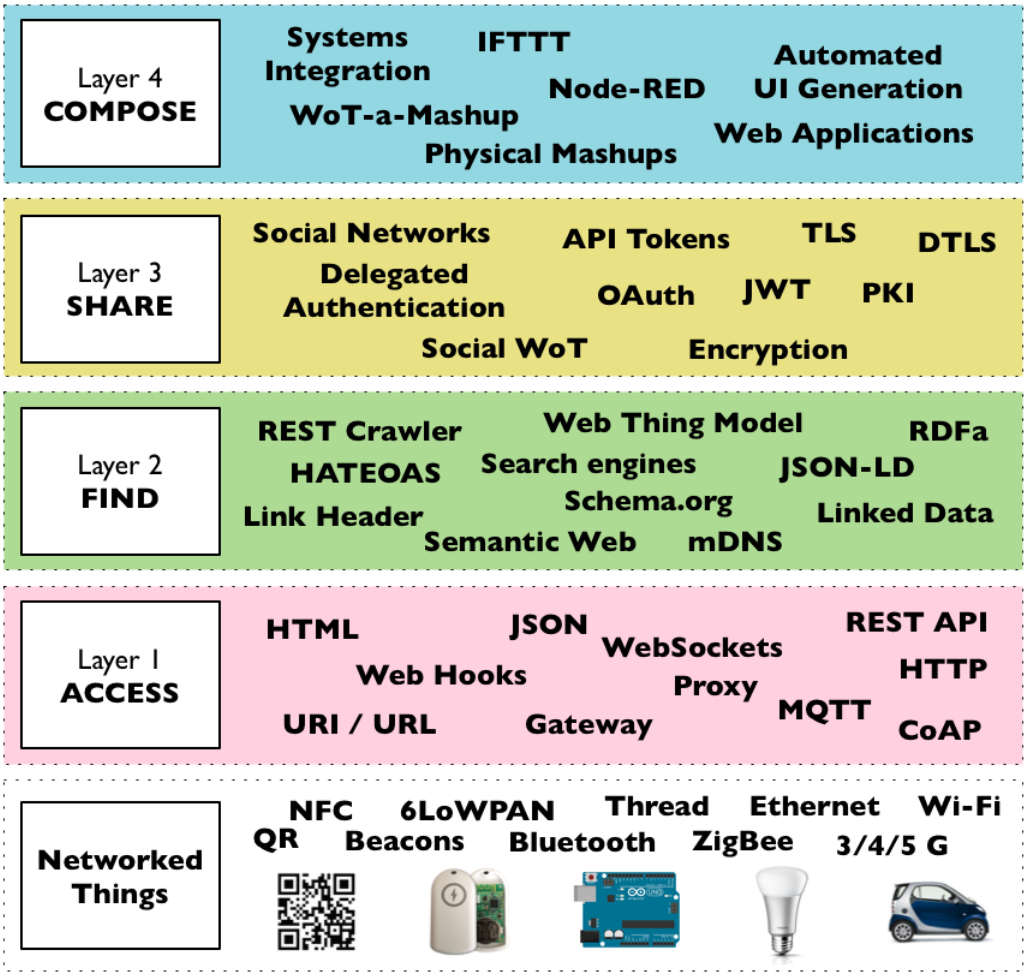
1.2.1 Access Layer

Questo livello è necessario per permettere alle "Things" di diventare "Web Things", ovvero entità simili a comuni servizi web che possano processare richieste HTTP. In particolare ogni Thing deve essere esposta attraverso RESTful API, in modo da essere integrata al massimo nel World Wide Web.

Dall'utilizzo di questo pattern viene a galla però una problematica insita nell'Internet of Things, ovvero la gestione di scenari Event-Driven derivanti, ad esempio, dall'utilizzo di sensori distribuiti nell'ambiente.

Ovviamente alcune Things possono accedere direttamente in rete attraverso connettività fissa o mobile, fornendo la propria interfaccia REST, per gli altri casi si prevede l'utilizzo di **Smart Gateways** che rendano accessibili le Things non connettibili in rete quali sensori non dotati di scheda di rete, etc.

Nelle prossime sezioni verrà trattato come progettare le RESTful API nella maniera corretta, analizzando le diverse operazioni e i modelli di interazione possibili con delle Web Things.



Source: Building the Web of Things: book.webofthings.io
Creative Commons Attribution 4.0

Figura 1.1: Architettura di Web of Things

Representational State Transfer (REST)

Nel presentare il livello corrente è stata utilizzata la parola RESTa referenziare le API. Con REST si intende un set di principi architetturali che ogni sistema distribuito può adottare ed è stato formalizzato da Roy Fielding:

REST propone una serie di vincoli architetturali che, quando applicati, enfatizzano la scalabilità delle interazioni tra i componenti, la generalizzazione

delle interfacce, il deployment indipendente dei componenti, e di componenti intermedi per ridurre la latenza, rinforzano la sicurezza, e permettono di incapsulare sistemi legacy (retrodatati)

1.2.2 Findability Layer

Una volta rese accessibili le Web Things, è compito di questo livello renderle altrettanto rintracciabili. È importante che queste siano automaticamente interrogabili e utilizzabili da ipotetiche altre things in rete. L'approccio che quindi si è scelto è quello di usare gli standards dettati dal Web Semantico per descrivere le proprietà e i servizi di ogni Thing.

Entra in gioco, per la descrizione delle Web Things, JSON come linguaggio..

1.2.3 Sharing Layer

Questo strato si occupa dello scambio di dati tra le varie Things, che deve risultare efficente e sicuro. Le considerazioni riguardo quali e quanti protocolli di sicurezza adottare (quali TLS (Transport Layer Security) o anche tecniche quali l'autenticazione web delegata (OAuth)) vengono effettuate in questo livello.

Anche per questo layer verrà trattata nelle prossime sezioni una panoramica riguardante i rischi di sicurezza di un sistema che integra WoT, e quali soluzioni poter adottare in merito.

Social Web of Things

Un altro modo per permettere uno scambio controllato e efficente di informazioni tra le Things è quello che si ispira fortemente all'idea di Social Network. Infatti, si può pensare a un oggetto nel Web come a un utente di un social network che voglia poter condividere contenuti con una certa cerchia di amici/parenti. Grazie alle API che molti social network ad oggi offrono, si possono creare delle piattaforme o delle Web Application (come ad esempio FAT, *Friends and Things*) che permettano a RESTful Things di comunicare con il proprio mondo personale.

1.2.4 Composition Layer

Si colloca in questo livello l'integrazione dei vari servizi offerti dalle Things, una volta che queste hanno avuto accesso al web (Livello 1), sono state rese

raggiungibili (Livello 2) e hanno la possibilità di comunicare in maniera sicura (Livello 3). Qui si trovano vari tool che integrano tra loro Web Things eterogenee in sistemi specifici, per esempio per fornire un livello di astrazione maggiore, dashboards programmabili, eccetera.

Sopra questo livello troviamo le applicazioni vere e proprie, quali per esempio "Xively" (aggregatore di dati provenienti da Web Things) o "ThingSpeak" .

1.3 Vantaggi derivanti dall'architettura

L'applicazione della visione Web of Things allo scenario di Internet of Things risulta una miglioria per i seguenti aspetti:

- Le singole Things grazie a REST hanno tutte un'interfaccia uniforme, questo risulta essere un mezzo molto potente in presenza di oggetti molto diversi ma che devono essere integrati nello stesso sistema.

- REST impone alle interazione di essere **stateless**, obbligando i messaggi scambiati a essere "self-descriptive". Questo lascia la responsabilità di controllare l'interazione al client, e libera il server di sovraccarichi che potrebbero inficiarne le prestazioni e la scalabilità. In questo scenario il client adotta meccanismi di caching delle risposte del server, ma deve stare attento a non utilizzare nelle iterazioni successive, risposte salvate ormai obsolete o inappropriate.

- L'interfaccia che ogni Thing espone all'esterno scherma e separa il client dal server, ovvero l'interazione che il client richiede non richiede la conoscenza dello storage presente sul server, ma solo quella del tipo di risposta. I server risultano così essere più semplici e facili da scalare.

- Le Things nel WoT possono essere considerate dei **servizi**, ed in quanto tali essi hanno i medesimi vantaggi dei servizi in una architettura SOA (Service Oriented Architecture), ovvero sono delle vere e proprie "black box" che espongono una certa interfaccia per l'utilizzo. Questo ricade nei vantaggi precedentemente elencati, in quando essendo i servizi disaccoppiati tra loro, e riusabili senza necessità di modifiche in sistemi di natura differente.

Facendo quindi un quadro generale dei vantaggi di Web of Things alla luce di quanto riportato, si può affermare che tra questi vi sia sicuramente la soluzione al problema della eterogeneicità dei dispositivi/things in gioco. Infine il

problema della scalabilità è minato dall'architettura a strati e dal paradigma RESTful.

1.4 Problemi emergenti della visione WoT

L'avvento di sistemi densamente popolati di dati e dispositivi come quelli presenti nello scenario di WoT certamente pone delle sfide importanti agli addetti alla sicurezza di questi ultimi. Bisogna ricordare che, per molte applicazioni reali possibili, potrebbero sorgere problematiche legate alla privacy necessaria delle informazioni trattate. Nella pratica, possiamo classificare le seguenti minacce possibili ad un sistema costruito in ottica Web of Things:

- Le connessioni presenti tra i dispositivi che compongono il sistema sono di per se insicure e possono essere sfruttate per la diffusione di *malware* o *worms*.
- Ogni Thing è potenzialmente vulnerabile ad attacchi quali *man-in-the-middle* o *replay* delle proprie credenziali.
- Un attaccante potrebbe "mascherarsi" fingendo di essere una Thing del sistema e richiedere informazioni/guadagnare privilegi a lui non concessi.
- Un *Denial of Service* lanciato ai danni di una o più Things potrebbero compromettere il buon funzionamento del sistema vittima dell'attacco.

1.5 Costruzione di sistemi Web of Things

Nelle sezioni precedenti è stato trattato **cosa** è Web of Things e come si compone concettualmente, nella sezione corrente verrà definito meglio **come** si dovrebbe progettare ed implementare lo stack Wot, tenendo conto di quanto trattato precedentemente in termini di problemi, vantaggi che l'architettura offre, etc.

1.5.1 Design di API per Web Things

Progettare le RESTful API per Web Things è una operazione che si colloca al livello di Accesso, e risulta cruciale in quanto effettuarla nel modo corretto significa rendere la propria Thing accessibile da librerie e web tool. Come già trattato, REST è un insieme di vincoli architetturali, è bene specificarli prima di procedere con le regole di design:

- **Client-Server**: le interazioni tra i componenti sono basate sul pattern *request-response*, in cui il client effettua la request e il server fornisce una response. In questo modo si accentua il disaccoppiamento tra le due entità, in modo da permettere al client di dover conoscere soltanto il tipo di risposta che potrà fornire il server, mentre il server non sarà obbligato a conoscere lo stato del client.

- **Interfaccia Uniforme**: essa è necessaria in scenari in cui componenti possono entrare ed uscire dal sistema in modo dinamico. Il disaccoppiamento tra essi può essere raggiunto soltanto attraverso una interfaccia globalmente uniforme.

- **Stateless**: l'interazione tra client e server deve essere completamente senza stato. per il server ogni interazione col client viene considerata uguale, ed è il client,in caso fosse necessario, a dover inserire il proprio stato nella richiesta che effettua, anche ripetutamente.

- **Caching**: ai client è permesso fare caching delle proprie interazioni col server, in modo da migliorare la scalabilità a fronte di un minor numero di richieste HTTP. I server possono, dalla loro, definire delle policies per il reload di risorse scadute/aggiornate.

- **Sistema a Livelli**: rende possibile l'utilizzo di server intermediari per migliorare la scalabilità e i tempi di risposta, oppure permette di mascherare sistemi legacy rendendoli accessibili sul web. Questo è incentivato dal disaccoppiamento ottenuto attraverso i principi precedenti.

Rendere ogni oggetto fisico accessibile attraverso le stesse regole del resto del web è il goal principale del Web of Things, per questo focalizzarsi sul concetto di *interfaccia uniforme* è la chiave per comprendere a fondo come progettare le API di Web Things. Guinard asserisce che vi siano 4 principi alla base del concetto di interfaccia uniforme:

- **Risorse Indirizzabili**: REST è una *Architettura resource-oriented*(ROA), in cui ogni thing, componente, proprietà all'interno di un sistema è una risorsa. Al fine di indirizzarle tutte univocamente, HTTP le referenzia attraverso un URL (Uniform Resource Locator), la quale sintassi comprende uno *schema* (HTTP/HTTPS), una *authority* (il nome dell'host, eventualmente correlato ad una porta), un *path* (il percorso gerarchico che porta alla risorsa) ed eventualmente dei parametri di query. Per fare un esempio, uno Led Smart che espone le proprie RESTful API avrà una proprietà *status* (ON/OFF) accessibile presumibilmente al path */led/properties/status*.

- **Manipolazione delle risorse attraverso rappresentazione**: ogni risorsa deve poter fornire una rappresentazione di se stessa, sia essa in una modalità più human-friendly (HTML), machine-readable (JSON, XML) o entrambe. Il formato suggerito per le Web Things da Guinard, come standard, è JSON, in quanto molto leggero e supportato da numerosi linguaggi.

- **Messaggi auto-descrittivi**: I client devono utilizzare solo i metodi forniti dal protocollo di riferimento: GET, POST, PUT e DELETE. Per rendere questi messaggi *auto-descrittivi*, è necessario definirne la semantica. Nel Web of Things, i 4 metodi appena citati vanno a mappare le 4 operazioni *CRUD* (create, read, update, delete). Nello specifico GET si associa all'operazione di lettura, involcarla non cambia lo stato del server ma semplicemente richiede la rappresentazione di una risorsa. POST viene associata alla creazione di risorse, la risposta a tale richiesta incorporerà l'indirizzo della risorsa appena creata. La PUT viene utilizzata per l'update, mentre DELETE si utilizza per distruggere risorse esistenti.

- **Hypermedia come motore per l'*application state***: con *application state*(AS) si intende che ogni singolo stato/proprietà presente in una applicazione WoT deve essere una risorsa RESTful. Tornando all'esempio del Led, lo status di quest'ultimo deve sempre essere raggiungibile. Il concetto di hypermedia (link tra risorse), è definito *motore* per l'AS in quanto richiedere una risorsa deve necessariamente comprendere la ricezione sia dello stato di essa, sia delle risorse a lei correlate direttamente. I links tra risorse sono fondamentali in campo WoT, in quanto consentono ai client di *scoprire* in autonomia le risorse che le Things mettono a disposizione, senza conoscerle a priori.

Chiarito e approfondito il concetto di interfaccia uniforme, si elencano le fasi che compongono il design di API per Web Things, strettamente correlate ai punti appena descritti:

- Design delle Risorse: in cui di identificano le funzionalità e i servizi di una Thing come risorse web indirizzabili.

- Design della Rappresentazione: in cui si decide quale rappresentazione fornire di ogni risorsa, sarà poi il client a scegliere, se necessario, quale richiedere.

- Design dell'interfaccia: in cui si decide quali comandi saranno possibili per ogni servizio/risorsa e i relativi codici di errore.

- Design dei collegamenti tra risorse: in cui si decide come le diverse risorse saranno legate tra loro ed in che modo esporre risorse e links.

Regole di Design

Guinard, nel suo trattato, rielabora tutti i principi appena descritti per progettare API, e propone delle regole di Design da seguire nella progettazione di API per Web Things.

- *Ogni Web Thing deve essere un server HTTP*: il supporto minimo da possedere è HTTP 1.1, meglio se 2.0.
- *Ogni Web Thing dovrebbe utilizzare connessioni HTTPS*: quando possibile al fine di minimizzare i rischi per le Things accessibili dall'esterno.
- *Ogni Web Thing deve possedere una risorsa root referenziata ad uno specifico URL HTTP*: non vi è necessità che l'URL sia pubblico o meno, potrebbe anche trattarsi semplicemente dell'indirizzo IP del dispositivo in una LAN.
- *Ogni Web Thing deve esporre le proprie proprietà attraverso una struttura gerarchica*: al fine di facilitare il processo di *discovery* della Thing da parte dei client. Uno specifico modello tipo per una Web Thing verrà proposta nelle prossime
- *Le Web Things devono supportare JSON come formato di rappresentazione di default*: questo non proibisce al designer di prevederne anche altri.
- *Le Web Things devono supportare la codifica UTF8 pre richieste e risposte*: questo non proibisce al designer di prevederne anche altri (in caso di descrizioni necessarie in Cinese/Russo, etc.).
- *Le Web Things potrebbero offrire una rappresentazione HTML*: non obbligatoria, ma consigliata in alcuni casi.
- *I server di Web Things devono supportare i verbi GET, POST, PUT, DELETE di HTTP*: con particolare attenzione all'assegnazione con le diverse operazioni CRUD.
- *I server di Web Things devono implementare i codici di errore HTTP 20X, 40X, 50X*: il set di codici consigliati sono: 200 OK, 201 CREATED, 202 ACCEPTED(per chiamate ad operazioni asincrone), 401 UNHAUTORIZED, 404 NOT FOUND, 500 INTERNAL SERVER ERROR, 501 SERVICE UNAVAILABLE.

- *Le Web Things dovrebbero sempre supportare la navigabilità attraverso links*: in particolare si consiglia di porre nella rappresentazione di ogni risorsa i collegamenti ai figli

- *Le Web Things potrebbero supportare l'operazione OPTIONS per ogni risorsa*: per fornire una lista di verbi HTTP supportati dalla singola risorsa.

Scenario Event-Driven

Fino ad ora si è parlato di REST come base del paradigma WoT, ma esso copre davvero tutte le esigenze che la visione di Web of Things possiede? Sicuramente attraverso REST ogni client può, a suo piacimento, interrogare agevolmente le things distribuite nel sistema e scoprire il valore di proprietà, invocare azioni etc., ma essendo WoT un'evoluzione di Internet of Things, bisogno tenere conto anche di tutti i casi d'uso del modello di quest'ultimo. Ci si rriferisce agli scenari Event-Driven di IoT in cui sono le Things, numerose e disseminate nel territorio, ad aggiornare in tempo reale gli agenti del sistema attraverso notifiche *push*.

È necessario estendere il modello request-response di HTTP con un protocollo applicazione *web friendly*: nel suo trattato Guinard suggerisce **WebSocket**. Standardizzato dall'IEFT come RFC 6455, WebSocket è una tecnologia web che fornisce canali di comunicazione full-duplex attraverso una singola connessione TCP, si integra con il protocollo HTTP per via di come effettua l'*handshaking*.

Infatti, per avviare una comunicazione via WebSocket, il client deve inviare una richiesta di *upgrade* al server incapsulata in una GET per una certa risorsa. Da quel momento il server apre e tiene attivo il canale con il client fino a che quest'ultimo non deciderà di chiuderlo. In ottica event-driven, sarà quello il canale utilizzato da una Web Thing per inviare i propri aggiornamenti ad un certo client.

Ricapitolando, quindi, intergrare WebSocket con REST significa mantenere intatto il modello di quest'ultimo, e qualora un client desiderasse ricevere aggiornamenti da una Web Thing per una certa risorsa, gli basterà cominciare l'handshaking per quella specifica risorsa, per poi tenere aperto il canale fino a che servirà.

1.5.2 Descrizione di Web Things

Nel livello dedicato alla *Findability*, come già menzionato, riveste un ruolo cruciale la possibilità di *scoprire* e *comprendere* una qualsiasi Web Thing. Raggiungere questo obiettivo non è semplice, in quanto non può bastare fornire una documentazione scritta delle API della propria Web Thing. Se così fosse, ogni sviluppatore che desideri inserire una Thing nel proprio sistema dovrebbe leggerne la documentazione, comprenderla e implementare le richieste nel modo corretto, in modo toalmente custom.

Per scongiurare lo scenario descritto, è stato formalizzato un *modello unico dei dati* per definire API di Web Things, chiamato **Web Thing Model**.

Web Thing Model

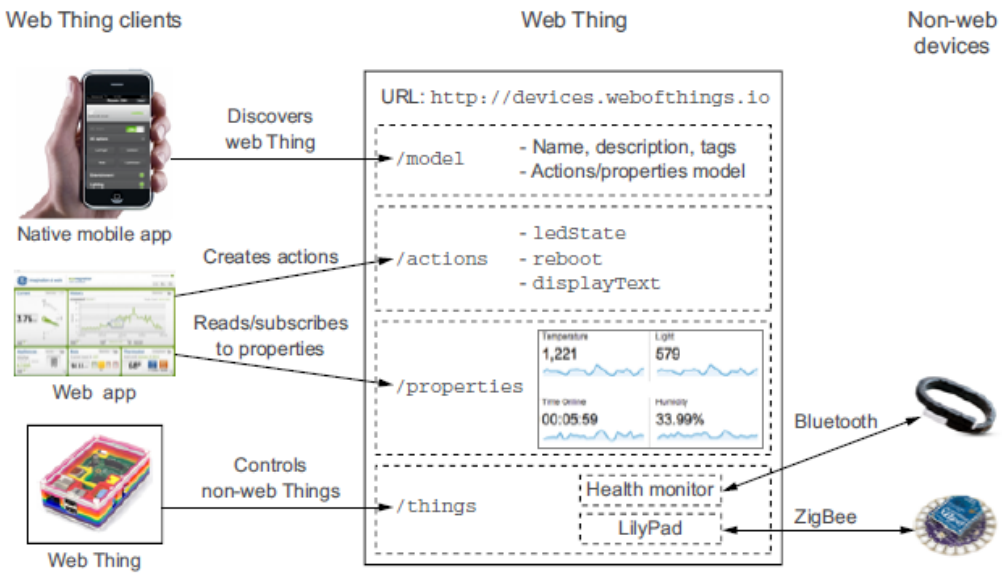


Figura 1.2: Modello di Web Thing

Questo modello descrive una Web Thing(WT) attraverso una serie di concetti ben conosciuti, ed è rappresentato in figura 1.2. Come viene mostrato, ogni Web Thing dovrebbe avere le seguenti risorse:

- *Model*: contiene i metadati della WT, quali nome, descrizione,configurazione, etc.

- *Properties*: le proprietà sono le variabili di una WT (analoghe alle proprietà di una classe) e rappresentano il suo stato interno.
- *Actions*: sono funzioni offerte da una WT, che i Clients possono invocare (esempi di actions possono essere *switchON/switchOFF* di un Led, o *open/close* di una porta).
- *Things*: una WT può essere un gateway per altri dispositivi che non hanno connessione ad internet, questa risorsa contiene tutte le WT a cui la corrente WT fa da proxy.

1.5.3 Rendere sicura la comunicazione tra Web Things

Nelle sezioni precedenti si è fatto riferimento ai rischi, in termini di sicurezza, che corrono i sistemi WoT. L'architettura proposta per WoT colloca nel livello di *Sharing* l'inserimento delle misure atte a mitigare questi rischi. Ad alimentare la necessità di integrare delle solide misure di sicurezza, va considerato che le Web Things sono oggetti fisici sparsi nel mondo fisico, e potenzialmente un attacco ad un sistema WoT potrebbe avere delle conseguenze molto più gravi di altri attacchi informatici a semplici piattaforme web.

Vi sono tre macro problemi da risolvere nella costruzione di un sistema WoT:

- Applicare la **crittografia** nelle comunicazioni tra due entità (per esempio tra una app client e una Web Thing).
- Applicare misure di **autenticazione** tra le parti comunicanti.
- Assicurarsi che sia attivo il **controllo degli accessi** alle risorse.

Per soddisfare *crittografia* e *autenticazione* in WoT si prevede l'utilizzo di **TLS**, un protocollo di livello Applicazione che può essere applicato alle comunicazioni HTTP (HTTPS), WebSocket (WSS) e MQTT(MQTTS).

Per quanto riguarda il *controllo degli accessi*, vi sono delle opzioni: la **token-based authentication** (un token unico per ogni client utilizzato per per autenticare ogni propria richiesta) o **OAuth**(il processo di autorizzazione viene delegato a servici di terze parti affidabili, quali Facebook, Google, etc.)

Capitolo 2

Augmented Reality

2.1 Cosa si intende per Realtà Aumentata

Come primo passo importante, bisogna cercare di definire al meglio il concetto di Realtà Aumentata, e per fare ciò è importante distinguere quali sono le sue caratteristiche e quali no.

Alan B. Craig si focalizza sul concetto di vedere la AR come un' *esperienza* compiuta da un utente in un certo ambito (sanità, industria, etc.). In questa esperienza la persona continua a iteragire/manipolare il mondo fisico che lo circonda, avendo però in aggiunta informazioni virtuali, sensibili e interattive con le stesse modalità di interazione che si hanno nella realtà. Un esempio di esperienza di AR potrebbe essere quella di riuscire a vedere ed esplorare uno spazio appena progettato dall'utente, senza che questo sia stato effettivamente ancora realizzato. Con *Aumentata* quindi, si intende un estensione della realtà, potenziando di conseguenza sensi e pensiero di chi la sperimenta.

Si può provare quindi a determinare le caratteristiche dell'AR come rifinizione di quelle citate da Ronald T. Azuna nel 1997 (combinazione di reale e virtuale, interattiva in tempo reale, registrata e percepita in 3D).

- **Mondo fisico aumentato con informazioni digitali *inserite/superimposte* nella vista del mondo reale**. Il punto chiave dell'aggiunta/superimposizione delle informazione digitali nel mondo fisico è il fatto di non essere interessati ad estraniare l'utente dal mondo reale, anzi il contrario. Più l'utente è immerso nel mondo fisico e opera seguendo le leggi che esso impone, nonostante questo sia Aumentato e quindi arricchito di informazioni digitali, meglio è. Ad oggi però non è possibile pensare alla AR senza coinvolgere apparecchiature hardware

di supporto, quali occhiali, auricolari e simili, che sicuramente minano il principio precedentemente descritto. In questo caso si ritiene "valide" come portatrici di realtà virtuali tutte quelle apparecchiature che non occludono i sensi dell'utente ma semplicemente li estendono (ad esempio occhiali trasparenti con cui è possibile vedere attraverso o auricolari che non schermano i suoni/rumori esterni).

- **Le informazioni sono inserite/usate nel mondo fisico con riferimenti spaziali(ed eventualmente temporali)**. Uno dei principali punti di forza dell'AR sta nel poter creare digitalmente qualsiasi tipo di informazione,compresi gli oggetti. Appurato che tramite l'apparecchiatura adeguata è possibile vedere questi oggetti, è necessario rendere questi realistici nel modo più efficace possibile.Da questo deriva la necessità per un oggetto aumentato di essere *registrato spazialmente*, ovvero possedere delle coordinate nelle quali esistere, e far si che tutti gli utenti lo possano vedere in maniera coerente con la propria posizione(per esempio non può essere visto se dietro a un muro, oppure avere una vista differente da sopra/di lato). Possiamo distinguere 2 tipi di registrazione spaziale:

  - **Assoluta**: ovvero l'oggetto virtuale possiede delle coordinate assolute di posizionamento. Per esempio in una applicazione che fa vedere in un cantiere un edificio già costruito.
  - **Relativa**: ovvero la locazione dell'oggetto è relativa alla posizione un altro oggetto fisico/virtuale. Per esempio in un'applicazione in cui si possono vedere i circuiti di una calcolatrice quando la si guarda.

  Per quanto riguarda invece la registrazione temporale, essa è più difficile da realizzare a causa dei lags dovuti dal processo delle informazioni. Questo problema ha dei *workarounds* possibili per evitare che ad esempio gli spostamenti di un oggetto avvengano con del lag, ma non potrà mai essere debellato del tutto.

- **Le informazioni mostrate sono dipendenti dalla locazione e dalla prospettiva di chi le guarda**. Come accennato nei punti precedenti, dare ad un oggetto aumentato una posizione si combina con la possibilità di un utente di vedere la realtà aumentata dal suo personale punto di vista. Questo è un aspetto chiave al fine di rendere le informazioni digitali aggiunte alla realtà osservabili nella stessa maniera in cui si osserva il mondo fisico.

- **L'esperienza deve essere interattiva**. L'utente immerso nella realtà virtuale percepisce le informazioni virtuali, e a suo piacimento può mu-

tarle o crearne di nuove, magari facendo uso di altre informazioni della stessa natura.

2.1.1 Il ruolo attuale della Realtà Aumentata

Ad oggi il settore della Realtà Aumentata è in continua crescita ed evoluzione, sia per quanto riguarda le applicazioni, le tecnologie abilitanti e lo sviluppo. La rosa di ambiti in cui questa tecnologia si è scoperto poter essere applicata, e le conseguenti opportunità di mercato, hanno favorito questa crescita.

Esempi di aree di applicazione della AR verranno descritte nella prossima sezione, per dare però uno spaccato di quelle che sono le principali direzioni che gli addetti ai lavori stanno percorrendo si citano i seguenti ambiti:

- Scientifico, Ingegneristico e Pubblico.
- Commerciale e di Impresa.
- Consumatori, Marketing e Pubblicità.

Certamente in alcuni settori la sperimentazione è attualmente ad uno stato avanzato (con prototipi di applicazioni quasi completi se non già utilizzati), in altri si è ad uno stato ancora embrionale, ma non si può mettere in dubbio quanto la AR si prospetta di essere la tecnologia chiave del prossimo futuro. È con questa consapevolezza che gli attuali sviluppatori e ingegneri in ambito IT non possono più permettersi di escluderla o ignorarla, per il grado di pervasività che possiede.

2.2 Aree di applicazione della AR

Potenzialmente la Realtà Aumentata, così come la si è definita, ha un'illimitata gamma di campi di applicazione. Dato che al momento è una tecnologia ancora in evoluzione, in questa sezione verranno menzionati alcuni di questi, tra quelli in via più avanzata di utilizzo della tecnologia.

Progettazione e modellazione

Uno dei punti di forza della Realtà Aumentata che sta venendo sfruttato da chi sviluppa in ambito di sistemi per la progettazione di edifici o macchinari di qualsiasi tipo, è la possibilità di poter visualizzare questi progetti prima che questi vengano costruiti fisicamente. Prima dell'avvento di queste tecnologie gli

ingegneri e designer potevano certamente realizzare progetti virtuali con programmi di tipo CAD (Computer Aided Design) o BIM (Building Information Modeling), ma questi rimanevano sullo schermo del loro pc/tablet.

Grazie alle possibilità offerte dall'AR oggi è possibile, tramite l'aiuto di visori o direttamente smartphone o tablet, vedere ed esplorare il progetto che si sta realizzando come se esso fosse "tra le loro mani", in 3D.

Chi si sta muovendo in questa direzione è, per esempio, **Autodesk**, che con il progetto "Fusion 360" ha abbracciato il visore di Microsoft "Hololens" al fine di consentire agli utilizzatori della piattaforma di poter progettare attraverso la realtà aumentata, interagendo con il progetto invece che tramite uno schermo, un mouse o una tastiera, come se questo fosse un oggetto reale, ovvero attraverso per esempio gestures delle mani.

Musei

Negli ultimi anni anche in ambito culturale storico sono stati mossi dei passi in direzione della realtà aumentata per molteplici ragioni, tra le quali rendere, banalmente, l'esperienza in un museo più accattivante e attrarre visitatori, oppure più semplicemente sfruttare la caratteristica principale dell'AR, che consinste nel aggiungere informazioni al mondo fisico, per *completare* una visita con più informazioni possibili riguardo quello che un visitatore sta osservando in un certo momento.

Si può affermare che il ruolo principale, quindi,del coinvolgimento della Realtà Aumentata possa essere quello di *intrattenere ed educare* nel caso dell'utilizzo nel campo della divulgazione culturale.Un esempio di possibile utilizzo concreto potrebbe essere:

- la sostituzione delle etichette cartacee con etichette virtuali interattive, che mostrino i collegamenti di una certa opera con altre opere/concetti/movimenti/avvenimenti storici

- in caso di spazi aperti la collocazione di animazioni coerenti con lo spazio che si sta osservando (per mostrare ricorstruzioni di eventi storici, creature estinte, etc.)

- l'introduzione di guide audio intelligenti che si attivino in base a cosa il visitatore sta osservando

Esempi di applicazioni reale possiamo ritrovarli nei progetti "Augmenting Rothko" della Harvard University, "Van Gogh Re-created", e nella chiesa di "Sant Climent de Taüll", dove opere d'arte visive vengono parzialmente ricreate digitalmente per permettere ai visitatori di apprezzarle nella loro interezza.

Vi sono altri campi in fermento per l'integrazione della Realtà Aumentata, tra questi si può citare l'Healthcare, le Neuroscienze, l'Industria, il settore dei veicoli quali Automobili o Aerei, etc.

2.3 Reality-Virtuality Continuum

Per quanto attuale, la ricerca studia la Realtà Aumentata da diverso tempo. P. Milgram, nel 1994, espone in un suo trattato una definizione di AR, basata sul concetto di **Reality-Vituality Continuum** (RV Continuum).

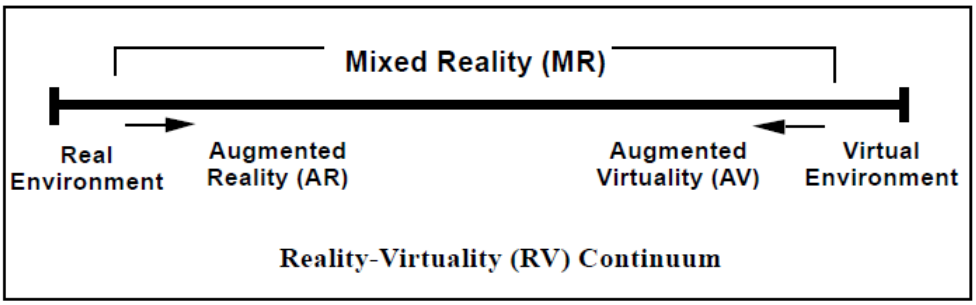


Figura 2.1: Rappresentazione semplificata del RV Continuum

La critica mossa da Milgram partiva dal fatto che antecedentemente a lui, nessuno avesse mai cercato di formalizzare e classificare i diversi tipi di *Mixed Reality*(MR, realtà con componenti fisiche e virtuali presentate insieme attraverso un unico "display"). Il RV Continuum venne concepito attraverso il ragionamento sulle seguenti domande:

- Quale relazione intercorresse tra Realtà Aumentata (AR) e Realtà Virtuale (VR).
- Se il concetto di AR fosse associato soltanto al caso di utilizzo di display *see-through head-mounted* (STHM, intesi come i moderni smart glasses, capaci di renderizzare ologrammi all'interno del mondo fisico osservato attraverso le lenti).

Prima di affrontarle, dunque, si descrive il RV Continuum (figura 2.1), denotandone gli estremi:

- *Real Environment*: ambiente costituito esclusivamente da oggetti fisici, comprende scenari di osservazione in prima persona (persona immesa nel mondo fisico) o attravero una qualche "finestra" (schermo, occhiali, etc.).
- *Virtual Environment*: ambiente costituito esclusivamente di oggetti virtuali, comprende simulazioni di Computer Graphics sia immersive che attraverso un monitor.

Tutto ciò che si pone all'interno di questi limiti unisce insieme(a rigor di logica) parti virtuali e parti fisiche, ed è ciò che più genericamente viene definito *Mixed Reality*.

Mostrato e spiegato il continuum di Milgram, è semplice dare una risposta alla prima delle sue due domande, affermando che la sostanziale differenza tra AR e VR risiede nella totale immersione (caso della VR) dell'osservatore in un mondo sintetico, senza nessuna interazione con oggetti fisici reali.

In figura si nota come avvicinandosi all'estremo reale, ci si riferisca alla *Augmented Reality*, mentre avvicinandosi all'estremo virtuale ad *Augmented Virtuality*(AV). La sostanziale differenza in questo caso, riguarda il cosidetto *substratum*, ossia il mondo sottostante in cui l'osservatore si immerge e sperimenta la MR. Con un substratum di tipo reale Milgram colloca e al tempo stesso definisce la AR, mentre ad un substratum di tipo virtuale pone la AV, ipotizzando uno scenario in cui un osservatore, immerso in un mondo simulato, interagisca in qualche modo anche con una componente fisica.

Definita la AR, si può così dare una risposta alla seconda domanda posta da Milgram, sostenendo che: avendo la AR un sottostrato reale, possono essere compresi sia scenari di osservazione con display STHM, sia l'utilizzo di monitor come "finestra" sul mondo reale.

In conclusione, applicando la tesi sul VR Continuum al giorno d'oggi, si asserisce che con AR si intende proprio la stessa concezione di Milgram, ovvero la collocazione di entità virtuali ad aumentare il mondo fisico (parte sinistra della MR in figura 2.1). Per motivi di avanzamento tecnologico, si viaggia in direzione dell'utilizzo esclusivo di dispositivi STHM, per agevolare la libertà di movimento e visione dell'osservatore e rendere l'esperienza quanto più realistica possibile.

2.4 Tecniche di localizzazione per AR

Come già menzionato, la caratteristica chiave della AR è l'aumento della realtà fisica con delle entità virtuali associate, spesso e volentieri, a degli ologrammi, disegnati (renderizzati) all'interno del mondo fisico dal dispositivo che l'osservatore utilizza. Questi ologrammi, con il movimento dell'utente nel mondo reale, dovrebbero continuamente rimanere *coerenti* con il suo punto di vista, attraverso l' aggiornamento in tempo reale della posizione del device.

In questa sezione si procede a descrivere, ad un livello superficiale, le diverse tecniche di localizzazione utilizzate in ambito AR. Prima di trattarle è necessario specificare che le applicazioni di AR basano la propria localizzazione sulla posizione della videocamera, accuratamente calcolata nello spazio tridimensionale, chiamata anche "pose", o "six-degrees of freedom"(6DOF). 6DOF (figura 2.2) descrive la posizione nello spazio 3D attraverso *forward/backward*, *up/down* e *right/left*, combinata alla orientazione rappresentata dal grado di rotazione sui tre assi perpendicolari, detti *pitch, yaw* e *roll*.

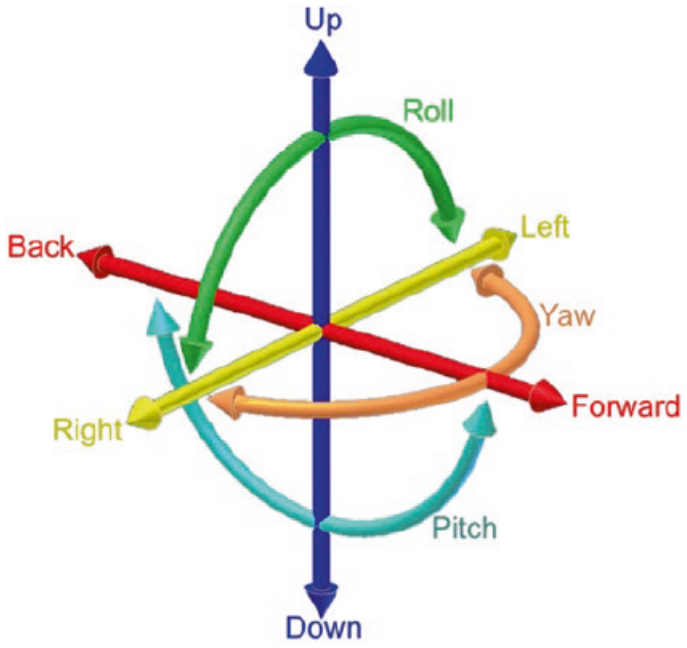


Figura 2.2: Rappresentazione di Six-degrees of freedom

Le tecniche principali utilizzate per la definizione in tempo reale della pose sono **tecniche ottiche**, e si possono classificare in **marker-based** e **markerless**:

2.4.1 Tecniche Marker-based

Tecnica caratterizzata dalla presenza nell'ambiente di "markers" (figura 2.3), ovvero delle figure geometriche di forma quadrata con pattern neri e bianchi che codificano l'informazione 3D che deve essere posta in loro corrispondenza. La funzione di localizzazione utilizza la videocamera per dedurre posizione e orientazione dell'osservatore in base a cosa essa "vede" in tempo reale. Questo è possibile perchè, per la forma e i pattern predefiniti dei markers, sono facilmente distinguibili (contrasto tra quadrati bianchi e neri) e la posizione è calcolabile data l'angolazione con cui i punti del marker sono visualizzati a video (attraverso trasformazioni geometriche). Questo tipo di localizzazione aggiunge significante robustezza in condizioni di poca luminosità o in presenza di marker anche lontani. Esistono anche altri tipi di marker oltre a quelli appena descritti, detti "visibili", come ad esempio quelli "attivi"(per esempio led infrarossi intermittenti) e "passivi" (retroriflettori)

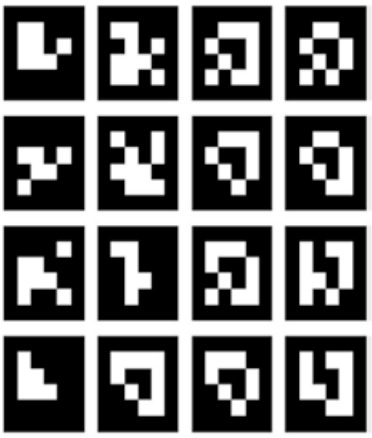


Figura 2.3: Esempio di marker utilizzato nella tecnica marker-based

2.4.2 Tecniche Markerless

Tecnica basata sul riconoscimento di *punti fiduciali* (punti di interesse) nell'ambiente che si osserva, per questa operazioni esistono due categorie di rilevatori: *Corner detectors* e *Blob detectors*. I primi, come il nome suggerisce, consentono di individuare la posizione di strutture angolari (importanti perchè

denotano la potenziale presenza di strutture/elementi nel paesaggio), i secondi individuano particolari regioni contraddistinte da proprietà non omogenee rispetto all'area che circonda la regione stessa (per esempio regioni circolari scure su sfondo chiaro o viceversa). Una volta definiti i punti fiduciali, esistono algoritmi di diversa natura per la ricostruzione della 6DOF dell'utente. Questo approccio di localizzazione risulta più oneroso (rispetto allo scenario marker-based) in termini di computazione e necessita di un precaricamento di dati, ma ha il vantaggio di non dover inquinare il paesaggio di marker e di poter essere utilizzato in ambienti outdoor. Capita, in queste tecniche, di comprendere dati derivati da GPS, accelerometro, giroscopio e altra sensoristica, per aumentarne l'affidabilità.

Capitolo 3

Web of Augmented Things

3.1 Visione di Web of Augmented Things

Web of Things, come spiegato nel primo capitolo, nasce dalla problematica emergente in ambito IoT di rendere tutte le Things fisiche (connesse in rete) interoperabili e fruibili da qualsiasi sistema ne implementi lo stack. In diversi settori l'adozione o integrazione di sistemi IoT è in crescita (si fa riferimento ad esempio all'industria 4.0, ai progetti di Smart Cities, etc.), ha senso dunque immaginare uno scenario in cui, nel prossimo futuro, si assisterà ad una crescita della pervasività delle Smart Things, e alla realizzazione delle più diverse applicazioni che le utilizzino.

D'altro canto, anche la Realtà Aumentata è un settore in forte sviluppo in diversi settori, molti dei quali in comune con IoT. Pensare che queste due visioni un giorno possano doversi integrare per necessità è del tutto lecito, e uno scenario ipotetico di questa unione sarebbe quello di un utente, immerso nel mondo reale, capace di interagire sia con Things fisiche che lo circondano, sia con delle *Things Aumentate* visualizzate come ologrammi ed esistenti come del software operante su una qualche macchina in remoto. Queste Things aumentate, esattamente come quelle fisiche, oltre ad una rappresentazione (ologramma) possono avere anche una certa interfaccia di interazione, un comportamento interno e un determinato stato.

Il primo grande scoglio da superare per realizzare questa idea è trovare un modello che accomuni entità fisiche ed entità aumentate e permetta di lavorare su entrambe facendo trasparire questa eterogeneità. Recentemente, in letteratura, è stato proposto il modello di **Augmented World**, ovvero una

classe di sistemi in cui i livelli fisici e virtuali vengono "profondamente connessi e integrati".[1]

3.2 Augmented Worlds

Alla luce di quanto trattato nel capitolo precedente, i recenti sviluppi in ambito di Realtà Aumentata hanno aperto a ricercatori e alle aziende un ampio spettro di applicazioni possibili. A rendere questo ancora più stimolante vi sono le nuove realizzazioni in ambito di *wearable computing*, che comprendono tutti quei sistemi embedded/mobile che un utente umano può indossare. Affiancato al concetto di AR si può porre quello di *Mixed Reality* (MR) che, differentemente dal primo che è un medium tra il mondo fisico e delle informazioni virtuali aggiunte ad esso, rappresenta l'unione dei due mondi (fisico e virtuale) al fine di produrre un nuovo macroambiente in cui oggetti reali e digitali coesistono e interagiscono tra loro in tempo reale.

Proprio in questo frangente entra in gioco il concetto di Augmented World, definito come "*una classe di applicazioni atte a ottenere la Mixed Reality attraverso l'utilizzo delle esistenti tecnologie di Augmentation*". Un utente immerso in un AW è capace di aumentare i proprio sensi e la propria esperienza interagendo con tutte le entità visibili della mixed reality, e può essere affiancato ad entità proattive che lo guidano/supportano in tutte le attività che egli compie.

L'esigenza di coinvolgere gli *agenti* come paradigma in questi Mondi Aumentati nasce dalla necessità della presenza, in un livello superiore che si può definire *aumentato*, di entità proattive che svolgono il compito di osservare ed elaborare dati e fornire indicazioni a utenti/cose nell'ambiente.

Gli Agenti nell'Augmented World

Per la concettualizzazione di AW i ricercatori hanno scelto di coinvolgere il paradigma ad Agenti nel modello in quanto esso presenta le seguenti caratteristiche:

- Gli agenti sono entità *proattive*.
- Ogni agente possiede un suo flusso di controllo e quindi può operare in autonomia secondo la sua programmazione.

[1] Modello proposto dal DISI (Dipartimento Ingegneria e Scienze Informatiche) dell'Università di Bologna

- Tra loro più agenti possono comunicare e apprendere nel corso della loro computazione, modificando anche il proprio comportamento al fine di compiere al meglio i propri Tasks.

Nell'ottica quindi di realizzare Mondi Aumentati, gli agenti sono orientati all'*assistenza*, e risiedono in uno stato superiore chiamato *Augmented Layer* (AL) (figura 3.1).

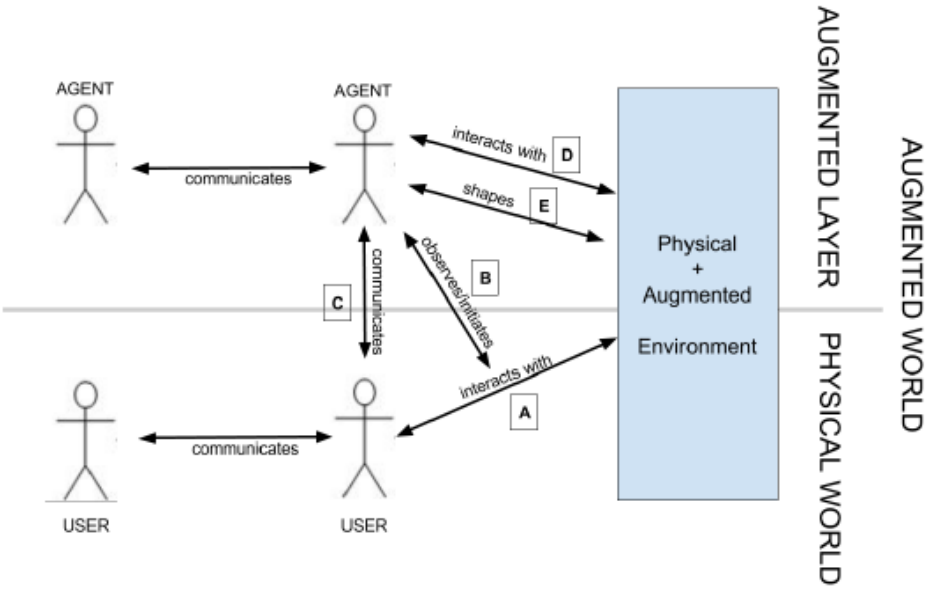


Figura 3.1: Elementi di un Augmented World basato su agenti

Gli agenti che abitano il Livello Aumentato possono, da modello, avere 2 diverse conformazioni:

- agenti che aumentano l'utente (girano su device da lui posseduti)
- agenti che aumentano l'ambiente (girano sull'infrastruttura dell'ambiente intelligente)

Per quanto riguarda gli agenti che aumentano l'utente si pensa a *Software Personal Assistant agents* (SPAs), ovvero agenti capaci di comunicare con l'utente ed essergli utili secondo le sue personali necessità (in base al loro scopo). Possono assistere l'utente durante le sue attività in molteplici maniere, ed è

possibile coinvolgere il concetto di *"I see what you see"*, ovvero l'agente riesce a "vedere", attraverso dell'hardware adeguato, quello che sta vedendo la persona con cui interagisce. Computando nell'Augmented Layer questi agenti possono comunicare con l'altra tipologia di agenti, descritta in seguito, o direttamente con lo Smart Environment al fine di effettuare i propri tasks.

La seconda tipologia di agente può anche essere vista come *building block* per tutte quelle applicazioni che vogliono seguire il concetto di Mixed Reality e Augmented World. Si ricorda infatti che la MR è una unione di entità fisiche e virtuali, e proprio queste ultime per necessità devono emulare le prime nel possedere una propria autonomia e proattività. Questi agenti in definitiva fanno parte dell'environment e possono avere una propria rappresentazione aumentata (ad esempio un ologramma) processabile da un device dell'utente.

Avvicinandosi alla definizione di un modello, si considera necessario citare il concetto di *Mirror Worlds*, associato a quello di MR, come caso particolare di Augmented World.

Mirror Worlds

Il concetto di Mirror World (MW) è semplice di per se, in quanto riconducibile a quello di Mixed Reality. Con MW si intende un mondo virtuale (che risiede su una qualsiasi piattaforma computazionale) che fa da specchio al mondo fisico. Persone e oggetti reali sono mappati in questo mondo specchio, e tutta l'attrezzatura hardware embedded montata su questi ultimi consente ad essi di poter inviare informazioni sul loro stato, in modo che la loro rappresentazione nel MW venga tenuta aggiornata.

La vera potenza dei MW sta nel riconoscere e sfruttare la possibilità di inserire nel mondo specchio anche entità che nel mondo fisico non sono presenti, quali agenti rappresentanti assistenti degli utenti, o oggetti virtuali quali monitor, etc. Queste entità puramente virtuali hanno la medesima rappresentazione di quelle reali (nello strato del MW), e possiedono opzionalmente delle rappresentazioni compatibili con i dispositivi wearable che l'utente sta indossando al fine di poter essere viste/percepite.

L'esempio proposto da [6] è quello del gioco *Ghosts in the City*, in cui lo scopo è, per l'utente, quello di scappare cercare tesori virtuali in un ambiente outdoor, scappando nel frattempo dai fantasmi che si aggirano nella città, oppure uccidendoli con la propria bacchetta magica (lo smartphone). Portando questo esempio nel MW, i singoli fantasmi divengono degli agenti nello strato

specchio, e possono essere visti dai giocatori attraverso *smart glasses*. Ogni fantasma può avere delle peculiarità differenti che ne influenzano il comportamento, per esempio un fantasma può prediligere i posti più illuminati, o umidi, e lo fa in base alle informazioni che arrivano dai sensori sparsi nell'ambiente e mappati ed osservabili nel MW. Anche il giocatore ha la sua rappresentazione nel MW, che tiene traccia della sua localizzazione, ed altre informazioni utili ai fini del gioco.

Per riuscire a implementare un ipotetico Mirror World, è importante individuare lo stack di tecnologie abilitanti. Si suppone di dividerlo in 3 livelli, mostrati in figura 3.2.

3.2.1 Modello concettuale

A questo punto è stato inquadrato abbastanza precisamente verso dove si vuole andare con il concetto di Augmented World. Nella sezione seguente sarà presentato uno dei modelli sviluppati dai ricercatori, che sottintende tutto quello che è stato menzionato precedentemente e lo formalizza. Prima di parlare del modello di AW verrà brevemente spiegato anche il concetto di Mirror World, dal quale esso si ispira fortemente.

Possiamo notare come Web of Things, trattato in precedenza, possa essere un middleware molto efficace per la realizzazione di MWs.

Formalizzazione del modello di AW

Tornando al modello di AW, al fine di aumentare la realtà è stato scelto di rappresentare entità reali e virtuali attraverso un comune oggetto computazionale: le *Augmented Entities* (AE). Queste AEs hanno una locazione nello spazio e sono interagibili e accessibili da parte degli agenti del sistema. Proprio come per i MW, le AE possono avere una rappresentazione reale compatibile con i devices dell'utente (si parla quindi di ologrammi per entità visibili, ad esempio),oppure possono essere lo specchio di una entità/oggetto fisico nel mondo reale. Anche i singoli utenti che si muovono e agiscono nel mondo fisico possono avere una AE che fa loro da specchio, chiaramente più questi utenti sono monitorati da devices che raccolgono informazioni su di loro, più gli agenti del sistema possono sfruttarre le entità ad essi associati per operare (avendo queste diverse informazioni sullo stato del singolo utente).

In termini di osservabilità, si intende dividere le varie AEs in più *regioni*, in modo tale che queste non siano accessibili da tutti gli agenti/AEs del sistema, ma solo da quelli presenti effettivamente nelle loro vicinanze (fisiche o virtuali).

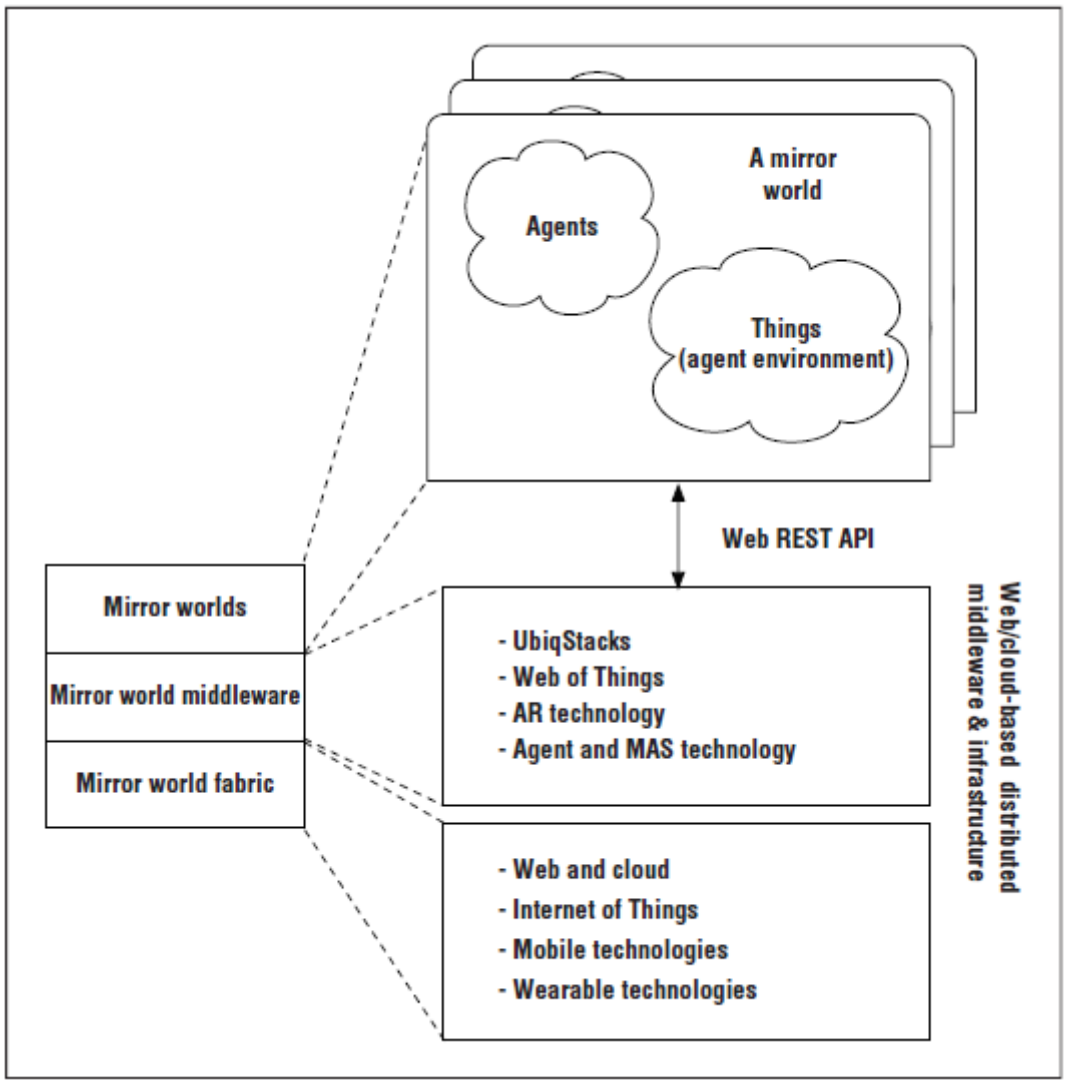


Figura 3.2: Stack di tecnologie abilitanti per Mirror Worlds

In figura 3.3 si può osservare quello che è il modello proposto dai ricercatori per l'AW. L'architettura è suddivisa in 2 macrolivelli: *Physical World* e *Digital World*.

Per quanto riguarda il primo livello non vi è molto da dire, si può notare che sono presenti delle regioni delimitate, che saranno replicate nel livello digitale. Inoltre gli ologrammi delle entità puramente virtuali vengono proiettati in questo livello di mondo fisico in modo da poter essere visibili e interagibili dagli utenti.

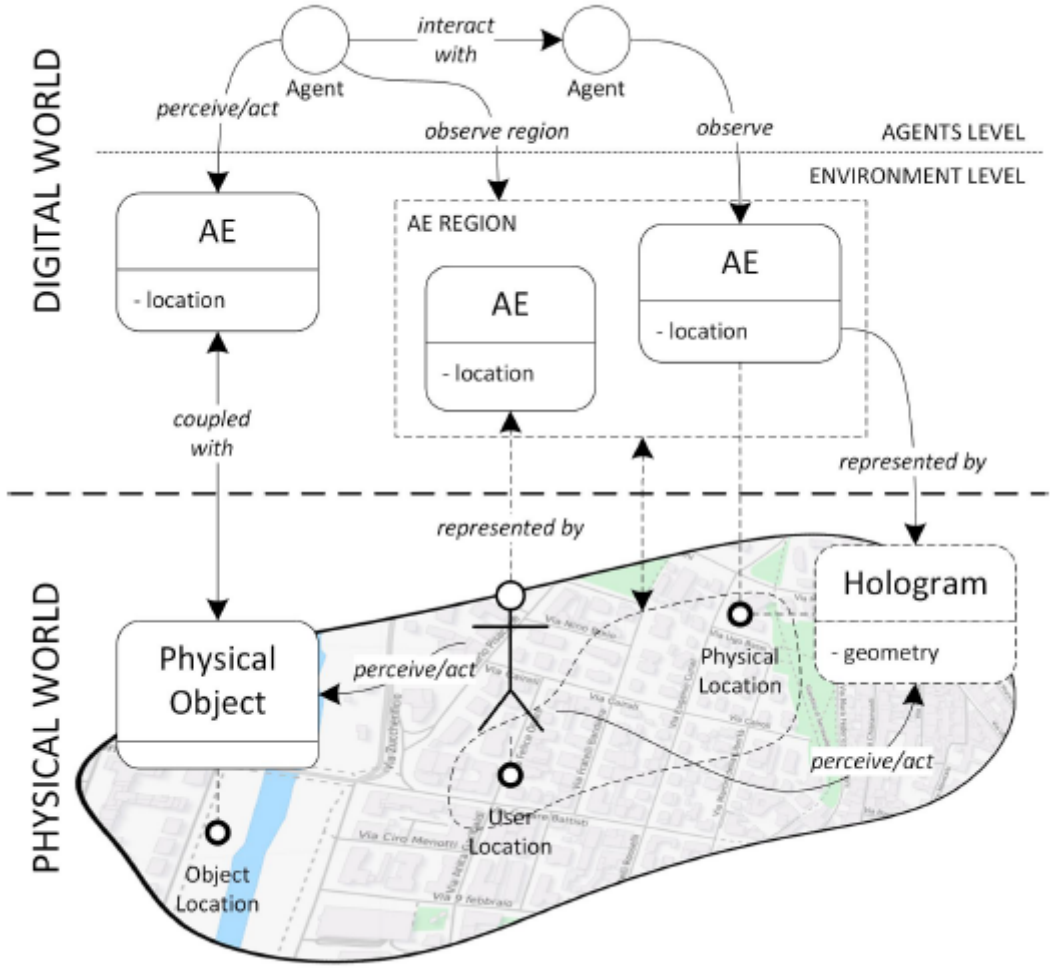


Figura 3.3: Modello concettuale di un Augmented World

Per quanto riguarda invece il livello superiore, il Digital World, esso è diviso in ulteriori 2 livelli:

- **Agents Level**: ove risiedono gli agenti che osservano le varie AEs e interagiscono tra di loro
- **Environment Level**: ove risiedono le AEs, eventualmente raggruppate in *AE Regions*

Al fine di delineare ancora più chiaramente le caratteristiche del concetto introdotto di Augmented Entitities, si definiscono in seguito le caratteristiche

che esse devono avere:

- Ogni AE deve avere un riferimento spaziale per sapere dove questa è locata nel mondo fisico (esprimibile con attributi *location*). La locazione può variare nel tempo e può dipendere dallo stato computazionale dell'entità.

- La *rintracciabilità* e l'*osservabilità* di una AE è legata alla sua locazione. Un agente che vuole accedere una entità deve effettuare una operazione di *lookup* nella regione che sta visitando, e non deve poter vedere oltre ad essa. Una AE può anche avere associati dei riferimenti al proprio *raggio* di osservabilità e osservazione, e divenire localizzabile solo nel momento in cui diventa osservabile.

- Gli oggetti fisici che vengono mappati con delle AEs montano dei sistemi mobile o embedded in modo da raccogliere informazioni sul proprio stato. Questi dati verranno poi raccolti dalle loro AEs e forniti agli agenti/AEs esterne con cui interagiscono.

- Come già specificato, gli utenti possono avere il proprio *corpo aumentato* rappresentato da una entità nell'Environment Level.

Data la natura del livello Digital World, è possibile ricondurre i concetti di agenti e AEs a quello che in letteratura è chiamato "*Agents & Artifacts*", in cui le Augmented Entities sono Artifacs che possono essere creati, osservati e utilizzati, mentre gli agenti rimangono agenti e popolano il livello superiore del Digital World. Utilizzando queste analogie si possono prototipare AW su piattaforme esistenti, ad esempio JaCaMo, un framework per la programmazione di sistemi multiagente.

Relazioni tra i componenti

In figura 3.4 è possibile vedere come siano state formalizzate le relazioni tra i componenti del modello di un AW (attraverso un diagramma delle classi UML).

Un Augmented World è una composizione di varie AEs, per le quali si definisce un nome, una posizione(come già trattato precedentemente), un'orientazione e delle azioni, e sono definite in relazione alle Regioni presenti, anch'esse facenti parte dell'AW. Un'entità aumentata può possedere un ologramma, in modo tale da poter essere processata dalle applicazioni utente montate sui singoli devices delle persone, al fine di rendere possibile l'interazione con esse nel

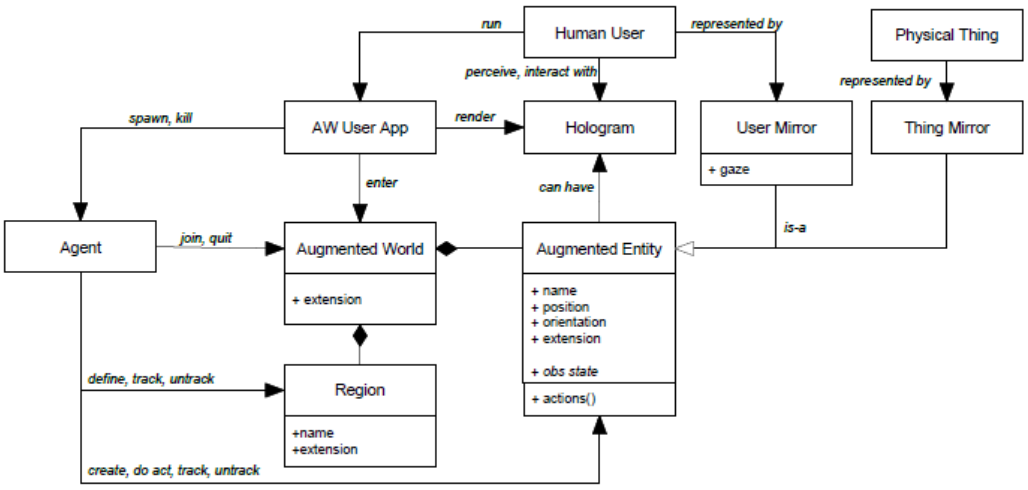


Figura 3.4: Relazioni tra i componenti del modello di AW

mondo fisico. Le applicazioni utente non fanno altro che creare e gestire una serie di agenti autonomi capaci di entrare (fare *join*)in un AW, monitorare una certa Regione e creare/distruggere/osservare AEs secondo i loro bisogni. Per quanto riguarda invece gli oggetti Fisici e gli Utenti stessi, vengono rappresentati virtualmente attraverso *Thing/User Mirrors*, che altro non sono che altre AEs (nel caso degli utenti, acquisiscono eventualmente anche la proprietà *gaze*, che indica il loro angolo di visuale).

3.3 Infrastruttura per Web of Augmented Things

Delineato AW come modello di riferimento per lavorare con Things di natura fisica ed aumentata e specificatone struttura ed interazioni, è necessario rendere le *AEs* di AW:

- Accessibili
- Interoperabili

Per farlo, è necessario sviluppare un protocollo di livello applicazione da porre alla base del livello AW del tutto analogo a WoT (in grado di risolvere le necessità elencate), chiamato **Web of Augmented Things**.

In termini più pratici, il progetto da realizzare in questa tesi sarà quello di ingegnerizzare un layer WoAT da affiancare ad una implementazione di AW,

che permetta di accedere e interagire con tutte le entità del sistema (siano esse *mirror* di things fisiche, oppure aumentate) come fossero Web Things, e mappi in chiave REST tutte le API previste dal modello di AW.

Capitolo 4

Progettazione dell'Infrastruttura per WoAT

4.1 Requisiti e problematiche emergenti

In questa sezione si passa in rassegna a tutti quelli che sono i requisiti per lo sviluppo di un'infrastruttura di Web of Augmented Things. Una volta delineato *cosa*, nello specifico, questo strato debba fare e quali features debba fornire, si andrà alla ricerca delle problematiche che emergono dai requisiti.

4.1.1 Aspetti funzionali

Ponendo l'Augmented World come modello di riferimento per WoAT, emerge il primo requisito per l'infrastruttura: tradurre le interazioni del modello di AW e gli aspetti strutturali in ambito WoT.

Estrapolando dal modello le varie azioni che utenti e agenti possono compiere su un AW, al fine di chiarirne meglio il funzionamento, si generato il diagramma dei casi d'uso in figura 4.1.

Come già specificato l'Augmented World dovrà poter essere acceduto dall'esterno, attraverso WoAT, da parte di agenti o semplicemente dal software attivo sui devices degli utenti (con agente, per chiarezza, si intende un'entità computazionale intelligente autonoma in grado di percepire l'ambiente che lo circonda e di reagire a certi eventi eseguendo delle azioni). Le API che permettono le funzionalità base di un AW, estrapolate anche dai casi d'uso, sono descritte in Tabella 4.1 (join nell'AW, creazione di AEs, join di Regioni, tracking di Regioni/Aes, etc.).

Primitive Augmented World	**Descrizione**
joinAW(awName, credentials):sessionID	Per fare join in un determinato AW di un certo nome in una certa zona, in restituzione l'ID di sessione
quitAW(awName, sessionID)	Per uscire dall'AW
createAE(awName, name, template, args,config): aeID	Per creare una entità aumentata in un certo AW, specificandone nome, template e parametri (associati al template), in restituzione l'ID dell'entità creata
disposeAE(aeID)	Per eliminare una AE esistente
trackAE(aeID)	Per cominciare a osservare una AE
stopTrackingAE(aeID)	Per smettere di osservare una certa AE
moveAE(aeID, pos, orientation)	Per cambiare posizione e orientamento di una entità, se consentito
defineRegion(awName, name, region)	Per definire una nuova regione, specificandone nome ed estensione
trackRegion(awName, name)	Per iniziare ad osservare una regione
stopTrackingRegion(awName, name)	Per smettere di osservare una regione

Tabella 4.1: Set di API per le funzionalità base di un AW

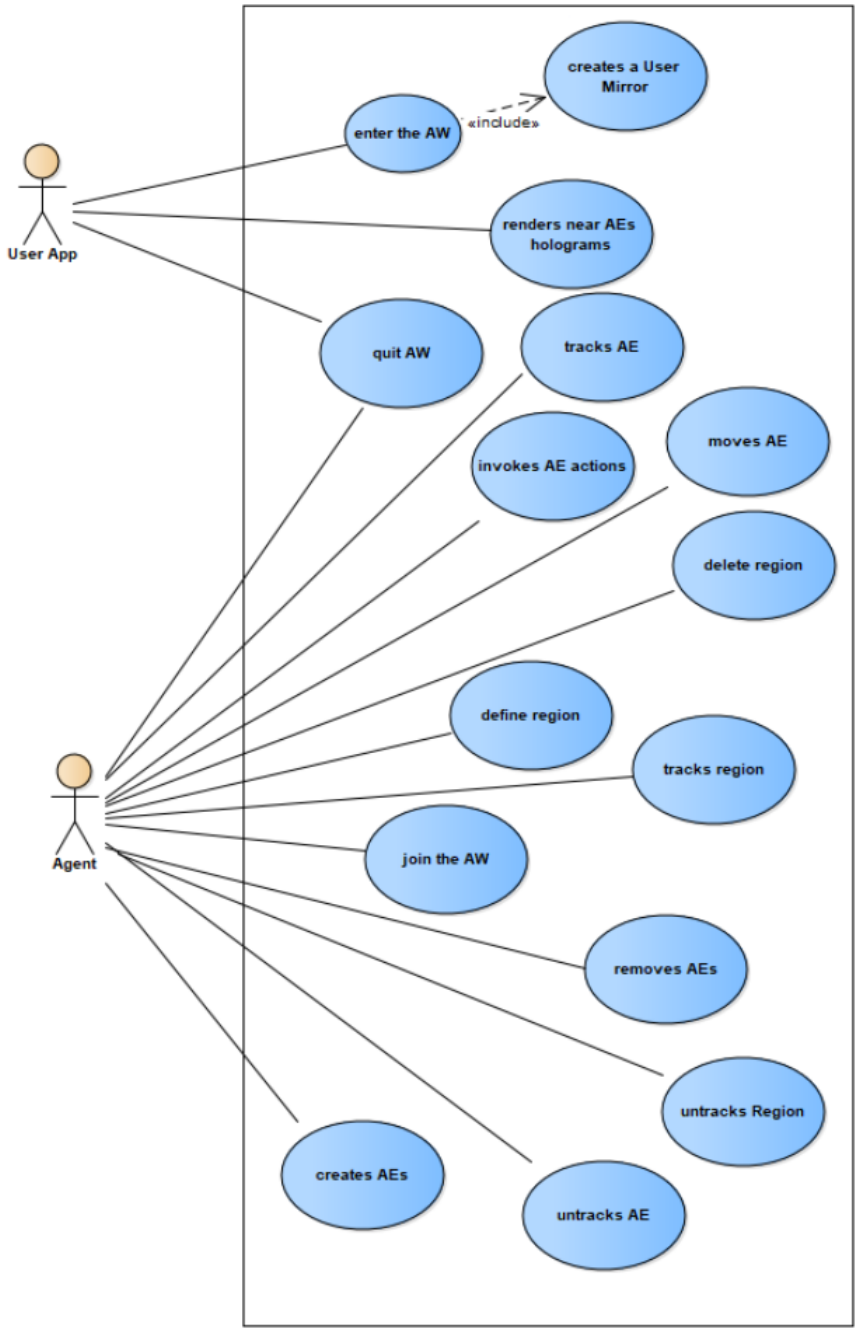


Figura 4.1: Diagramma dei casi d'uso del modello di AW

Oltre ad esse vi sono da considerare tutte le API che consentono l'osservazione di *proprietà* delle singole AEs e le chiamate alle *azioni* di quest'ultime.

La problematica che quindi nasce da questo è il come risolvere tale *mapping* a livello *concettuale* e *tecnologico*.

- **Mapping concettuale**: si intende l'applicazione del paradigma REST all'Augmented World in termini di modello strutturale (deve essere tradotta la gerarchia strutturale di AW in modo coerente) e delle interazioni (WoAT deve wrappare tutte le interazioni che un agente/software esterno può effettuare su un AW, descritte come requisito poc'anzi, traducendole in API REST).

- **Mapping tecnologico**: si intende la decisione di quali framework/-tecnologie/modalità utilizzare al fine di soddisfare tutte le tipologie di interazioni comprese nel modello di AW (tra le quali, per esempio, il tracking di AEs in scenari event-oriented).

Un altro aspetto funzionale da tenere in considerazione è la *enter* nell'AW da parte di utenti attraverso la User App (figura 3.4). Quando un utente accede al sistema, oltre ad attivare una serie di agenti computazional per il tracking delle entità che lo circondano (per il *rendering* degli ologrammi), possono eventualmente creare un proprio *User Mirror*(avatar), ovvero una entità aumentata con la proprietà gaze ("sguardo", ovvero dove in un dato momento un utente ha la visuale) e priva di azioni. Per quella specifica entità, solamente l'utente potrà modificarne posizione e gaze, mentre per tutti gli altri agenti/user tali proprietà dovranno essere solamente *readable*.

La problematica che ne deriva è quella di riuscire ad inserire l'avatar utente tra le entità dell'AW, ma permettere soltanto a lui di modificarlo/eliminarlo.

4.1.2 Aspetti non funzionali

Nello scenario WoAT gli utenti del sistema devono (da requisiti) poter essere numerosi e *dinamici*, ovvero possono entrare e uscire dal Mondo Aumentato a loro piacimento.

Un ulteriore requisito emerge dal fatto che si vuole *aumentare il mondo fisico* arricchendolo con delle entità totalmente virtuali, in questo scenario tutto l'apparato virtuale (AEs e agenti che accedono all'AW) deve essere *sincronizzato* globalmente, onde evitare lag e inconsistenze.

Da questi requisiti sorgono le seguenti problematiche:

- Trovare un modo efficace per gestire la **sincronizzazione** globale delle entità. Essendo un punto critico (necessità *fisica* che va esaudita da un'infrastruttura virtuale) bisogna cercare di arginare al massimo le inconsistenze tra le visioni dell'AW da parte di tutte le entità. Si utilizza il termine arginare in quanto, per definizione, il ritardo computazionale è inevitabile di per sè se si parla di software. L'architettura REST del Web di suo aiuta in questo frangente in quanto altamente scalabile, ma sarà necessario investigare più a fondo riguardo questi aspetti nelle fasi successive. La sincronizzazione è stata definita *necessità* perchè, in ottica di sistemi aumentati, gli utenti non solo potranno interagire con entità virtuali nella realtà, ma inizieranno a basare i propri ragionamenti su di esse, facendole diventare parte del proprio processo cognitivo durante il lavoro. Avere entità aumentate non sincronizzate, in quest'ottica, potrebbe comportare conseguenze piuttosto spiacevoli, e si deve includere tra le problematiche anche quella della visualizzazione del *metastato* delle AE all'utente, qualora queste andassero *offline* per qualche ragione (non si potessero più aggiornare).

- Collegato al primo punto per via indiretta, sarà necessario sicuramente decidere le modalità di gestione degli utenti che si collegano tramite WoAT alle entità dell'AW, in quanto *molteplici* e *dinamici*.

- Infine, gli AWs possono essere di grandi dimensioni (grande mole di entità che li popolano), e non è concepibile che restino su un singolo nodo computazionale, è quindi un problema da valutare quello della gestione della *distribuzione* di un singolo AW su più nodi (con attenzione a consistenza e sincronizzazione) mantenendo alta la *scalabilità*.

4.2 Problema del Mapping: design API in chiave WoT

In questa sezione si procederà con il mapping delle API del modello di AW in API coerenti al paradigma REST in ambito WoT. Per fare ciò verrà definito prima un *Modello strutturale delle risorse* del modello di AW, dunque verranno progettate, applicando i principi di design menzionati nel primo capitolo, le singole chiamate HTTP che il livello WoAT dovrà gestire nell'infrastruttura da implementare.

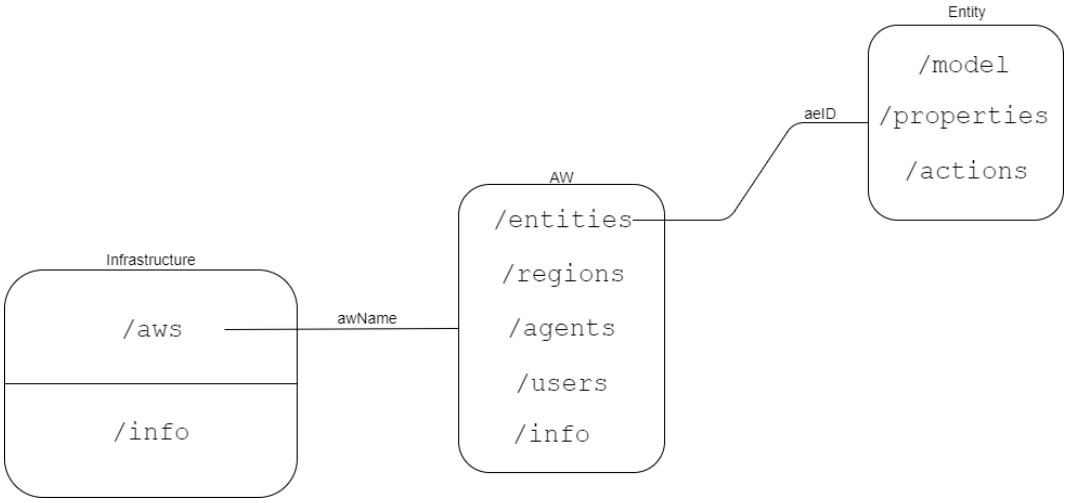


Figura 4.2: Modello REST di Augmented World

4.2.1 Definizione del modello strutturale di AW in termini REST

La prima proposta di modello strutturale che si formalizza è quella mostrata in figura 4.1. Si definisce come *Infrastructure* il nodo in cui è istanziato il server WoAT, il quale servirà le richieste agli *aws* ad esso collegati.

Ogni risorsa AW è accessibile attraverso il proprio *awName*, e contiene al proprio interno il riferimento alle proprie *entities* (AEs) e *regions* (che da modello in figura 3.4 compongono l'AW). Poichè ad un AW possono effettuare *join* gli agenti ed *enter* le applicazioni Utente, in ottica di gestione dei *sessionID* di entrambe le categorie, vengono previsti (come figli della risorsa AW) i path *users* e *agents*.

Le entità che abitano un AW, essendo da requisito da considerare Web Things vere e proprie, seguono il *Modello di Web Thing* formalizzato da Guinard e citato nella sezione 1.5.2. Presentano infatti come figli *model, properties* ed *actions*.

4.2.2 Mapping Concettuale API Request-Response

Nel procedere nel mapping delle API, si decide di cominciare con tutte le interazioni di tipo *request-response*, quindi si escludono, e verranno trattate successivamente, le API riguardante il *tracking* di entità e Regioni. Per definire

le singole chiamate verrà utilizzato il tool di modellazione di RESTful API **Swagger**.

Join di un AW (lato Agente)

Questa API permette all'agente richiedente di fare *join* in un determinato AW, attraverso la creazione di un *sessionID* univoco che poi deve essere restituito al mittente. Analizzando la richiesta, si nota che la semantica di *join* non è presente tra le operazioni CRUD, ma la necessità di creare un id per ogni richiesta di questo tipo, suggerisce la strada da prendere per il mapping. Dal modello strutturale proposto, infatti, si possono raccogliere tutti gli id di sessione come figli di */aws/awName/agents*, e si decide dunque di mappare l'azione di *join* in una creazione di risorsa.

A seguire il mapping proposto, in codifica *YAML*.

```
'/aw/{awID}/agents':
    post:
      description: Join of a Agent in the AW
      operationId: agentJoin
      parameters:
        - name: awID
          in: path
          description: AW Id
          type: string
          required: true
        - name: credentials
          in: body
          description: agent credentials
          schema:
            $ref: '#/definitions/AgentCredentials'
          required: true
      responses:
        '201':
          description: 'Successful, join the agent and retrieve session ID'
          schema:
            $ref: '#/definitions/SessionID'
        '404':
          description: 'Not Found'
        '400':
          description: 'Bad Request'
```

Quit di un AW (lato Agente)

Questa API permette all'agente richiedente di fare *quit* da un determinato AW, attraverso l'eliminazione dell'ID di sessione ad esso assegnato. Poichè l'operazione di join è stata progettata come una creazione della risorsa *sessionID* sotto la risorsa *../agents*, l'operazione di quit andrà a contrapporsi ad essa eliminando tale risorsa.

A seguire il mapping proposto, in codifica *YAML*.

```
'/aw/{awID}/agents/{sessionID}':
    delete:
      description: Remove an agent to the aw, deleting the session
      operationId: agentQuit
      parameters:
        - name: awID
          in: path
          description: AW Id
          type: string
          required: true
        - name: sessionID
          in: path
          description: agent session ID
          type: string
          required: true
      responses:
        '202':
          description: 'Successful, quit the agent from the AW'
        '404':
          description: 'Not Found'
        '400':
          description: 'Bad Request'
```

Creazione di Entità Aumentate

La creazione rientra tra le operazioni CRUD supportate dai verbi dell'HTTP nell'ottica di REST, risulta quindi abbastanza semplice definire un mapping per questa API.

A seguire il mapping proposto, in codifica *YAML*.

```
 '/aw/{awID}/entities':
```

```
    post:
      description: Create a new entity in aw
      operationId: createAE
      parameters:
        - name: awID
          in: path
          description: AW Id
          type: string
          required: true
        - name: entityDetails
          in: body
          description: AE details
          schema:
            $ref: '#/definitions/AEComplete'
          required: true
      responses:
        '201':
          description: 'Successful, creates the new AE a retrieves its ID'
          schema:
            $ref: '#/definitions/ID'
        '404':
          description: 'Not Found'
        '400':
          description: 'Bad Request'
        '401':
          description: 'Unauthorized'
```

Distruzione di Entità Aumentate

Questa API si contrappone alla precedente, ed in chiave REST può essere facilmente mappata con una *delete* di risorsa.

A seguire il mapping proposto, in codifica *YAML*.

```
'/aw/{awID}/entities/{entityID}':
    delete:
      description: Remove an AE from the aw
      operationId: removeAE
      parameters:
        - name: awID
          in: path
```

```
        description: AW Id
        type: string
        required: true
      - name: entityID
        in: path
        description: AE Id
        type: string
        required: true
    responses:
      '202':
        description: 'Successful, removes the AE from the AW'
      '404':
        description: 'Not Found'
      '400':
        description: 'Bad Request'
      '401':
        description: 'Unauthorized'
```

Spostamento di una entità

In questo caso è necessario ragionare su quale sia la scelta più coerente al paradigma REST da fare in questo caso, poichè modellando la struttura di una AE analogamente ad una Web Thing, essa possiede al suo interno la proprietà *Location*.

Dato che non vi sono specifiche ulteriori per questa API oltre alla sua signature, si decide di optare per la soluzione più semplice: ovvero mappare la *move* di un'entità nell'update della proprietà *Location*.

A seguire il mapping proposto per la modifica generale di proprietà (da applicare alla locazione per la corrente API), in codifica *YAML*.

```
'/aw/{awID}/entities/{entityID}/properties/{property}':
    put:
      description: Update an AE's property with the value in input
      operationId: updateAEProperty
      parameters:
        - name: awID
          in: path
          description: AW Id
          type: string
```

```
        required: true
      - name: entityID
        in: path
        description: AE Id
        type: string
        required: true
      - name: property
        in: path
        description: AE property
        type: string
        required: true
      - name: propertyValue
        in: body
        description: new property value
        schema:
          type: object
        required: true
    responses:
      '202':
        description: 'Successful, updates the AE property with new value'
      '404':
        description: 'Not Found'
      '400':
        description: 'Bad Request'
      '401':
        description: 'Unauthorized'
```

Creazione di una Regione

Dato il modello strutturale, la creazione di una regione, in chiave REST, è associabile alla creazione di risorsa al ramo */aws/awName/regions*.

A seguire il mapping proposto, in codifica *YAML*.

```
'/aw/{awID}/regions':
  post:
    description: Define a new region in aw
    operationId: defineRegion
    parameters:
      - name: awID
        in: path
```

```
      description: AW Id
      type: string
      required: true
    - name: regionDetails
      in: body
      description: region details
      schema:
        $ref: '#/definitions/RegionComplete'
      required: true
  responses:
    '201':
      description: 'Successful, define the new region a retrieves its ID'
      schema:
        $ref: '#/definitions/ID'
    '404':
      description: 'Not Found'
    '400':
      description: 'Bad Request'
    '401':
      description: 'Unauthorized'
```

4.2.3 Mapping Tecnologico scenario Event-Driven

Tra le API da convertire coerentemente con il paradigma WoT troviamo il *tracking* di regioni ed entità presenti in un AW. Queste operazioni implicano un tipo di comunicazione Event-Driven, bisogna pertanto applicare le proposte di Web of Things per questo tipo di comunicazione (paragrafo 1.5.1).

Sarà necessario l'impiego di *WebSocket*, si definiscono in seguito le routes che si intende aprire al protocollo:

- Tutte le risorse figlie di */aws/awName/regions*, al fine di mappare il tracking di regioni.
- Tutte le risorse figlie di */aws/awName/entities*, al fine di mappare il tracking di entità.
- Tutte le risorse figlie di */aws/awName/entities/aeID/properties*, anche se non richiesto, non si ritiene che l'effort dell'aggiunta di tracking su singole proprietà di AEs possa risultare eccessivo una volta implementato quello per regioni ed entità.

Per le risorse elencate, basterà effettuare una chiamata GET in linea con l'Handshaking del protocollo WebSocket (aggiungendo la richiesta di *upgrade*).

4.3 Problema della gestione dell'utente

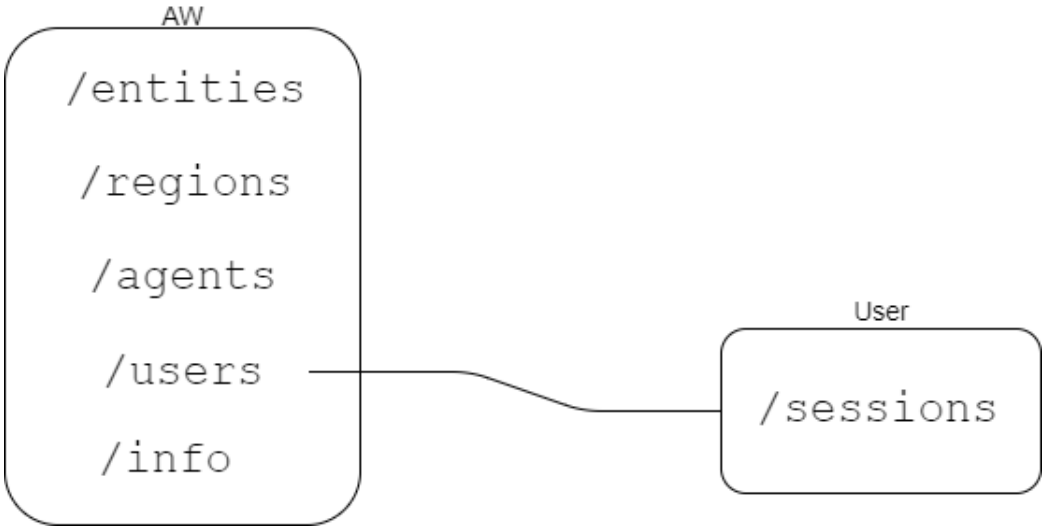


Figura 4.3: Modello strutturale lato Utente

Nello scenario previsto dai requisiti, le User App (applicazioni su devices che abilitano la AR quali occhiali, visori, smartphones, etc.) per poter accedere ad un mondo aumentato e parteciparvi devono effettuare l'azione di *enter*. La enter non è accomunabile alla *join* degli agenti in quanto opzionalmente, un utente può inserire all'interno dell'AW un suo avatar/mirror, che solo lui può gestire, mentre tutti possono osservare, essendo questo una entità aumentata a tutti gli effetti.

Si decide dunque di differenziare l'accesso degli agenti da quello degli utenti, creando un *ID si sessione univoco* ad ogni *enter* effettuata da un utente. Inoltre, si prevede l'aggiunta degli endpoint al server WoAT necessari all'aggiornamento di *location* e *gaze* del proprio avatar (aggiornamenti che si compiranno solamente mostrando il proprio ID di sessione).

In figura 4.2 si mostra l'aggiunta fatta al modello strutturale, in cui si espande la parte user con *sessions*, route alla quale si potranno inserire e rimuovere chiavi di sessione nello stesso modo previsto per la parte ad agenti (join).

Si decide dunque di dotare le User App di un Client specifico, che dovrà essere implementato per la singola tecnologia impiegata dalla user app, seguendo la progettazione del componente prevista in figura 4.5.

Il interazioni definite per la *enter* e la modifica di proprietà ad uno avatar utente sono descritte dal diagramma di sequenza in figura 4.3, in modo da dimostrare il concetto principale, ovvero che in ottica di realizzazione di User Apps , una volta sviluppato un client adatto per una tecnologia specifica, allo sviluppatore basterà utilizzarlo e effettuare la *enter* per attivare un processo di continuo tracciamento ed aggiornamento automatico (eventualmente parametrizzabile) di posizione e gaze.

4.4 Problema della dinamicità degli utenti

A monte della definizione del modello strutturale in chiave REST e delle API di gestione, si può affermare che la problematica sia già stata affrontata e risolta, in quanto allo stato attuale è prevista un operazione di *join/quit* per le entità Agente, e di *enter/quit* per le entità Utente. Non vi sono limiti teorici al numero di utenti/agenti che vogliono partecipare al sistema, e non vi sono risorse vincolate a nessun partecipante, fatta eccezione per gli avatar utente, la cui esistenza ad ogni modo non avrebbe senso all'uscita dell'utente dall'AW.

4.5 Problema della distribuzione

La problematica della distribuzione, nel caso di questa tesi, si è scelto di prenderla in considerazione ma demandandone la realizzazione in release successive.

Nello scenario distribuito gli AWs, anche se istanziati in remoto, devono comunque poter essere fruibili attraverso le API progettate, per cui lo strato WoAT deve necessariamente mascherare questa distribuzione. Si decide di demandare la gestione degli AW locali e remoti alla *Infrastruttura* che il layer WoAT servirà. All'interno di essa, gli AW estenderanno il proprio modello (figura 3.4) aggiungendo alle proprietà un **AwAddress**, ovvero il riferimento all'ip e porta in cui risiedono.

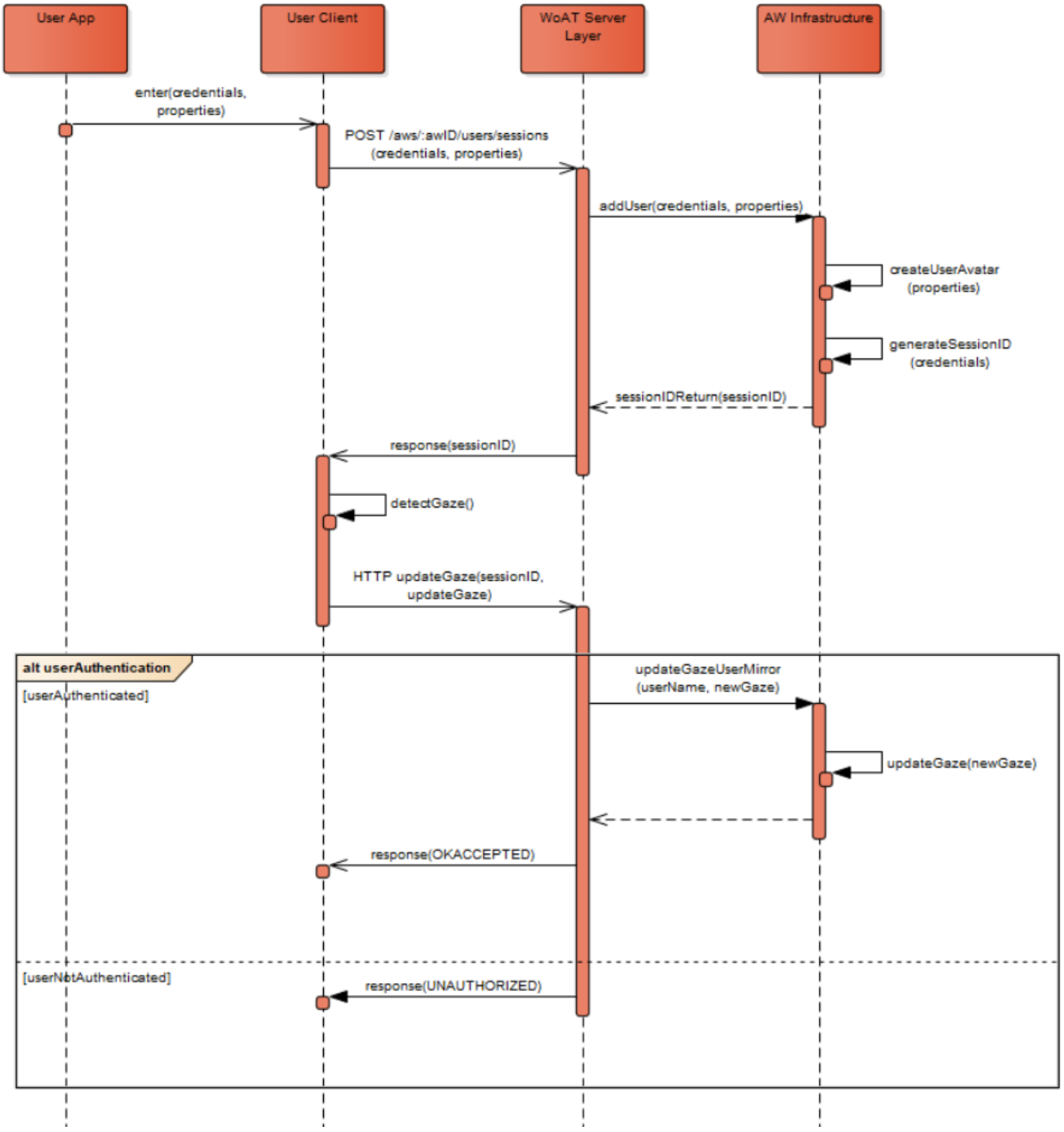


Figura 4.4: Modello interazioni tra utente e AW tramite WoAT

4.6 Problema della sincronizzazione

Analizzando il requisito legato alla problematica, si decide di dividerla in due parti: la sincronizzazione degli *ologrammi* e la sincronizzazione dello *stato*.

4.6.1 Sincronizzazione degli ologrammi

La sincronizzazione degli ologrammi si inquadra nella parte di modello di AW (è una proprietà che una AE può avere e di cui una User App può fare rendering). Per questo, tutta la gestione dell'aggiornamento di essi è demandata all'infrastruttura su cui è attivo l'AW, precisamente si prevede la presenza di un *Hologram Engine* interno all'infrastruttura AW attraverso il quale tutte le AE che cambiano il proprio stato possano propagare questo cambiamento al proprio ologramma.

Si delinea un modello in cui esisterà una controparte "server" di *Hologram Engine* residente su una qualche tecnologia di rendering, che raccoglierà tutti gli update inviatigli dall'infrastruttura e aggiornerà globalmente gli ologrammi di tutti i dispositivi collegati all'AW, senza passare da WoT. La motivazione di ciò è legata principalmente al fatto che si esclude che la sincronizzazione degli ologrammi possa essere legata a meccanismi di tracking via WoT.

Per questo motivo, non si tratterà oltre questa problematica, poichè seppur presente, fuori dal focus del layer WoAT.

4.6.2 Sincronizzazione dello stato

La sincronizzazione dello stato è circoscritta a tutte quelle entità accoppiate ad oggetti/utenti fisici, poichè con stato si intende *il livello di sincronizzazione attuale tra entità fisica ed entità aumentata*. Per proporre un esempio: se una entità aumentata perdesse il contatto con una delle proprie Things associate (errore di rete, etc.), il suo stato diverrebbe "offline/offsync".

Si definisce una specializzazione di AE, chiamata **Avatar**, in cui sarà presente un riferimento allo stato della sincronizzazione (presumibilmente booleano), da porre nel modello di AW (figura 3.4), dalla quale poi si specializzeranno *User Mirror* e *Thing Mirror*.

Non resta che considerare la realizzazione di *drivers* per mantenere aggiornato lo stato della sincronizzazione (invio di *keepalive*) da porre su: applicazioni agente di things fisiche e user apps. L'invio di *keepalive* verrà implementato direttamente nei drivers specifici per ogni singola tipologia di connessione.

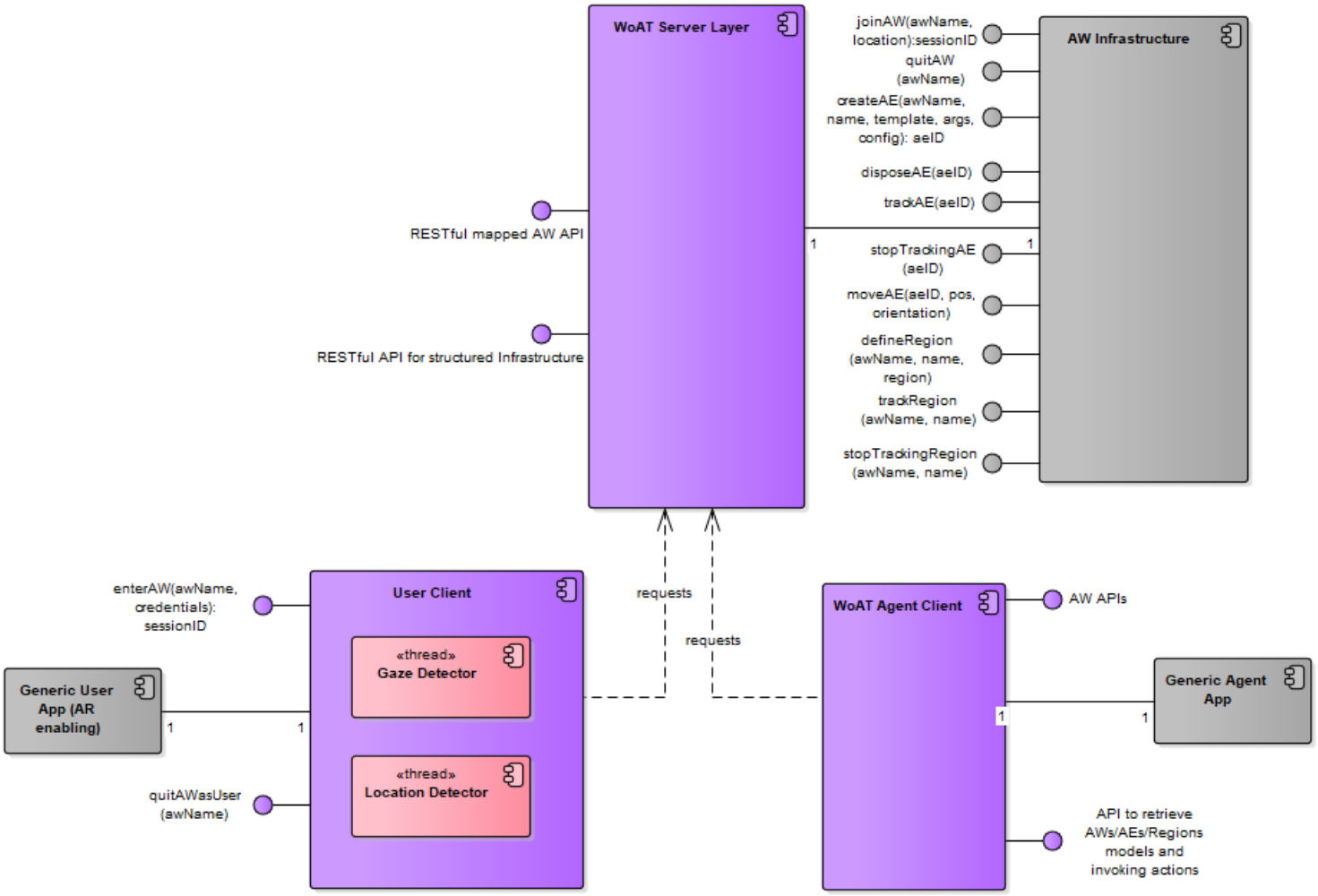


Figura 4.5: Architettura globale dell'infrastruttura WoAT

4.7 Architettura del Sistema

Analizzate le problematiche dettate dai requisiti del sistema, si delinea un'architettura ad alto livello che mostri le componenti del sistema (figura 4.4). Colorate in viola vi sono le parti in cui la tesi si focalizza, ovvero la realizzazione dell'infrastruttura che abilita la navigazione e l'utilizzo di un mondo aumentato strutturato secondo il modello AW attraverso Web of Things.

4.7.1 Descrizione dei componenti

AW Infrastructure

L'infrastruttura è mostrata in figura 4.6, ed è un "raccoglitore" di mondi aumentati, distinti tra *locali* e *remoti*, in modo da prevedere uno scenario distribuito.

La parte riguardante l'AW riprende il modello descritto in figura 3.4 , ed è progettato per fornire le APIs definite nei requisiti, le entità e le regioni che lo compongono (tutte o singolarmente), e il proprio ID. Ogni AW, a valle della problematica della distribuzione possiede un *AWAddress*, che verrà utilizzato nel caso di AW remoti.

Affrontando la problematica della molteplicità degli utenti unita a quella della sincronizzazione, si è deciso di estendere il concetto di AE con *Avatar*. In linea di principio, Avatar rappresenta tutte quelle AE *specchio* di una componente reale, sia essa un utente o un oggetto, che rende pubblica la propria effettiva sincronizzazione con la controparte fisica. Da essa derivano:

- *UserAvatar*: possiede la proprietà di "gaze" propria di un utente, rispecchia lo *User Mirror* nel modello di AW.
- *ThingAvatar*: possiede la proprietà di "things" per poterla connettere direttamente con gli oggetti fisici che la compongono, rispecchia lo *Thing Mirror* nel modello di AW.

HologramEngine è quel componente che riceve ed invia alla sua controparte server sulla tecnologia di rendering tutte le richieste per creazione di nuovi ologrammi/cambio di proprietà delle AEs esistenti.

WoAT Server Layer

La parte "server" del middleware WoAT non fa altro che tradurre le chiamate HTTP in entrata (API RESTful) in chiamate alle API del modello di AW, seguendo il mapping progettato. Eventuali meccanismi di sicurezza e controllo di ID di sessione verranno incapsulati in questo componente, così come la gestione delle risposte da inviare ai client e i relativi codici di risultato.

Fornisce gli endpoints per la modifica di proprietà di entità legate a utenti o things fisiche, con correlato controllo degli accessi.

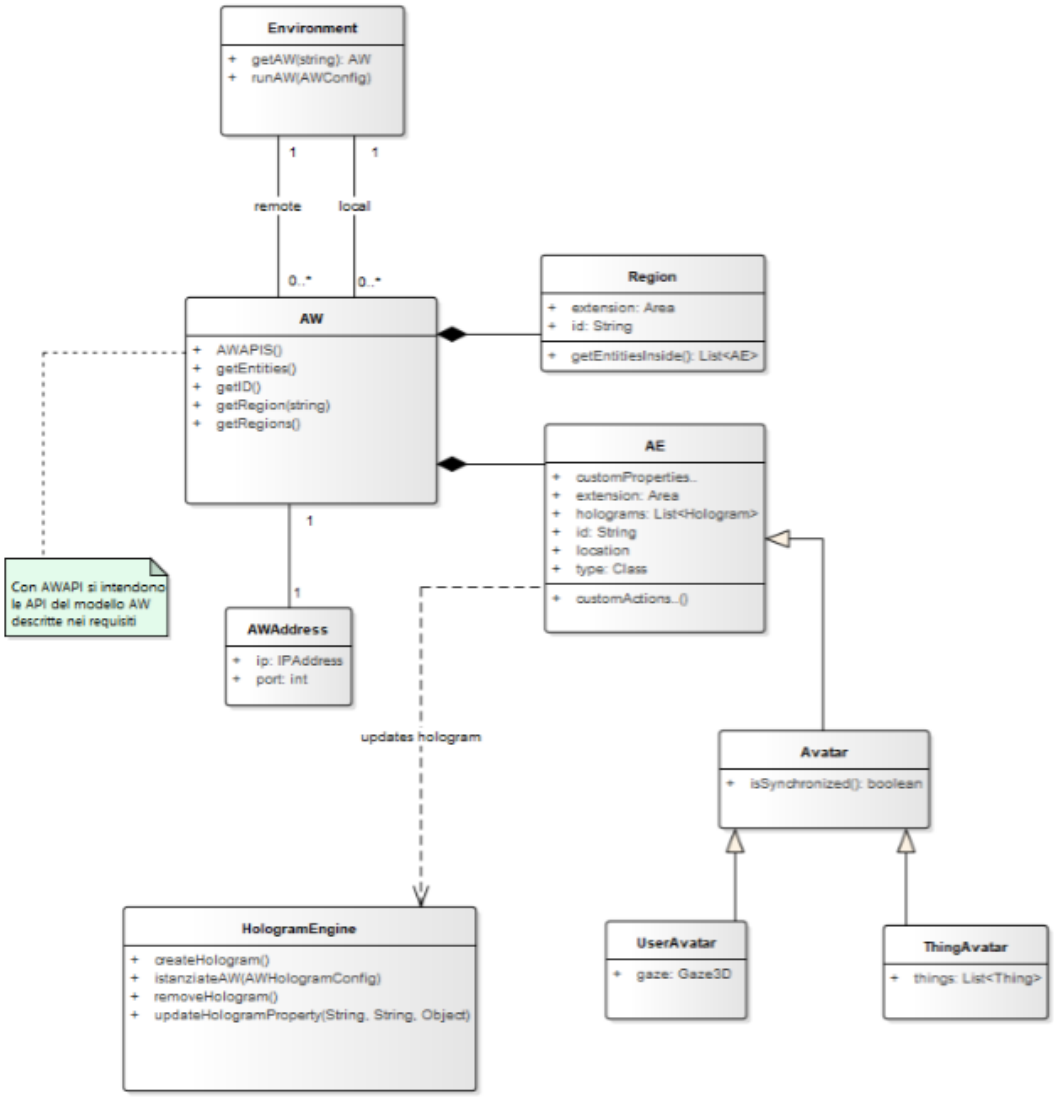


Figura 4.6: Diagramma delle classi ad alto livello della *Infrastructure*

WoAT Clients

WoAT ha come primo obiettivo la promozione della interoperabilità delle entità aumentate presenti negli AW, parlare dunque di Clients può essere fuoriviante. Porre API RESTful ad un AW permette a chiunque sul web di interagire con esso indifferentemente. Si possono pensare quindi, in direzione di possibili *librerie* da sviluppare oltre al Server Layer, due tipologie:

- *WoAT Agent Client*

- *WoAT User Client*

WoAT Agent Client

Rappresenta la parte client utilizzata da agenti che vogliono entrare in un AW, questo componente incapsula la logica del mapping di AW API in RESTful API, e fornisce alle applicazioni che lo utilizzano i metodi base per lavorare con WoAT (comprese le richieste del modello delle risorse, non previste nelle API elencate in tabella 4.1).

Ponendo il focus sui *Drivers* da inserire nelle Things che poi avranno un avatar nel mondo aumentato, essi potrenno usare questo client, sarà l'infrastruttura a estrapolare dalle richieste di creazione di ThingAvatar i dati necessari per il controllo degli accessi, e sarà demandata al tipo di tecnologia di collegamento la trasmissione del keepalive.

WoAT User Client

Già in parte trattato nella sezione 4.3, lo User Client è il client utilizzato dalle User App per accedere al sistema (operazione *enter* da requisito) registrando l'utente che la utilizza e generandone un avatar. Attraverso questo client si potranno mantenere automaticamente aggiornate *location* e *gaze* del proprio avatar,attraverso l'impiego di processi atti al tracciamento e all'aggiornamento periodico di queste proprietà.

User App

Il concetto di User App si trova anche nel modello di AW (figura 3.4). Essa viene lanciata da un device che abilita l'AR all'utente, e può sia generare agenti, sia creare un avatar utente e gestirlo in autonomia utilizzando i dati relativi a posizione e sguardo ricavati dal dispositivo. Si interfaccia con WoAT utilizzando il WoAT User Client.

Nello scenario corrente, si ipotizza che la user App impieghi una tecnologia per il rendering degli ologrammi compatibile con la tecnologia su cui risiede la parte server dell'*Hologram Engine*.

Capitolo 5

Implementazione prototipale di WoAT

In questa sezione si procederà a descrivere l'implementazione del sistema suddividendola per componenti.

5.1 Infrastructure

L'infrastruttura è una componente essenziale per il sistema in quanto è ciò che viene mascherato dal layer WoAT, nel procedere nell'implementazione, si è studiato e poi ampliato un prototipo sviluppato precedentemente dal gruppo di ricerca promotore di questa tesi.

5.1.1 Situazione iniziale: prototipo esistente per Infrastructure

Il prototipo esistenze, architetturalmente parlando, è stato progettato rispettando il modello di AW, e presentava le seguenti caratteristiche:

- Implementazione di tutto il modello in progetto chiamato **core** scritto in *Java*, corredato di concetto di *Environment* (analogo alla concezione di Infrastructure) contenitore di un singolo AW (*Singleton*).

- Implementazione ad uno stato avanzato e completo della parte di AE, capace di fornire la possibilità agli sviluppatori che utilizzano il core di creare templates di entità utilizzando le dichiarazioni *@HOLOGRAM, @PROPERTY* E *@ACTION*.

- Implementazione dell'Hologram Engine: lato client nel core, e lato server con *Unity 3D*[1] , equipaggiato con il puglin di *Vuforia*[2] per la gestione degli aspetti di AR attraverso script sviluppati in C# per le features specifiche di AW. Per associare ad una AE il suo ologramma, si inserisce il riferimento al prefab in Unity nella proprietà @HOLOGRAM.
- Inserimento delle *settings* dell'AW via file JSON.
- Implementazione utilities per lavorare con locazione (geografica, cartesiana) ed estensione (circolare).
- Implementazione di un *Hologram Engine* lato server funzionante in Unity 3D.

5.1.2 Evoluzione del prototipo

Rispetto al prototipo iniziale, sono state effettuate le seguenti modifiche:

- Modificato il modello e le ontologie interne al core per farle aderire con la progettazione (AW multipli, locali o remoti, aggiunta regioni, estese AE con il concetto di Avatar).
- Inserito il concetto di *RemoteObserver* per modellare i mittenti delle richieste di tracking e poterli notificare.
- Inserite utilities per controllo degli accessi di utenti e agenti, necessarie in quanto la molteplicità di AWs implica un controllo specifico per ognuno di essi. Aggiunte API che lo strato di WoAT Server Layer utilizzerà per verificare accesso di utenti ed agenti.
- Inserimento delle settings di ogni AW spostate nel codice per scongiurare problemi nell'eventualità di AW dinamici.
- Ulteriori modifiche dovute a bug/rifattorizzazioni, tutte in ambito Java.

5.2 WoAT Server Layer

Il layer lato server di WoAT è stato sviluppato in Java, coerentemente con la parte di infrastruttura, utilizzando **Vert.x**[3] , un *application framework* che gira su Java Virtual Machine, descritto dalle seguenti caratteristiche:

[1] https://unity3d.com/
[2] https://www.vuforia.com/
[3] https://vertx.io/

- Poliglotta: i componenti delle applicazioni possono essere scritti in diversi linguaggi, tra i quali Java.
- Modello di concorrenza semplice: tutto il codice sfrutta un singolo thread, evitando problemi derivati dal multi-threading.
- Modello di programmazione asincrono: per sviluppare applicazioni scalabili e non bloccanti.
- Event bus distribuito tra client e server, per lo sviluppo di applicazioni *real time*.
- Repository pubblico , per riutilizzare o condividere componenti.

Questo ha dato la possibilità di dividere l'implementazione del layer in due parti principali:

- Una parte che estende il concetto di Vert.x di *AbstractVerticle*, in cui si genera un server HTTP in ascolto su una determinata porta
- Nella seconda parte vengono implementati gli *handlers* delle richieste in entrata, che in linea di principio, in base alla richiesta, accedono all'Infrastruttura chiamando API del modello AW o ottenendo il modello di entità/regioni/AW richiesti. Negli handlers è stato implementato il controllo degli accessi alle AEs, attraverso la richiesta di validità dei sessionID all'AW "bersaglio".

5.2.1 Implementazione API

Per implementare le RESTful API descritte in fase di progettazione, sono state definite nel server HTTP istanziato nell'*AbstractVerticle* le routes possibili a cui si può fare richiesta, correlate dai verbi HTTP adeguati. Per ogni routes è stato assegnato un handler da invocare ad ogni richiesta di un certo tipo. Grazie al framework utilizzato, osservando il codice può agilmente essere visualizzato l'insieme di API implementate:

```
router.get("/info").handler(handleGetInfrastructureInfo);
router.get("/aw/:awID/info").handler(handleGetAWInfo);
router.post("/aw/:awID/agents").handler(handlePostAgentJoinAW);
router.delete("/aw/:awID/agents/:sessionID").handler(handleDeleteQuitAW);
router.post("/aw/:awID/users/sessions").handler(handlePostUserJoinAW);
router.delete("/aw/:awID/users/sessions/:sessionID").handler(handleDeleteQuitUser);
router.post("/aw/:awID/regions").handler(handleDefineRegion);
router.get("/aw/:awID/regions").handler(handleGetRegionsDetails);
router.get("/aw/:awID/regions/:regionID").handler(handleGetRegion);
```

```
router.get("/aw/:awID/entities").handler(handleGetEntities);
router.post("/aw/:awID/entities").handler(handleAddAugmentedEntity);
router.get("/aw/:awID/entities/:entityID").handler(handleGetEntityByID);
router.delete("/aw/:awID/entities/:entityID").handler(handleRemoveEntityByID);
router.get("/aw/:awID/entities/:entityID/model").handler(handleGetEntityModel);
router.get("/aw/:awID/entities/:entityID/model/:modelElement")
        .handler(handleGetEntityModelElement);
router.get("/aw/:awID/entities/:entityID/properties")
        .handler(handleGetEntityProperties);
router.get("/aw/:awID/entities/:entityID/properties/:property")
        .handler(handleGetEntityProperty);
router.put("/aw/:awID/entities/:entityID/properties/:property")
        .handler(handleSetEntityProperty);
router.get("/aw/:awID/entities/:entityID/actions").handler(handleGetEntityActions);
router.post("/aw/:awID/entities/:entityID/actions/:action")
        .handler(handleDoActionOnEntity);
```

Un esempio di implementazione di API REST è mostrato in figura 5.2, in cui si riporta il caso della join di un agente: in cima la definizione della route da gestire (inserita in un *Router* di Vert.x, addetto a indirizzare alla prima route facente *match* le richieste entranti al server HTTP), in basso l'handler correlato.

5.2.2 Implementazione scenario Event-Driven (tracking)

Per gestire le richieste di tracking è stato associato al server HTTP istanziato un *webSocketHandler* (disponibile attraverso le API di Vert.x), in cui viene prima analizzata la richiesta in entrata per capire il *target* del tracking, poi effettuato il controllo degli accessi, per poi accettare ed attivare la comunicazione con client. Ogni client viene mappato come *RemoteObserver*, ed il riferimento alla WebSocket associata verrà mantenuto in memoria dalle entità/regioni osservate.

L'implementazione della ricezione di richieste di tracking è riportata in figura 5.3: istanziazione di webSocketHandler nel Verticle e handler correlato (handler generale, che smista a sua volta la richiesta in base a che tipo di risorsa è oggetto di tracking).

5.3 WoAT Client Layer

In fase di progettazione sono stati delineate le tipologie di client, seppur non strettamente necessarie per lavorare con AW attraverso WoAT, che potrebbero

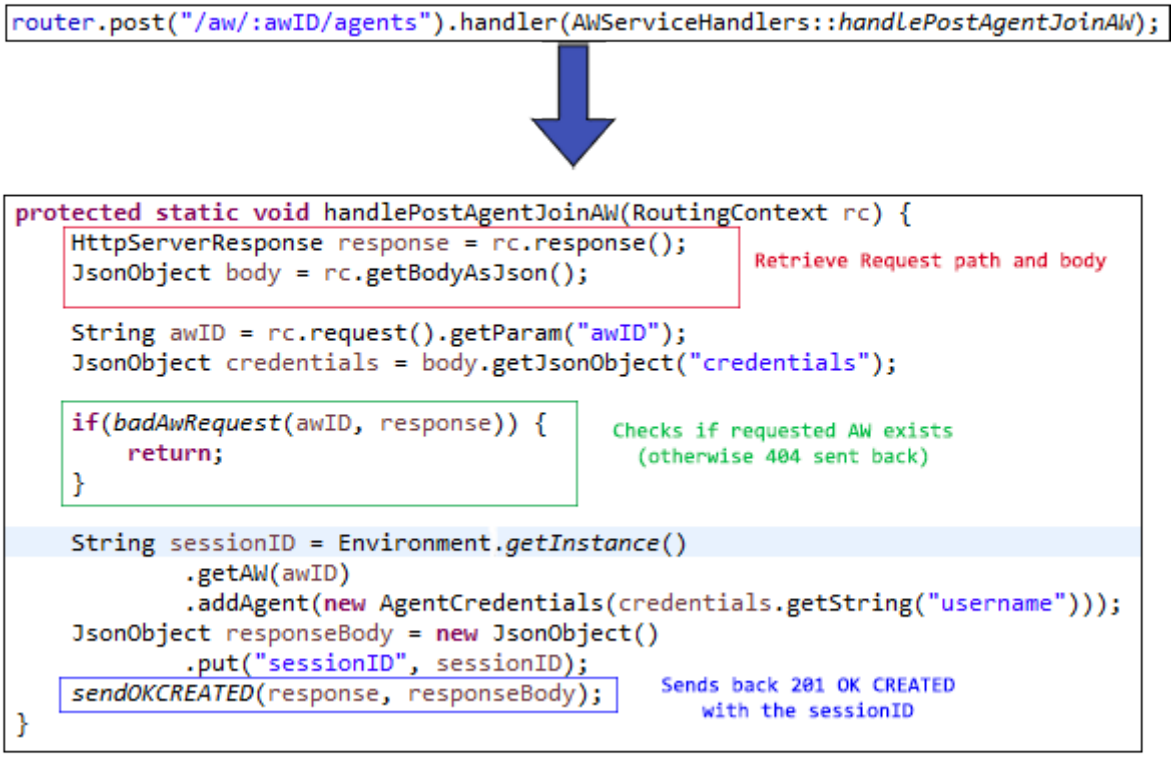


Figura 5.1: Implementazione richiesta di Join Agente

essere sviluppati in veste di *facilities*. Per la tesi è stata sviluppato un WoAT Agent Client, anche per facilitare la parte di test in validazione.

5.3.1 Implementazione WoAT Agent Client

Questo client è stato sviluppato in Java sottoforma di libreria composta principalmente da un oggetto *Singleton* capace di fornire tutte le API di un AW esplicate tra i requisiti, in aggiunta ad altri metodi per l'ottenimento di risorse interne all'AW quali informazioni su regioni, AE, etc. Si è cercato di definire delle classi semanticamente aderenti al modello di AW, ad esempio per modellare le diverse tipologie di proprietà di una AE.

Per mappare le API in richieste HTTP al server, si è utilizzato anche in questo caso la piattaforma Vert.x (descritta precedentemente) per la generazione e utilizzazione di due tipi di client:

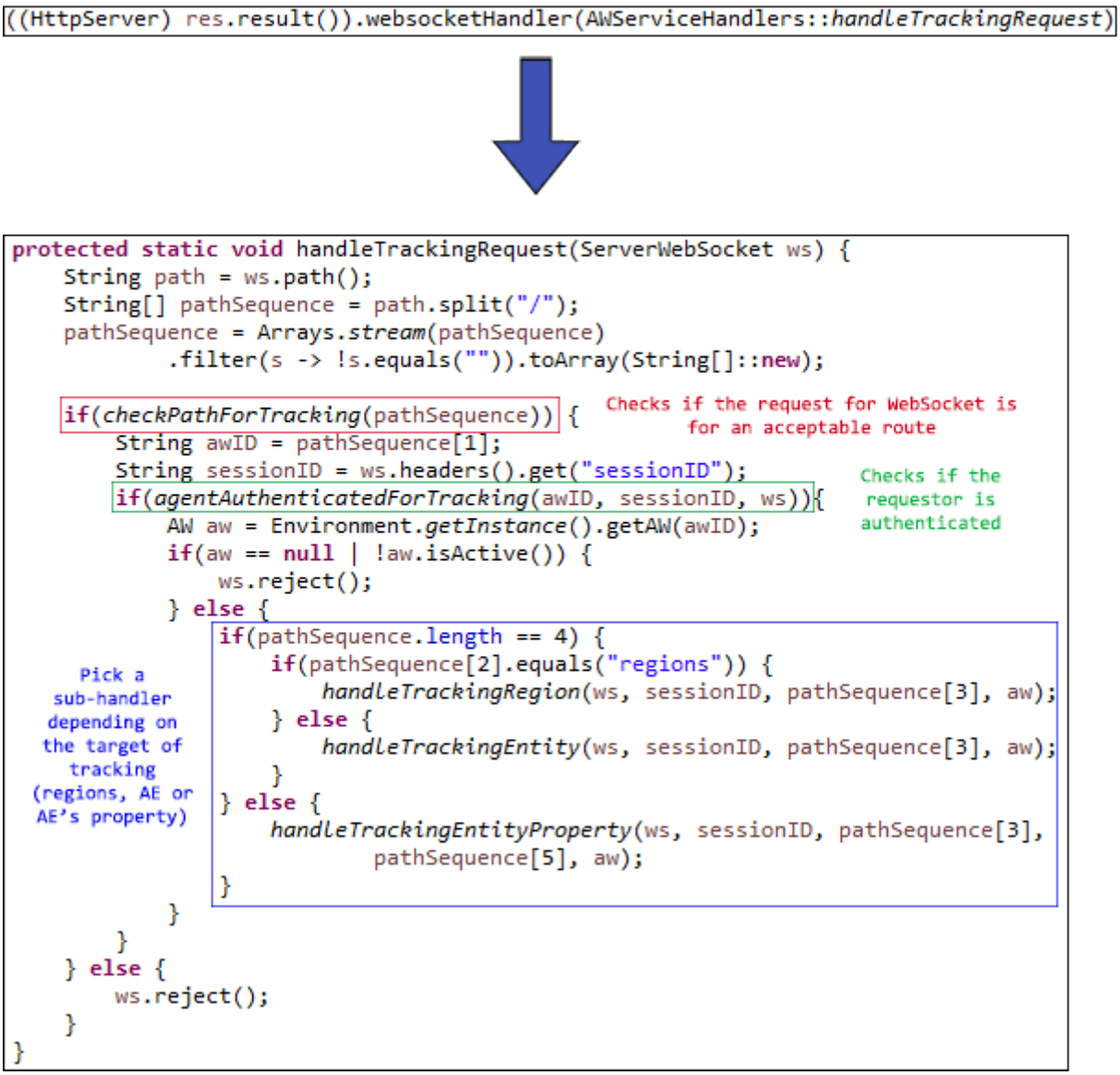


Figura 5.2: Implementazione gestione richiesta di tracking via WebSocket

- *HttpClient*: modella un HTTP client asincrono, che permette di effettuare richieste HTTP ad un qualsiasi server, aprire WebSockets, creare un *pool* di connessioni HTTP. È stato sfruttato il meccanismo di gestione WebSockets utilizzato per le API di tracking.

- *WebClient*: utilizzato per la parte di API RESTful, in quanto più agevo-

le dell'HttpClient. Questo client è molto simile al precedente, e presenta le seguenti peculiarità: codifica/decodifica JSON, inserimento parametri alle richieste, gestione errori unificata, sottomissione di forms. Non consente, però la gestione di WebSockets (che spiega perchè non si è utilizzato soltanto questo tipo di client).

5.3.2 WoAT User Client

Questa tipologia di client è stato sviluppato in Unity 3D, coerentemente con la tecnologia di implementazione della parte server dell'*Hologram Engine*, per la piattaforma Android. Come da progettazione del componente, esso fornisce come interfaccia la possibilità di entrare in un AW, e una volta azionato, contiene al suo interno dei processi autonomi in grado di ottenere periodicamente l'aggiornamento della posizione e del gaze dell'utente. Ognuno di questi aggiornamenti viene propagato attraverso una chiamata al server WoAT, attraverso le seguenti routes (codice estrapolato dal WoAT Server Layer):

```
router.post("/aw/:awID/users/actions/updateGaze").handler(handleUserUpdateGaze);
router.post("/aw/:awID/users/actions/updateLocation")
        .handler(handleUserUpdateLocation);
```

Capitolo 6

Validazione

In questo capitolo si descrive il processo impiegato per effettuare una validazione dell'infrastruttura, costituito sostanzialmente in:

- Test di performance e robustezza della parte server su API significative a livello di requisito.
- Sviluppo di una applicazione dimostrativa che comprenda l'immersione di più utenti in un AW, in modo da validare in sistema a livello funzionale.

6.1 Test di perfomance e robustezza API

La progettazione le lo sviluppo del layer WoAT lato server per il modello di AW è uno dei punti focali di questa tesi. Si vuole validarne la robustezza attraverso *stress tests* delle API RESTful implementate. Sono state scelte le seguenti API per il test:

- Creazione di entità aumentate.
- Tracking di regioni.

6.1.1 Creazione simultanea di AEs

Finalità dei test

Si intende testare questa API in modo da misurarne i limiti in termini di richieste andate a buon fine in presenza di un numero elevato di queste simultaneamente.

Ambiente di test

Per il seguente test si riproduce uno scenario in cui quattro agenti computazionali, in maniera simultanea, inviano un certo quantitativo di richieste consecutive all'AW, e sono in grado di misurare quelle andate a buon fine. Il test viene eseguito in rete locale, su due applicazioni differenti: una su cui gira il WoAT Server Layer e la sua Infrastructure, una su computano i thread (sono quattro come il numero di core presenti nella macchina che li lancia, per garantire il massimo parallelismo possibile).

Per controprovare il risultato finale fornito dai singoli thread (richieste andate a buon fine), si provvede a richiedere il numero totale delle AEs abitanti l'AW al termine della loro esecuzione, in modo da validare o meno il risultato dato dai singoli agenti, e dunque il singolo test.

Per massimizzare il rate richieste al secondo, nessun agente attende tra l'invio di due richieste consecutive.

Risultati e considerazioni

Si classifica ogni test in base al numero di richieste totali effettuate, si compiono 5 reiterazioni dello stesso test e ne si estrapola una *media* dei risultati singoli come risultato finale. I risultati dei test sono stati raccolti in figura 6.1.

Richieste	Perc. Successo (Media)	Tempo Impiegato (secondi)
1000	100,00%	1,4
2000	100,00%	2,2
5000	100,00%	4,2
8000	98,80%	6,8
10000	98,30%	9,2
20000	70,74%	14,6

Figura 6.1: Risultati stress-test sulla creazione di AEs

Alla luce dei risultati, si può affermare che:

- Il rate di richieste al secondo (r/s) aumenta globalmente (714 r/s per 1000 richieste, 1370 r/s per 20000), ma la percentuale di successo degrada.
- Il rate di *richieste effettuate con successo al secondo* globalmente rimane circa costante, assestandosi intorno alle 1000 r/s, ponendo un limite abbastanza chiaro e significativo per il test corrente.

In definitiva, con l'ambiente descritto, l'infrastruttura pare reggere un massimo di 1000 richieste di creazione di entità al secondo, non si esclude che questo dato possa calare in presenza di una configurazione con agenti e infrastruttura reciprocamente remoti.

6.1.2 Test di reattività del tracking

Finalità dei test

Si intende testare il meccanismo di tracking in modo da studiarne il comportamento ed eventuali anomalie in occorrenza di update frequenti e multipli osservatori simultanei. Si vuole capire se un elevato rate di aggiornamenti comporti una situazione in cui, per esempio, un secondo update arrivi ad un osservatore mentre un ad un altro non sia stato ancora notificato il primo.

Ambiente di test

Per il seguente test si decide di riprodurre uno scenario in cui 20 agenti computazionali osservano la medesima regione, mentre un ultimo agente crea periodicamente (con intervalli di tempo variabili) entità aumentate all'interno di quest'ultima. Il test viene eseguito in rete locale, su due applicazioni differenti, analogamente al test precedente.

Ogni ascoltatore mostra, non appena riceve un aggiornamento, il momento in cui l'ha ricevuto, in modo da provare la sequenza di arrivo degli aggiornamenti. Sia da notare che questo tipo di test potrebbe non essere banale in casi in remoto, in quanto i diversi orologi potrebbero non essere sincronizzati, e sarebbe complicato capire l'effettiva tempistica degli aggiornamenti.

Risultati e considerazioni

Si differenziano i test in base al *delay* impostato tra un update e l'altro della regione (ovvero tra un aggiunta di entità e l'altra), e si va ad osservare se prima di ricevere un certo update, tutti gli osservatori abbiano ricevuto o meno il precedente. Una volta trovato il "punto di rottura" con 20 tracker simultanei, si prova a aumentarli per effettuare nuove prove alla stessa velocita.

Effettuando almeno 3 prove per ogni impostazione di delay, si osservano i casi di 5, 10, 20, 40, 50 *updates al secondo*(u/s). Solamente nell'ultimo caso iniziano ad accavallarsi notifiche di update in modo che qualche agente risulti avanti di più di un update (generalmente 2).

Una volta definito 50 u/s come limite per 20 osservatori, è stato provato a raddoppiare il numero e dunque rieffettuare le prove, e il risultato non è cambiato (nessuna anomalia nell'ordine di notifica fino ai 50 u/s).

6.2 Validazione funzionale

Per la validazione funzionale, è necessario sviluppare un'applicazione dimostrativa che possa essere lanciata su dispositivi abilitanti AR, e integri l'infrastruttura WoAT realizzata in questa tesi. L'obiettivo della demo è quello di riuscire a dimostrare anche la fattibilità dell'integrazione della AR con il prototipo di infrastruttura WoAT, secondo quanto progettato, in modo semplice e senza impiegare tempo e risorse eccessive.

6.2.1 Applicazione Demo: descrizione

L'applicazione dimostrativa che si è scelto di realizzare con l'infrastruttura prototipale di WoAT è un'applicazione in cui 3 utenti e 2 agenti computazionali entrano nello stesso mondo aumentato:

- Gli agenti dopo aver effettuato la *join*, processano in modo differente. Uno, detto *coloratore*, non appena un utente entra nel mondo aumentato, crea dinamicamente un cubo ed una sfera aumentati iniziando il tracciamento del primo, mentre l'altro agente detto *rotatore* attende la creazione del cubo, per poi iniziare a farlo ruotare continuamente su un suo asse. Ogni volta che il cubo compie una rotazione di 90 gradi, il coloratore cambia il colore della sfera (prima giallo, poi blu ad intermittenza al raggiungimento di 90, 180 gradi, etc.).
- I tre utenti potranno osservare la scena immersi nel mondo aumentato attraverso un device mobile.

In figura 6.2 è formalizzata in un diagramma l'organizzazione complessiva del sistema.

Questa applicazione si dimostra una corretta forma di validazione funzionale in quanto, nel complesso, prevede il coinvolgimento dei seguenti casi d'uso (in riferimento ai requisiti in figura 4.1):

- *Enter* di utenti (dinamico) in un AW (con relativa creazione di UserAvatar).
- *Join* di agenti (dinamico) nell'AW

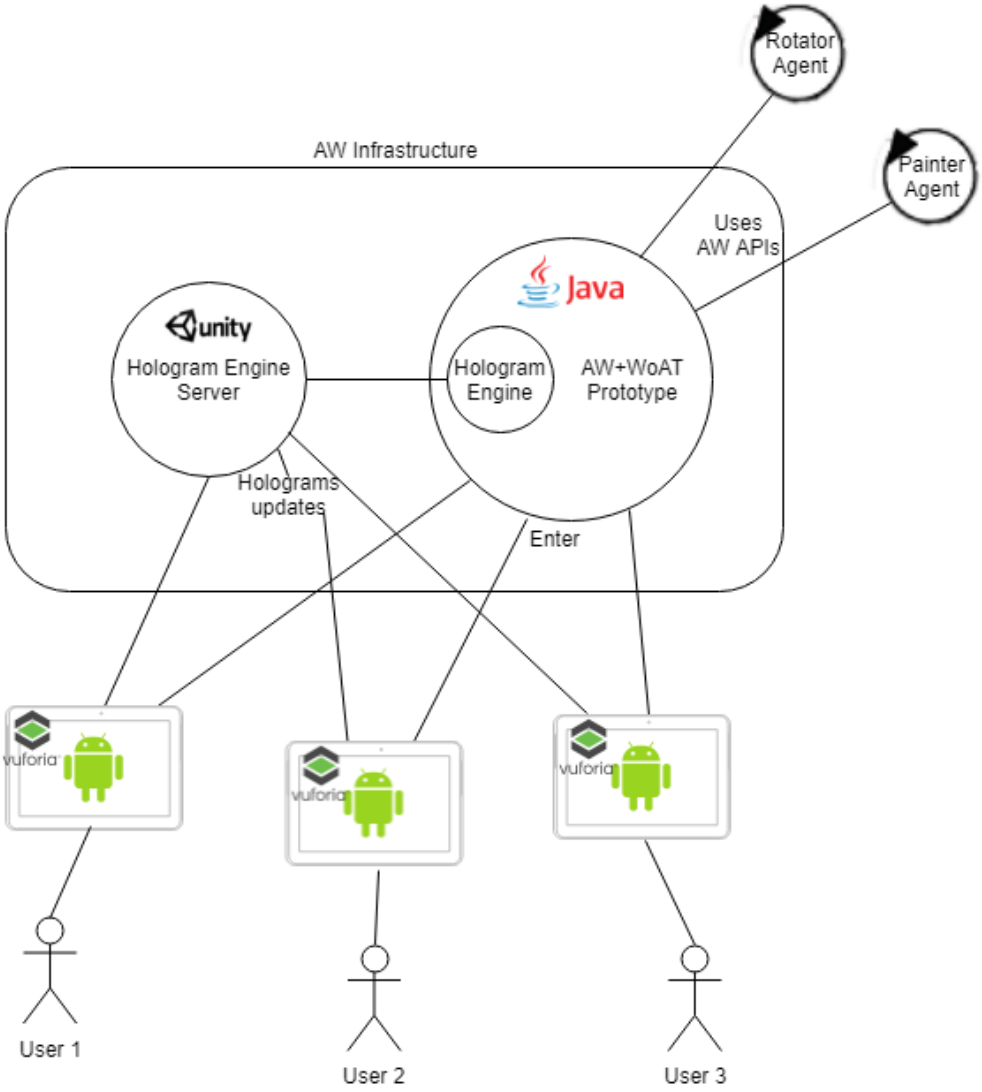


Figura 6.2: Panoramica globale dei componenti dell'applicazione di Demo

- *Rendering* degli ologrammi da parte dell'utente.
- *Tracking* di regione (per l'azionamento del coloratore e del rotatore) e di entità (coloratore osserva il cubo).
- Invocazione di azioni su un AE (rotatore per ruotare il cubo).
- Creazione di AEs (coloratore crea cubo e sfera).

- Definizione di regione (inizialmente dovrà esserne definita una per permettere agli agenti di osservare l'entrata di utenti/creazione di AEs).

Gli unici casi d'uso non descritti sono *controparti* dei casi d'uso elencati (operazioni di *quit, untrack, remove*), che verranno necessariamente compresi nella procedura di chiusura dell'applicazione dimostrativa (quando l'applicazione viene chiusa, si procede a rimuovere agenti e utenti dall'aw, bloccare i trackings, etc.).

6.2.2 Applicazione Demo: implementazione

Il prototipo di infrastruttura WoAT sviluppata in questa tesi ha permesso di creare l'applicazione di demo in tempi rapidi, è bastato, a tal fine, effettuare le seguenti operazioni:

- Creazione AW attraverso al definizione delle entità e della configurazione remota con l'Hologram Engine.
- Sviluppo dell'Hologram Engine lato server con Unity 3D, attraverso la definizione dei controller da associare agli ologrammi delle entità definite (Cubo e Sfera).
- Creazione di un applicativo Java che ospiti gli agenti Rotatore e Coloratore (implementati secondo specifica trattata precedentemente).
- Creazione User Apps in Unity per Android con la parte di User Client (*enter* nell'AW e aggiornamento posizione e gaze automatiche).

La definizione dei templates delle entità coinvolte nell'applicazione si è svolta sfruttando il meccanismo di annotazione implementato nel prototipo per questo tipo di operazioni, in seguito si riportano le definizioni di Sfera e Cubo.

```
@HOLOGRAM("Cube")
public class Cube extends AE {

   @ACTION
   public void rotate(Double degree) {
      AngularOrientation oldOrientation = (AngularOrientation)
          orientation();
      orientation(new AngularOrientation(
            oldOrientation.roll(),
            (oldOrientation.pitch() + degree)%360,
            oldOrientation.yaw()));
   }
```

```
}
```

```
@HOLOGRAM("Sphere")
public class Sphere extends AE {

   @PROPERTY
   private String color = "blue";

   @ACTION
   public void blink() {
      System.out.println("color changed");
      try {
         if (customProperty("color").equals("blue")) {
            customProperty("color", "yellow");
         } else {
            customProperty("color", "blue");
         }
      } catch (PropertyNotFoundException e) {
         e.printStackTrace();
      }
   }
}
```

Lo sviluppo dell'Hologram Engine lato server è consistito principalmente di: definizione dei prefab Unity legati alle entità aumentate, definizione dei controller legati alle entità (capaci di reagire agli update in arrivo dalla *AW Infrastructure*) e definizione del marker associato all'origine del sistema di riferimento.

Una panoramica del progetto Unity viene fornita in figura 6.3, in cui è stato riportato uno screenshot della schermata principale. I controller associati alle entità sono stati implementati estendendo agevolmente il concetto già presente nel prototipo di *HologramController*, in ascolto degli update (alle entità) derivanti dalla parte client dell'Engine. In seguito ne è riportato il codice.

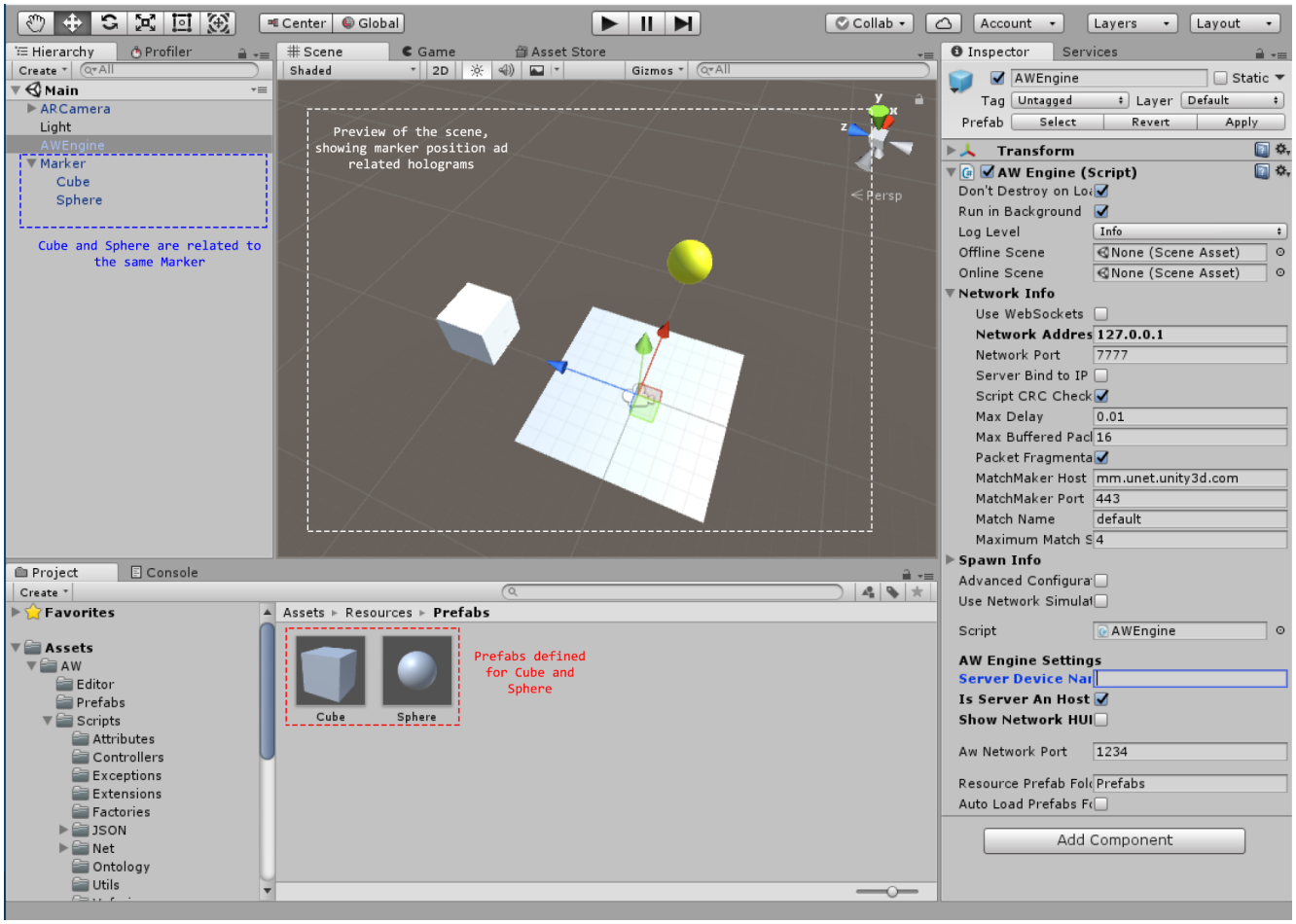


Figura 6.3: Schermata progetto Unity dell'applicazione di Demo

```
public class SphereController : HologramController {

    private const string COLOR_PROPERTY = "color";

    public override void onCustomPropertyUpdateReceived(string property,
        object value)
    {

        switch (property)
        {
            case COLOR_PROPERTY:
                manageChangeColor(value.ToString());
                break;
        }
```

```
    }

    private void manageChangeColor(string color)
    {
        if(color.Equals("blue"))
        {
            HologramRenderer.material.color = Color.blue;
        }

        if (color.Equals("yellow"))
        {
            HologramRenderer.material.color = Color.yellow;
        }
    }
}
```

La generazione del marker è avvenuto attraverso l'utilizzo di una piattaforma online preposta alla creazione di marker univoci ideali per applicazioni di Realtà Aumentata. In figura 6.4 il marker utilizzato nell'applicazione di demo.

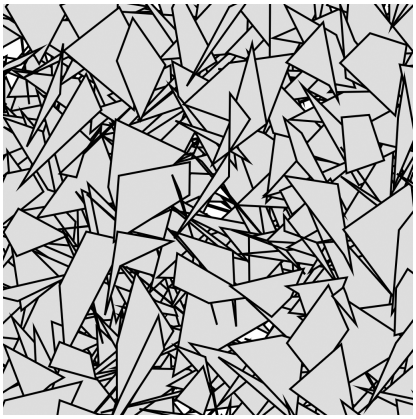


Figura 6.4: Marker utilizzato per il sistema di riferimento dell'Applicazione di Demo

Lo sviluppo degli agenti *Rotatore* e *Coloratore* è risultato decisamente semplice, a valle dei loro requisiti (descritti in precedenza), poichè è stato utilizzato il WoAT Agent Client sviluppato nel prototipo per interagire con l'AW sull'infrastruttura attiva. Essi sono stati implementati, per semplicità, come Threads Java.

Le applicazioni utente da installare sui dispositivi Android sono state generate con Unity come Clients remoti a partire dal progetto.

6.2.3 Applicazione Demo: risultato

Il risultato finale dell'applicazione demo rileva, tenendo conto della copertura trattata dei requisiti funzionali di WoAT, un discreto livello di prestazioni. Si pensi che, al contrario di una qualsiasi applicazione di realtà aumentata, alla base della Demo vi è lo strato di WoAT (Infrastruttura + WoAT Server Layer) che raccoglie tutte le informazioni che riguardano il mondo aumentato in cui l'utente si immerge, e sono queste informazioni che vengono poi trasmesse al livello di Hologram Engine (implementato in Unity 3D) che si occuperà della renderizzazione sui devices utente. Vi è una decentralizzazione, così, di tutta la logica applicativa (a cui fanno riferimento poi gli ologrammi), che viene portata fuori dalla tecnologia abilitante AR che si utilizza, che prende il ruolo così di puro *strumento*.

Date le premesse, la reattività e la sincronizzazione degli ologrammi, che sono poi il cuore dell'esperienza dell'utente, non presentano alcun lag sensibile dovuto all'ampliamento del modello architetturale del sistema nel suo complesso. Si dimostra così che, allo stato attuale, costruire e utilizzare con successo AWs, seguendo la progettazione proposta in questa tesi, affiancati ad un'implementazione di Hologram Engine con una qualche tecnologia di rendering di ologrammi (possibilmente performante), è possibile con discreti risultati.

Una dimostrazione della demo all'opera di trova in figura 6.5, in cui tutti i tre utenti sono connessi allo stesso AW e percepiscono, attraverso i loro dispositivi, le stesse entità aumentate da angolazioni differenti.

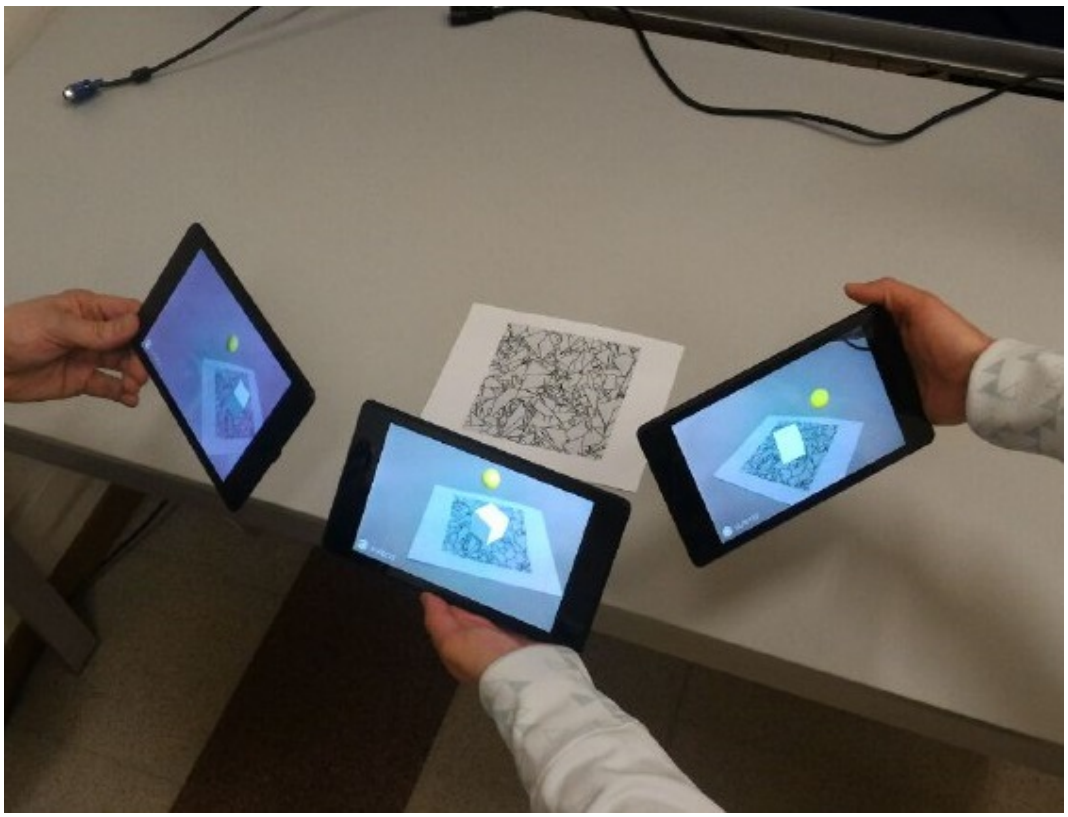


Figura 6.5: Fotografia della Demo in funzione

Conclusioni

Il modello capace di unire AR e IoT che è stato riconosciuto come riferimento definitivo per WoAT è Augmented Worlds. L'interoperabilità è stata raggiunta poi, in fase di progettazione, associando una infrastruttura di AW ad uno strato WoAT attraverso il mapping delle API di modello in chiave REST, accomunando il concetto di AE a quello di Web Thing.

Il prototipo realizzato, in linea con quanto progettato, ha permesso una retrospettiva sulle le problematiche previste in fase di analisi, consentendo di considerarle risolte ed eventualmente facendone emergere di nuove. Al termine del percorso di tesi, e alla luce del framework prototipale sviluppato, si può essere certi che una migrazione coerente con il paradigma WoT di un modello di AW sia possibile (comprendendo tutte le API del modello senza problemi).

Per quanto riguarda invece le problematiche sorte, seppur non centrali nel focus della tesi, sicuramente sarà necessario indagare meglio la parte AR che comprende la gestione degli ologrammi, in quanto allo stato attuale compatibile soltano con Unity 3D. Di possibile ampliazione rimane anche la parte legata ai drivers da applicare alle Smart Things fisiche, che al momento possono essere integrate al sistema, ma l'implementazione dei diversi meccanismi di sincronizzazione sono ancora delegate allo sviluppatore. Questi ed altri punti possono essere di ispirazione per eventuali sviluppi futuri, quali:

- Estendere il prototipo aggiungendo *distribuendo* gli AW. Attualmente tutta l'infrastruttura gira su un singolo nodo, ma i mondi aumentati sono già categorizzati in locali e remoti, per questa eventuale estensione futura.
- Estendere eventualmente (se possibile) il modello infrastrutturale della parte riguardante gli ologrammi, per rendere intercompatibile l'immersione nell'AW alle di tecnologie di AR utilizzate dagli utenti.
- Sviluppare librerie per l'integrazione nel sistemi multi-agente sviluppati attraverso frameworks quali CArtAgO per l'utilizzo di agenti BDI nel-

l'ambito di AW, che possano, ad esempio, mascherare le AEs in artefatti, osservabili poi dai diversi agenti.

La visione di un framework completo consiste in uno scenario in cui gli sviluppatori, una volta a conoscenza del modello di AW, possano scrivere applicazioni capaci di interfacciarsi ad un mondo remoto in modo agile (magari provvedendo delle librerie/componenti opportuni), accedendo ad esso sia attraverso dispositivi abilitanti AR che subito li immergano all'interno, sia attraverso agenti computazionali che ne osservino il modello o ne traccino regioni/entità per un qualche determinato scopo.

Bibliografia

[1] Vandana C. P., Suman Thapa, Pradip Thapa *Web of Things*. International Journal of Scientific Research on Computer Science, Engineering and Information Technology, 2017

[2] Dominique Guinard, Vlad M. Trifa *Building the Web of Things*. Manning Publications, 2016

[3] Andrew Yeh Ching Nee *Augmented Reality - Some Emerging Application Areas*. InTech, 2011

[4] Alan B. Craig *Understanding Augmented Reality*. Morgan Kauffman, 2013

[5] Angelo Croatti, Alessandro Ricci *Towards the Engineering of Agent-Based Augmented Worlds*. 5th International Workshop on Engineering Multi-Agent Systems (EMAS), 2017

[6] Alessandro Ricci, Michele Piunti, Luca Tummolini, Cristiano Castelfranchi *The Mirror World: Preparing for Mixed-Reality Living*. IEEE Pervasive Computing, 2015

[7] Angelo Croatti, Alessandro Ricci *Towards the Web of Augmented Things*. IEEE International Conference on Software Architecture Workshops (ICSAW), April 2017

[8] Angelo Croatti, Alessandro Ricci *Integrating Multi-Agent Systems with Mixed Reality: the Augmented World Approach*. (Unpublished)

[9] Paul Milgram, Haruo Takemura, Akira Utsumi, Fumio Kishino *Augmented Reality: A class of displays on the reality-virtuality continuum*. Telemanipulator and Telepresence Technologies, 1994

[10] M. Cowling, J. Tanenbaum, J. Birt, K. Tanenbaum *Augmenting Reality for Augmented Reality*. Interactions, 2017