Facoltà di Scienze Matematiche Fisiche e Naturali - Corso di Laurea Magistrale in Informatica

Un'architettura per l'osservazione e il controllo di ambienti instrumented

Sintesi della tesi di laurea magistrale di ***Marco Covelli***
matr. 702957, email: m.covelli4@campus.unimib.it / marco.covelli@gmail.com

Supervisori: Dott.ssa Daniela Micucci, Dott. Francesco Fiamberti
Sessione di laurea: marzo 2013

Introduzione

Gli *ambienti instrumented* [4] sono comuni ambienti arricchiti da dispositivi in grado di estrarne informazioni ed agire su di essi. In un'ottica tecnologica, essi costituiscono l'elemento fondante dei *Responsive Environment* [8], sistemi in grado di percepire l'ambiente e di rispondere ad esso ed agli utenti in esso presenti. Tali sistemi richiedono l'integrazione di una molteplicità di dispositivi. Al giorno d'oggi si ha a che fare con un mercato fortemente eterogeneo in termini sia di componentistica (singoli dispositivi) che di soluzioni integrate (e.g., gateway domotici). Ognuno di questi elementi spesso utilizza protocolli di comunicazione proprietari, nell'interesse del produttore che vuole imporli come standard *de facto*.

A causa di questa imperante *eterogeneità*, molti sistemi sono realizzati *ad hoc* per lo specifico contesto applicativo, rimanendo spesso legati a tecnologie e protocolli specifici.

Esistono numerosi contributi per la definizione e la realizzazione di piattaforme tecnologiche che consentono l'integrazione e l'interoperabilità di dispositivi eterogenei, spesso orientati all'offerta di tecnologie abilitanti [7, 9] od alternativamente a soluzioni più vicine a logiche applicative [10, 1]. Nel primo caso, essi offrono meramente una infrastruttura generica per l'unificazione dei meccanismi di comunicazione con i dispositivi; nel secondo, si considerano invece piattaforme più vaste che si specializzano però solo in particolari domini applicativi.

Questa tesi si pone a metà strada tra i due approcci, proponendo un'architettura software che permetta di integrare componentistica hardware eterogenea, rimanendo aperta ed adattabile al dominio applicativo ed offrendo alle applicazioni che la sfruttano delle modalità semplificate di accesso.

Gli obiettivi della tesi sono, quindi, la definizione e la progettazione dell'architettura suddetta e, secondariamente, la realizzazione di una implementazione prototipale ai fini della sua validazione tramite la contestualizzazione rispetto ad un dominio specifico, che coinvolge componentistica disponibile all'interno del laboratorio presso cui è stato svolto questo lavoro.

Le sezioni seguenti descrivono la soluzione architetturale proposta, le scelte implementative per la realizzazione del prototipo, il processo di contestualizzazione in un dominio ed alcune considerazioni legate alla sua applicazione ad un caso di studio.

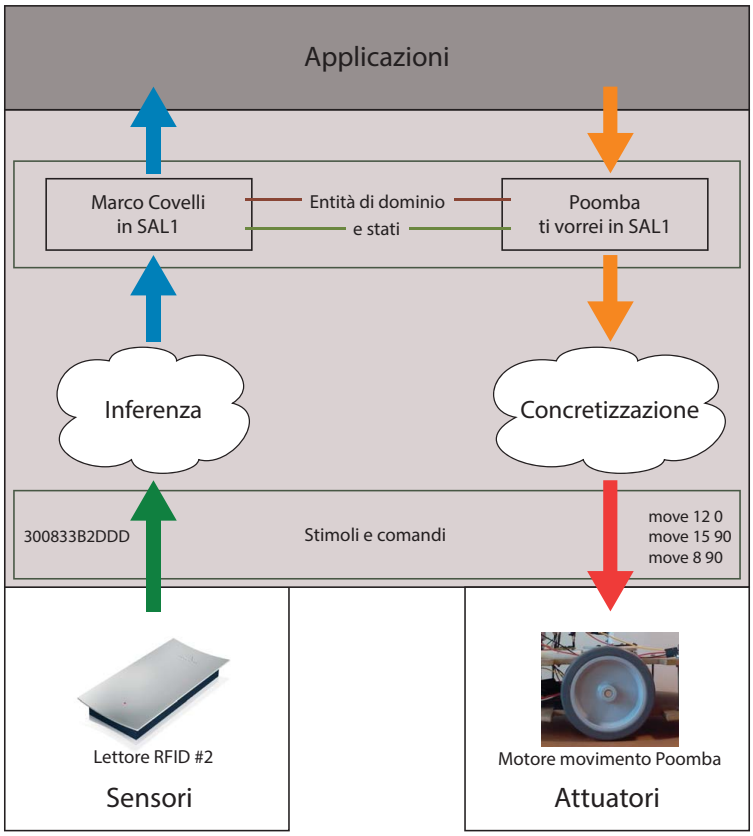

Figura 1: Schema dell'architettura proposta

Architettura proposta

In Figura 1 è illustrato uno schema esemplificativo dell'architettura proposta.

L'architettura abilita la costruzione di applicazioni end-user che interagiscono con il modello dell'ambiente attraverso l'osservazione ed il controllo dello stato di entità di dominio significative.

Un'*entità di dominio* è l'unità elementare costituente il modello. Essa descrive qualsiasi elemento di interesse il cui stato sia inferibile da stimoli sensoriali ed eventualmente modificabile tramite comandi attuabili sull'ambiente. Considerando un dominio domotico, esempi di entità possono essere: lampade, attrezzature come finestre e porte automatiche, ma anche persone e robot.

Ogni entità è composta da una serie di proprietà con un valore associato. Questo insieme descrive in toto l'entità stessa. Ad esempio, nel caso di una lampada, ciò che si vuole rappresentare sono sicuramente il suo stato di accensione e la sua localizzazione spaziale all'interno dell'ambiente. A questo scopo, alcune proprietà modellano *caratteristiche immutabili* delle entità (come la localizzazione della lampada[1]), altre modellano *stati* (come quello di accensione).

Gli stimoli provenienti dai sensori sono sfruttati per inferire stati di entità di dominio, le cui rappresentazioni sono esposte alle applicazioni. Nel flusso inverso, stati desiderati espressi dalle applicazioni si concretizzano in comandi per degli attuatori.

L'astrazione da stimoli e comandi ad entità e relativi stati individua tre livelli distinti: l'interfacciamento diretto ai dispositivi, l'omogenizzazione dei dati grezzi da essi prodotti o consumati e l'astrazione degli stessi in dati di dominio. Ognuno di questi livelli è delegato a componenti software specializzati, che comunicano tramite interfacce ben definite.

[1] In questo caso, l'assunzione è che essa sia fissa.

Partendo dal livello più basso, gli stimoli provenienti dai sensori sono raccolti e quindi poi astratti ad una rappresentazione omogenea da due componenti, *SensingWrapper* e *Stimuli-Translator*. Il primo gestisce la comunicazione a basso livello (mediante tecnologie e protocolli dipendenti dal dispositivo), il secondo traduce gli stimoli sensoriali in un vocabolario condiviso. Questi primi due livelli realizzano concretamente l'integrazione della sensoristica a campo, esponendo quindi degli stimoli astratti che conservano la semantica degli originari, ma che sono espressi secondo una sintassi omogenea. Gli stimoli ambientali così rappresentati sono consumati da specifici componenti (chiamati *StatusGuesser*) per inferire degli stati afferenti ad entità di dominio. Come illustrato in Figura 1, la lettura di un tag RFID permette di inferire che l'entità di dominio Marco Covelli, possessore del tag, si trova nel laboratorio SAL1.

Parlando di inferenza, è bene specificare che il risultato di tale attività possiede intrinsecamente un grado di incertezza che può dipendere da molteplici fattori, come l'affidabilità dei sensori, ma anche l'"audacia" dell'algoritmo di inferenza, che può cercare di inferire stati da informazioni incomplete o di livello molto basso. A ciascun dato prodotto è quindi associato un valore di *confidenza* in base al quale il consumatore dello stesso può valutarne l'attendibilità.

Al termine del flusso, lo stato inferito è usato per aggiornare il modello dell'ambiente da parte di un componente detto *EnvironmentModelManager*.

In un flusso speculare, componenti specializzati (detti *WishReasoner*) prendono in carico stati desiderati espressi su entità e li concretizzano in comandi astratti necessari alla loro realizzazione, espressi secondo una sintassi omogenea ed indipendente dall'attuatore specifico al quale sono inoltrati. Questi sono poi tradotti nella sintassi comprensibile all'attuatore destinatario da uno *AbstractCommandTranslator* ed attuati da un componente denominato *ActuationWrapper*. Sempre citando l'esempio in Figura 1, si desidera avere un'entità di dominio, il robot Poomba[2], nel laboratorio SAL1: tale richiesta è convertita in comandi di movimento attuabili ed inoltrati quindi al motore installato sul robot.

Ad eccezione dell'*EnvironmentModelManager*, i componenti che realizzano questi due flussi costituiscono la parte di architettura che è dipendente dallo specifico dominio applicativo. Secondo un'ottica modulare, l'architettura abilita inoltre alla separazione delle competenze tra componenti afferenti allo stesso livello di astrazione, ognuno dei quali può operare su uno specifico sottoinsieme di entità o di dispositivi.

L'architettura definisce un meccanismo di interazione con le applicazioni basato su messaggi, che permette di formulare richieste inerenti ad entità di dominio senza la necessità di elencarle esplicitamente. Sfruttando un sottoinsieme delle nozioni della *logica dei predicati*, è possibile riferirsi ad entità di dominio tramite le loro proprietà ed i relativi valori. Ciò permette di formulare richieste articolate, definendo dichiarativamente insiemi astratti di entità tramite predicati logici sulle loro proprietà.

Un messaggio di richiesta è tipicamente costituito da un *payload*, che specifica la particolare richiesta, e da un *destinatario*, specificato tramite un predicato logico che descrive le proprietà delle entità a cui la richiesta è indirizzata. Ad un livello concettuale astratto, l'approccio è quello di indirizzare i messaggi di richiesta direttamente alle entità di dominio, che rispondono singolarmente nel merito. Gli eventuali messaggi di risposta sono infatti caratterizzati sempre da un *payload*, che contiene le informazioni richieste, e da un *mittente* che individua l'entità a cui fanno riferimento le informazioni stesse.

Tramite questo livello di espressività, l'architettura abilita le applicazioni a:

- manifestare interesse per cambiamenti di stato di insiemi astratti di entità, ottenendo una notifica all'occorrenza di tali cambiamenti;
- interrogare l'ambiente, ottenendo proattivamente informazioni puntuali su proprietà di entità;

[2]Unità mobile progettata ed implementata presso il Laboratorio di Architetture Software (SAL) dell'Università di Milano-Bicocca

- esprimere stati desiderati su entità.

Nel primo caso, ad esempio, è possibile manifestare interesse per tutti i futuri cambiamenti di stato di accensione delle luci poste in un determinato locale dell'ambiente, senza doverle elencare esplicitamente. Nel secondo, è possibile interrogare l'ambiente, chiedendogli di fornire il nome delle persone presenti in una stanza al momento della richiesta. Nel terzo, si può richiedere che tutte le luci siano spente.

Per ognuna di queste modalità di interazione sono coinvolti uno o più componenti software di mediazione, come di seguito descritto.

L'interesse a cambiamenti di stato è concretizzato da messaggi di richiesta di sottoscrizione, inviati ad un *SubscriptionRequestHandler*, che gestisce anche le eventuali de-sottoscrizioni. Le relative notifiche saranno invece gestite da un componente denominato *StatusChangeNotifier* che, "catturando" gli stati inferiti dagli *StatusGuesser*, li inoltra tramite messaggi alle applicazioni interessate.

Per interrogare l'ambiente, le applicazioni si interfacciano con un componente detto *ObservationRequestHandler*, che riceve messaggi di richiesta di osservazione e, interrogando il modello dell'ambiente, risponde nel merito.

La gestione di stati desiderati su entità è invece gestita da uno *WishRequestHandler*, a cui sono indirizzati messaggi di richiesta che saranno quindi inoltrati ai *WishReasoner* opportuni.

Implementazione

Le scelte implementative per la realizzazione del prototipo sono state effettuate allo scopo di costruire rapidamente un prototipo semplice per la validazione architetturale della soluzione, sfruttando perlopiù conoscenze specifiche dell'autore di questa tesi. Sviluppi futuri potrebbero individuare tuttavia tecnologie più efficaci ed efficienti. L'intera implementazione è stata realizzata utilizzando il linguaggio di programmazione Java e tecnologie affini.

Le scelte di deployment e di implementazione hanno portato ad una piattaforma distribuita che utilizza *web service* [2] come punti di accesso verso le applicazioni.

Per la realizzazione concreta di gran parte dei flussi informativi tra i componenti è stato scelto il framework *Space Integration Services* (*SIS*) [3], che consente di scambiare informazioni utilizzando un paradigma *publish/subscribe* secondo una metafora basata su *spazi multipli*. Il framework in questione fornisce inoltre un'infrastruttura per la rappresentazione di *ambienti spaziali* ed altre informazioni relazionali.

Le implementazioni dei componenti dell'architettura sono suddivise in due categorie: quelle dei componenti *generici* (indipendenti dallo specifico scenario applicativo) e quelle dei componenti *specifici* (dipendenti dalle scelte in termini di entità rappresentate e dispositivi a campo).

I componenti monolitici, individuati in fase di progettazione, fanno parte della prima categoria. Essi sono l'*EnvironmentModelManager*, l'*ObservationRequestHandler*, il *SubscriptionRequestHandler*, il *WishRequestHandler* e lo *StatusChangeNotifier*.

L'*EnvironmentModelManager* è stato implementato usando un approccio ibrido che integra il già citato *SIS*, nella sua accezione di repository di ambienti spaziali, ed il motore a regole *Jess* [6]. Ciò abilita agilmente alla rappresentazione dell'ambiente in termini di entità come composizione di proprietà ed all'interazione con lo stesso mediante la logica dei predicati. L'interfaccia verso il componente in questione, che è centrale all'interno dell'architettura (è interrogato sia dai gestori delle richieste che, eventualmente, da *StatusGuesser* e *WishReasoner*), è poi esposta al resto dei componenti come *web service*.

I tre gestori di richieste sono anch'essi esposti come web service (nella fattispecie, in un unico *endpoint*, detto *RequestEndpoint*), mentre il notificatore di cambiamenti di stato, per vincoli tecnologici, va ad integrarsi in parte con l'*EnvironmentModelManager* ed utilizza un meccanismo basato sul protocollo *WebSocket* [5] per la comunicazione asincrona con le applicazioni.

I componenti concreti restanti dipendono dal dominio applicativo, ovvero devono essere realizzati considerando le tipologie delle entità rappresentate e degli stimoli e dei comandi disponibili per l'ambiente considerato. Per questi componenti, a questo livello, possono solo essere definiti dei semplici requisiti di riferimento che consentono la comunicazione con il resto del prototipo. Tali requisiti si riducono alla definizione delle modalità di interazione con SIS. La prossima sezione tratta brevemente il processo di contestualizzazione in un dominio che coinvolge, a vari livelli, la loro implementazione ed integrazione al fine di ottenere una piattaforma specifica ed effettivamente fruibile da applicazioni end-user.

L'implementazione interna di questi ultimi componenti non è vincolata, in generale, da scelte specifiche. Una soluzione consigliata come best-practice consiste nell'accoppiamento di ogni *SensingWrapper* con un opportuno *StimuliTranslator* in un unico *SensingComponent* e, analogamente, di ogni *AbstractCommandTranslator* con un *ActuationWrapper* in un unico *ActuationComponent*. Ciò significa che le due tipologie di componenti risultanti si occuperanno sia dell'interfacciamento con i dispositivi che dell'astrazione di stimoli e comandi. Questa scelta è motivata dal fatto che nei dispositivi presenti sul mercato spesso sia i meccanismi di interfacciamento che le sintassi protocollari cambiano da un produttore all'altro.

Contestualizzazione in un dominio

Il processo di contestualizzazione dell'architettura rispetto ad un dominio applicativo è suddivisibile in due sotto-processi. Un primo, di tipo concettuale, definisce i modelli delle entità per quel particolare dominio applicativo ed individua le tipologie di stimoli e comandi astratti supportati, implementando inoltre i componenti che operano le inferenze e le concretizzazioni (*StatusGuesser* e *WishReasoner*). Un secondo sotto-processo, di carattere tecnologico, implementa invece i componenti che comunicano direttamente con gli specifici dispositivi eterogenei ed espongono gli stimoli ed i comandi astratti sopracitati, cioè i *SensingComponent* e gli *ActuationComponent*. Seguendo questa suddivisione, si ottiene, al primo stadio, una soluzione che è ancora indipendente dai dettagli strettamente tecnologici e quindi riusabile a fronte di differenti dispositivi; questa è poi ulteriormente specializzata per sfruttare la componentistica hardware disponibile per una sua istanziazione in un ambiente reale.

Completato il processo, l'implementazione è quindi configurabile per essere utilizzata concretamente.

Il risultato così ottenuto abilita alla costruzione di applicazioni che si appoggiano all'architettura per realizzare logiche di dominio.

Quanto esposto è stato applicato al prototipo, realizzando un'effettiva contestualizzazione in un dominio reale, allo scopo di validare la soluzione finale dal punto di vista architetturale con un'applicazione di studio. Nel dominio in questione è possibile monitorare la localizzazione di personale riconosciuto, osservare e controllare quella di unità mobili e, allo stesso modo, lo stato di accensione di lampade. Quanto ottenuto è stato adeguatamente configurato per essere applicato all'interno dei laboratori presso cui è stato svolto il lavoro di tesi, utilizzando i dispositivi che ne arricchiscono i locali.

Caso di studio

L'applicazione di studio realizza alcuni comportamenti che rispondono a particolari condizioni ambientali. Utilizzando una logica reattiva, essa realizza strategie di accensione e spegnimento delle luci di un laboratorio a fronte della presenza o meno di persone all'interno dello stesso; allo stesso modo, all'ingresso di un professore, colloca un'unità mobile nello stesso locale.

Lo scopo della sperimentazione è stato dimostrare l'effettivo accesso semplificato all'ambiente introdotto dalla soluzione architetturale.

La progettazione e l'implementazione dell'applicazione di studio hanno fatto rilevare una estrema semplicità di sviluppo legata sia all'esiguità dei requisiti necessari per interagirvi che alla possibilità di sviluppare logiche applicative flessibili. La conoscenza necessaria all'interazione con l'architettura si riduce all'invio ed alla ricezione di messaggi ben strutturati, contenenti dati che si riferiscono unicamente ai modelli di entità definiti e noti alle applicazioni. L'utilizzo dei modelli di entità e di un meccanismo di interazione legato ai predicati logici sugli stessi ha permesso inoltre di adattare facilmente le applicazioni a configurazioni ambientali differenti.

Il poter ragionare su dati di dominio ha inoltre reso meno complessa l'implementazione della logica applicativa, riducendola alla semplice applicazione di regole.

Da un punto di vista funzionale, non ci sono stati particolari problemi: le strategie sono state applicate in maniera corretta e senza problemi riconducibili al modello architetturale.

Conclusioni

In questa tesi è stata definita e progettata un'architettura che permette l'integrazione di componentistica hardware eterogenea al fine di offrire alle applicazioni end-user una rappresentazione dell'ambiente al livello di astrazione desiderato. Questo modello è esportato alle applicazioni che possono quindi osservare e controllare l'ambiente ragionando solo su informazioni di dominio.

Il risultato è un'architettura indipendente dal dominio applicativo, fortemente modulare ed aperta. Essa infatti non è progettata per uno scenario specifico, ma definisce precisi livelli di astrazione, nei quali sono collocabili componenti ben definiti che la contestualizzano in un dominio applicativo. In un'ottica modulare, questi ultimi sono caratterizzati da una elevata indipendenza concettuale ed hanno interfacce ben definite, che specificano la struttura dei dati da trattare e le modalità di comunicazione con il resto dell'architettura. Ciò abilita all'apertura della soluzione, poiché favorisce l'aggiunta di componenti che aderiscono alle interfacce e realizzano i flussi di astrazione necessari, rendendo semplice il supporto incrementale a nuovi dispositivi e modelli di entità.

L'implementazione di un caso di studio ha inoltre dimostrato l'effettiva semplificazione nell'accesso all'ambiente da parte delle applicazioni. In particolare, si è potuto rilevare come la soluzione architetturale permetta di separare nettamente le strategie applicative (di competenza delle applicazioni) dalle modalità interazionali con cui esse percepiscono e modificano l'ambiente (aspetti completamente gestiti dall'architettura).

Sviluppi futuri includeranno l'individuazione di una soluzione al problema dell'aging degli stati delle entità. Tale problematica, individuata in fase di analisi, riguarda l'aggiornamento del livello di confidenza delle proprietà per cui tale attributo decade in assenza di stimoli sensoriali.

L'implementazione realizzata costituisce solo un prototipo per la validazione architetturale della soluzione proposta. Sarà quindi necessario realizzare un'implementazione più robusta che la elevi a piattaforma distribuibile presso i diversi laboratori del Dipartimento presso cui è stato svolto questo lavoro. Ciò potrà essere utile per verificare la validità dell'architettura su scala più grande, applicandola a scenari e casi di studio più complessi di quelli presentati in questa tesi. A questo scopo sarà importante analizzare l'implementazione risultante anche sotto il profilo prestazionale, attività non approfondita in questa sede.

Riferimenti bibliografici

[1] M. Aiello e S. Dustdar. Are our homes ready for services? A domotic infrastructure based on the web service stack. *Pervasive and Mobile Computing*, 4(4):506–525, 2008.

[2] G. Alonso, F. Casati, H. Kuno, e V. Machiraju. *Web services: concepts, architectures and applications*. Springer, 2003.

[3] D. Bernini, F. Fiamberti, D. Micucci, e F. Tisato. Architectural abstractions for spaces-based communication in smart environments. *Journal of Ambient Intelligence and Smart Environments*, 4(3):253–277, 2012.

[4] A. Butz e A. Krüger. A generalized peephole metaphor for augmented reality and instrumented environments. In *Proceedings of The International Workshop on Software Technology for Augmented Reality Systems (STARS)*, 2003.

[5] I. Fette e A. Melnikov. The WebSocket protocol. RFC 6455, IETF, 2011.

[6] E. Friedman-Hill. Jess, the Java Expert System Shell. Relazione Tecnica SAND–98-8206, Sandia Labs., Livermore, CA (United States), 1997.

[7] J. Kusznir e D.J. Cook. Designing lightweight software architectures for smart environments. In *2010 Sixth International Conference on Intelligent Environments (IE)*, pp. 220 –224. IEEE, 2010.

[8] N. Negroponte. *Soft architecture machines*. MIT Press, 1975.

[9] H. Ristau. Publish/process/subscribe: Message based communication for smart environments. In *2008 IET 4th International Conference on Intelligent Environments*, pp. 1–7, 2008.

[10] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R.H. Campbell, e K. Nahrstedt. A middleware infrastructure for active spaces. *Pervasive Computing, IEEE*, 1(4):74 –83, 2002.