

---

# Università degli Studi di Napoli “Federico II”



FACOLTÀ DI INGEGNERIA  
CORSO DI LAUREA IN INGEGNERIA  
INFORMATICA

## **TESI DI LAUREA** **Metodologie e politiche di gestione dei servizi in** **UbiSystem**

RELATORI  
Ch.mo Prof. Antonio d'Acerno  
Ch.mo Prof. Giuseppe De Pietro

CANDIDATO  
Raffaella Laraia  
Matr.: 041/2479

CO-RELATORE  
Ch.mo Ing. Esposito Massimo

ANNO ACCADEMICO 2003/2004

---

---

# INDICE

<b>INDICE .....</b>	<b>I</b>
<b>INTRODUZIONE .....</b>	<b>VI</b>
<b>I. IL PERVASIVE COMPUTING .....</b>	<b>1</b>
I.1 INTRODUZIONE .....	1
I.2 L'EVOLUZIONE VERSO IL PERVASIVE COMPUTING .....	4
I.2.1 SISTEMI DISTRIBUITI .....	4
I.2.2 SISTEMI MOBILI .....	4
I.2.3 SISTEMI PERVASIVI .....	5
I.3 BACKGROUND E STATO DELL'ARTE .....	6
I.3.1 AURA .....	7
I.3.1.1 Architettura di AURA .....	8
I.3.1.2 Architettura di un AURA Client .....	9
I.3.1.3 AURA a Lavoro .....	10
I.3.2 GAIA: UN MIDDLEWARE PER PERVASIVE ENVIRONMENTS .....	11
I.3.3 COOLTOWN.....	14
I.3.4 CENTAURUS.....	17
I.3.5 PROGETTO CoBRA.....	19
I.4 CARATTERISTICHE DI UN SISTEMA PERVASIVO .....	21
I.4.1 CONTEXT INFORMATION E RUN-TIME ADAPTATION .....	22
I.4.2 TASK RECOGNITION E PRO-ACTIVITY .....	23

---

I.4.3 RESOURCE ABSTRACTION E DISCOVERY .....	24
I.4.4 ETEROGENEITY E SERVICE UI ADAPTATION .....	24
I.4.5 SECURITY E PRIVACY .....	25
I.4.6 FAULT-TOLERANCE E SCALABILITY .....	25
<b>II. TECNOLOGIE PER IL PERVASIVE COMPUTING .....</b>	<b>27</b>
II.1 LA COMUNICAZIONE NEI SISTEMI DISTRIBUITI .....	27
II.2 PARADIGMI DI COMUNICAZIONE .....	28
II.2.1 REMOTE PROCEDURE CALL .....	28
II.2.2 XML-BASED RPC .....	29
II.2.3 REMOTE METHOD INVOCATION.....	30
II.3 MIDDLEWARE ED INFRASTRUTTURE DI COMUNICAZIONE .....	31
II.3.1 CORBA.....	31
II.3.2 JINI .....	36
II.3.2.1 Discovery .....	37
II.3.2.2 Join .....	37
II.3.2.3 Lookup.....	38
II.3.3 WEB SERVICES .....	39
II.3.3.1 WSDL.....	40
II.3.3.2 UDDI.....	41
II.4 LA COMUNICAZIONE IN AMBIENTI PERVASIVI .....	44
II.4.1 L'INTEROPERABILITÀ: ESIGENZA DI NUOVI STRUMENTI.....	45
II.5 WEB SEMANTICO .....	46
II.5.1 ARCHITETTURA E LINGUAGGI DEL WEB SEMANTICO .....	47
II.5.2 XML E XML SCHEMA .....	48
II.5.3 RDF E RDF SCHEMA .....	50
II.5.4 DAML+OIL.....	52
II.5.4.1 Sistemi basati su Frame .....	53
II.5.4.2 Description Logics.....	54
II.5.5 OWL .....	55
<b>III. MODELLO ARCHITETTURALE .....</b>	<b>57</b>

---

---

III.1 INTRODUZIONE.....	57
III.2 REQUISITI DEL SISTEMA .....	57
III.3 LO SCENARIO E GLI ATTORI .....	58
III.3.1 PERCHÉ I WEB SERVICES .....	59
III.4 IL MODELLO ARCHITETTURALE .....	60
III.5 TASSONOMIA DEI SERVIZI IN UBISYSTEM.....	62
III.6 SYSTEM SERVICES MANAGER.....	64
III.6.1 SYSTEM SERVICES.....	65
III.6.1.1 DHCP Service .....	65
III.6.1.2 Ontology Service .....	66
III.6.1.3 Asynchronous Communication Service .....	67
III.6.1.4 Location Service .....	68
III.6.1.5 Context Service .....	69
III.6.1.6 User Manager Service .....	71
III.6.1.7 Session Manager .....	72
III.7 APPLICATION SERVICES MANAGER .....	73
III.7.1 APPLICATION SERVICES .....	74
III.7.1.1 Servizio di stampa localizzata .....	74
III.7.1.2 Servizio di Mp3 JukeBox.....	75
III.7.1.3 Servizio di musica d’ambiente .....	75
III.7.1.4 Servizio di videoproiezione .....	75
III.7.1.5 Servizio di streaming .....	75
III.8 UBIQUITOUS GATEWAY .....	75
<b>IV. COMPONENTI IMPLEMENTATI.....</b>	<b>77</b>
IV.1 INTRODUZIONE .....	77
IV.2 BUSINESS CLASS DIAGRAMS .....	78
IV.2.1 UBIQUITOUS GATEWAY .....	78
IV.2.2 APPLICATION SERVICES MANAGER.....	80
IV.2.3 SYSTEM SERVICES MANAGER.....	82
IV.3 SEQUENCE DIAGRAMS .....	83
IV.3.1 REGISTRAZIONE DI UN SERVIZIO DI SISTEMA .....	84
IV.3.2 UTILIZZO DI UN SERVIZIO DI SISTEMA .....	85

---

---

IV.3.3 REGISTRAZIONE DI UN SERVIZIO APPLICATIVO.....	86
IV.3.4 UTILIZZO DI UN SERVIZIO APPLICATIVO.....	87
IV.3.5 REGISTRAZIONE DI UN DEVICE.....	88
IV.3.6 REGISTRAZIONE DI UN SERVIZIO OFFERTO DALL'UTENTE.....	89
IV.3.7 RICHIESTA DELLA LISTA DEI SERVIZI DA PARTE DELL'AMMINISTRATORE .....	90
IV.3.8 RICHIESTA DELLA LISTA DEI SERVIZI APPLICATIVI.....	91
IV.3.9 RILASCIO DI UN SERVIZIO APPLICATIVO .....	93
IV.3.10 LOGOUT.....	93
IV.3.11 ABBANDONO DELL'AMBIENTE SENZA LOGOUT .....	94
IV.3.12 COMPLETAMENTO DEI BUSINESS CLASS DIAGRAMS .....	96
<b>IV.4 CLASS DIAGRAMS.....</b>	<b>96</b>
IV.4.1 UBIQUITOUS GATEWAY .....	96
IV.4.2 APPLICATION SERVICES MANAGER.....	97
IV.4.3 SYSTEM SERVICES MANAGER.....	99
<b>IV.5 DETTAGLI IMPLEMENTATIVI .....</b>	<b>100</b>
IV.5.1 IMPLEMENTAZIONE DI UN OGGETTO CORBA.....	100
IV.5.2 UBIQUITOUS GATEWAY .....	103
IV.5.2.1 UG come Oggetto CORBA.....	103
IV.5.2.2 Registrazione del Dispositivo Client .....	106
IV.5.2.3 UG come Web-Service.....	112
IV.5.3 APPLICATION SERVICES MANAGER.....	119
IV.5.4 SYSTEM SERVICES MANAGER.....	123
<b>V. SERVIZI APPLICATIVI .....</b>	<b>127</b>
V.1 INTRODUZIONE.....	127
V.2 DESCRIZIONE FUNZIONALE DEI SERVIZI.....	128
V.2.1 MP3 JUKEBOX .....	128
V.2.2 PDFVIEWER.....	129
V.2.3 LOCAL PRINT SERVICE.....	130
V.2.4 MUSICA D'AMBIENTE .....	131
V.3 IMPLEMENTAZIONE DEI SERVIZI.....	132
V.3.1 IMPLEMENTAZIONE DI UN OGGETTO CORBA .....	132
V.3.1.1 Registrazione di un Servizio Applicativo .....	133

---

---

V.3.2 WRAPPING DI UN SERVIZIO.....	145
<b>VI. ESEMPIO D'USO.....</b>	<b>148</b>
VI.1 LO SCENARIO IN CUI OPERANO I SERVIZI .....	148
VI.1.1 LO START-UP DI UBISYSTEM .....	150
VI.1.2 L'INTERFACCIA UTENTE.....	151
VI.2 UBISYSTEM IN AZIONE.....	152
VI.2.1 AUTENTICAZIONE DI UN UTENTE.....	154
VI.2.2 IL SERVIZIO PDF VIEWER .....	156
VI.2.3 IL SERVIZIO MP3 PLAYER .....	159
VI.2.4 IL SERVIZIO DI STAMPA LOCALE .....	160
VI.2.5 AUTENTICAZIONE DELL' AMMINISTRATORE .....	164
VI.2.6 UTILIZZO DEI SERVIZI DI SISTEMA .....	166
VI.2.6.1 Session Manager.....	166
VI.2.6.2 User Manager Service.....	167
VI.2.7 INFORMAZIONI RELATIVE AD UN SERVIZIO APPLICATIVO.....	179
VI.2.8 MESSAGGI D'ERRORE .....	181
<b>CONCLUSIONI E SVILUPPI FUTURI.....</b>	<b>182</b>
<b>BIBLIOGRAFIA .....</b>	<b>184</b>

## INTRODUZIONE

I recenti progressi nelle tecnologie hardware e di comunicazione stanno lentamente cambiando il modo di concepire l'elaborazione informatica e le modalità d'interazione fra l'uomo ed i computer. In breve, ci troveremo a vivere in ambienti popolati da una serie di dispositivi "intelligenti" che comunicano e cooperano tra di loro per assisterci nello svolgimento delle nostre comuni attività.

A questo tipo di scenari ci si riferisce quando si parla di *Ubiquitous* (o *Pervasive*) *Computing*: scenari in cui, capacità di calcolo e di elaborazione assumono caratteristiche di ubiquità, si diffondono nella nostra realtà e penetrano all'interno degli oggetti che ci circondano.

Piuttosto che un fenomeno tecnologico, *l'Ubiquitous Computing* è una visione che ci spinge a riconsiderare il ruolo dei computer e della computazione nel nostro mondo quotidiano. E' una visione alla ricerca di una qualità fondamentale nuova dell'utilizzo del calcolatore.

All'interno dei laboratori del CNR è stato definito un modello architetturale, ed implementato un prototipo battezzato *UbiSystem*, di sistema pervasivo, con il fine di realizzare un ambiente dinamico e scalabile per l'erogazione, la ricerca e la fruizione di servizi.

L'obiettivo di questa tesi è l'espansione del suddetto prototipo di *UbiSystem*.

La tesi è così articolata: nel capitolo 1 vengono introdotti i concetti di *Pervasive* ed *Ubiquitous Computing*, discutendo problematiche, aspetti innovativi e stato dell'arte; nel capitolo 2 vengono descritte le principali tecnologie che sono alla base del *Pervasive Computing*, in particolar modo, quelle impiegate per lo sviluppo della tesi; nel capitolo 3 viene descritto il modello architetturale di sistema pervasivo

proposto; nel capitolo 4 viene illustrato il progetto dei moduli implementati e i dettagli implementativi; nel capitolo 5 vengono descritte le funzionalità dei servizi esistenti e le modifiche apportate per integrarli con i nuovi componenti; nel capitolo 6 viene illustrato il funzionamento dell'intero sistema; nelle Conclusioni e sviluppi futuri, infine, vengono discussi i risultati ottenuti e prospettate le possibili evoluzioni del sistema.

Il presente lavoro è stato sviluppato presso l'ICAR-CNR sezione Napoli nei laboratori di San Giorgio (NA).

# I. IL PERVASIVE COMPUTING

## I.1 INTRODUZIONE

Era il 1991 quando Mark Weiser, direttore scientifico delle ricerche tecnologiche allo Xerox PARK, in un suo articolo d'avanguardia, utilizzò per la prima volta il termine *Ubiquitous Computing* [36]. Nel suo articolo, Weiser annunciava un cambiamento nel modo di concepire l'elaborazione automatica e descriveva scenari in cui i computer, onnipresenti, entravano sempre più a far parte della vita di tutti i giorni.

I *computer* ed il *computing* in generale, infatti, stanno lentamente ed inesorabilmente navigando verso nuovi paradigmi. Negli anni sessanta, alla parola "*computer*" venivano associati grandi e costosi *mainframe*, caratterizzati da un grosso numero di utenti che ne condividevano le risorse. Si parlava di paradigma "*many people per computer*": molti utenti per una sola macchina. Il progresso tecnologico ha poi consentito la realizzazione dei *personal computer* che hanno significativamente modificato il tipo di utilizzo dei sistemi di calcolo, trasformando il paradigma in "*one person per computer*": ogni persona poteva disporre di un proprio calcolatore. Nell'ultimo decennio, la diffusione di *laptop*, *Personal Digital Assistant* (PDA), telefoni cellulari multifunzione, dispositivi portatili dotati di microprocessori e di capacità di immagazzinare dati, ha mutato ulteriormente il rapporto uomo-computer, aprendo le porte all'era dei "*many computers per person*": tanti elaboratori per una singola persona [31].

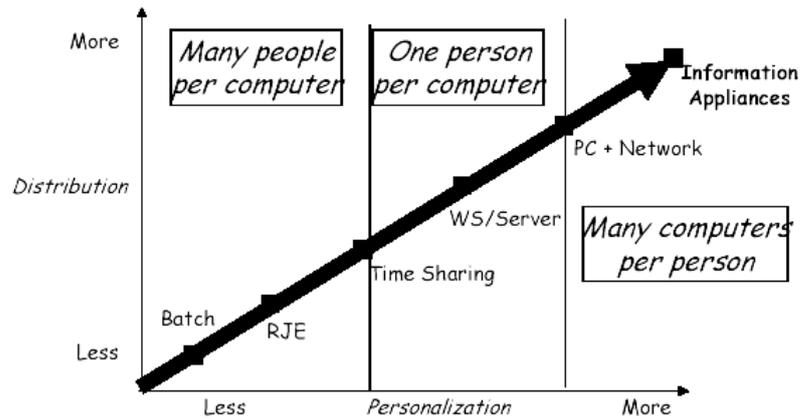


Fig. I.1 L'evoluzione del computing dai mainframes ad oggi.

Il proliferare di questi dispositivi intelligenti, sempre più piccoli e meno costosi, insieme con i progressi delle tecnologie di comunicazione, ha indotto Mark Weiser ad immaginare un futuro, non troppo lontano, nel quale i computer entrano a far parte integrante di ogni oggetto della realtà quotidiana, favorendo la comunicazione e l'elaborazione di informazioni in maniera naturale: *everywhere, all the time*.

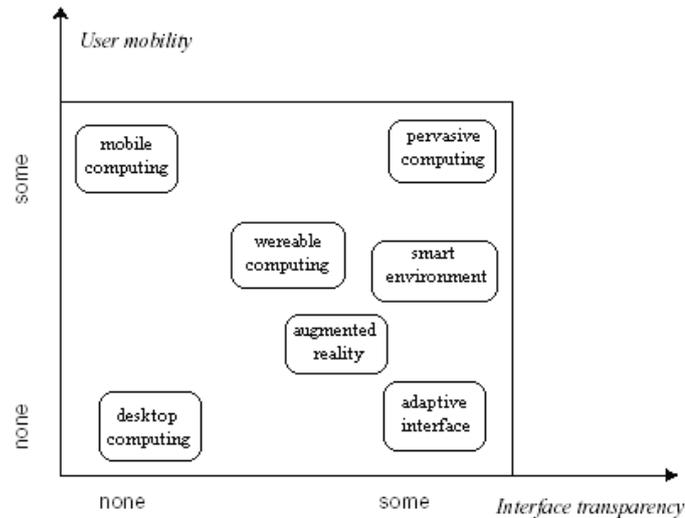
L'essenza della sua idea era *"the creation of environments saturated with computing and computational capability, yet gracefully integrated with human users"* [32].

Un sistema di Ubiquitous Computing è caratterizzato da due attributi fondamentali [22]:

- *Ubiquità*: l'interazione con il sistema è disponibile dovunque l'utente ne abbia bisogno;
- *Trasparenza*: il sistema non è intrusivo ed è integrato negli ambienti della vita quotidiana.

In accordo con questa visione è possibile identificare due dimensioni che forniscono una più chiara definizione degli *ubiquitous system* ed esprimono le relazioni esistenti con le altre aree di ricerca emergenti:

- *Mobilità dell'utente*: esprime la libertà che l'utente ha di muoversi quando interagisce con il sistema;
- *Trasparenza di interfaccia*: riflette lo sforzo consapevole e l'attenzione che il sistema richiede all'utente, sia per operare su di esso che per percepirne i suoi output.



**Fig. I.2** Attributi di un sistema pervasivo.

Quindi, nell’ottica di Weiser, l’*Ubiquitous Computing* mira alla realizzazione di un mondo non più vincolato alle scrivanie, ma composto da ambienti dotati di capacità computazionali e comunicative, talmente integrati con l’utente da diventare una “tecnologia che svanisce” [32], utilizzabile in maniera trasparente ed inconscia. Non una realtà virtuale, in cui le persone sono inserite in un mondo generato dai computer, ma piuttosto una virtualità reale che porta i computer a vivere nel mondo reale, insieme con le persone [36].

Tuttavia, gli scenari dipinti da Weiser 13 anni fa erano anacronistici; la tecnologia hardware necessaria per la loro realizzazione semplicemente non esisteva. E così, i tentativi compiuti allo Xerox PARC fallirono.

Dopo diversi anni, i recenti sviluppi tecnologici hanno dato nuovo impulso alle ricerche sull’*Ubiquitous Computing*, di recente ribattezzato anche col nome di *Pervasive Computing*, per suggerire il carattere pervasivo con cui l’“intelligenza elaborativa” si diffonde e si manifesta negli oggetti che ci circondano.

Probabilmente, i tempi non sono ancora maturi e la visione di Weiser resta ancora futuristica: “*The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it*”. Di sicuro, però, oggi ci troviamo in una posizione migliore, rispetto a quella dei ricercatori della Xerox del 1991, per affrontare le problematiche relative all’*Ubiquitous, o Pervasive, Computing*.

Da qualche anno, di fatto, in diverse Università e Centri di Ricerca del mondo sono sorte numerose iniziative che seguono la direzione tracciata da Weiser; ciascun progetto affronta i vari problemi da differenti punti di vista, talvolta contrastanti, ma tutti sono impegnati nello sforzo comune di rendere il *Pervasive Computing* una realtà.

## **I.2 L'EVOLUZIONE VERSO IL PERVASIVE COMPUTING**

Il Pervasive Computing può essere visto come una nuova forma di computazione altamente dinamica e disaggregata: gli utenti sono mobili e i servizi sono forniti da una serie di componenti distribuiti che collaborano tra loro [5]. Le applicazioni necessarie per supportare queste nuove esigenze sono, da un punto di vista architeturale, non monolitiche bensì costituite da moduli allocati in numerosi nodi della rete.

In tal senso, il *Pervasive Computing* costituisce una nuova tappa nel percorso evolutivo dell'elaborazione e del calcolo distribuito [32].

### **I.2.1 SISTEMI DISTRIBUITI**

Il *Distributed Computing* mira a ripartire dati e capacità computazionali su componenti indipendenti, residenti su macchine diverse, che comunicano tra loro attraverso reti di interconnessione.

Le metodologie attraverso le quali comunicare ed accedere ai dati e agli strumenti di calcolo da postazioni remote, le tecniche sulla replicazione e ridondanza dei dati, che favoriscono la disponibilità e la reperibilità delle informazioni e aumentano l'affidabilità del sistema nel complesso, rappresentano l'oggetto di studio per questa forma di computing.

### **I.2.2 SISTEMI MOBILI**

Con l'introduzione di vincoli e problematiche legate al concetto di mobilità, è stato necessario pensare a nuove soluzioni tecnologiche che hanno portato alla creazione di una nuova forma di *computing*, ossia il *Mobile Computing* [24].

In questi nuovi scenari di calcolo distribuito, non si hanno più nodi di rete fissi, con connessioni stabili e veloci, ma nodi costituiti da dispositivi mobili che accedono alla rete e la abbandonano continuamente ed in maniera del tutto imprevedibile, dotati di connessioni precarie e contraddistinte da forti cambiamenti sulle caratteristiche di banda.

Le limitate capacità di calcolo e di memoria dei dispositivi mobili, le esigenze di risparmio energetico, rappresentano ulteriori aspetti di cui il *Mobile Computing* si sta occupando.

### I.2.3 SISTEMI PERVASIVI

I sistemi di *Pervasive Computing* sono a loro volta anche sistemi distribuiti e mobili. Pertanto, le problematiche inerenti il *mobile* ed il *distributed computing* vengono riprese in questo nuovo paradigma ma, in certo senso, amplificate oltremodo a causa dei requisiti stringenti e dei particolari contesti definiti in questi nuovi ambienti.

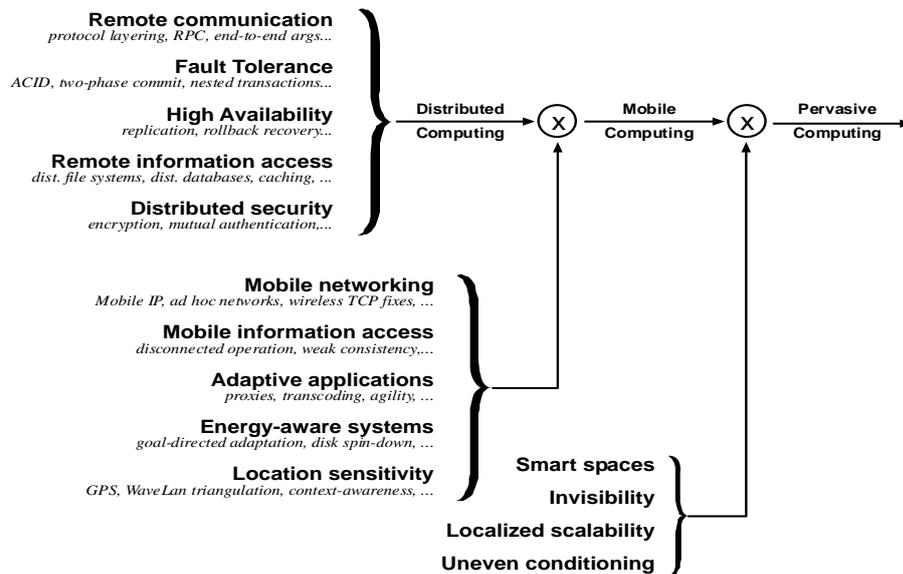
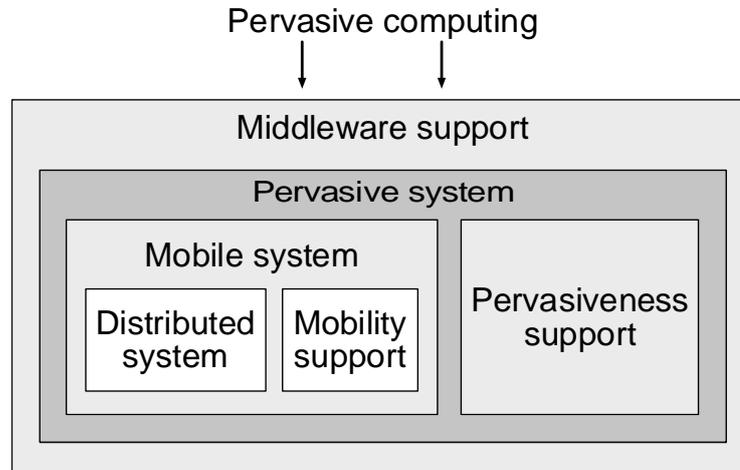


Fig. I.3 Dal Distributed Computing al Pervasive Computing.

La figura I.3 mostra come i problemi di ricerca nel *Pervasive Computing* sono relazionati a quelli del *Distributed* e del *Mobile Computing*. Muovendoci da sinistra verso destra, incontriamo nuove incognite. Inoltre, le soluzioni di problematiche precedentemente affrontate diventano ancora più complesse: come suggeriscono i simboli in figura, l'aumento di complessità è moltiplicativo piuttosto che additivo. E'

molto più difficile progettare e realizzare un sistema pervasivo che un sistema distribuito di comparabile robustezza e maturità [32].



**Fig. I.4** Un modello architetturale per sistemi pervasivi.

Concludiamo questo paragrafo con la figura I.4 che riassume quanto finora detto, illustrando come si compone l'architettura di un sistema pervasivo a partire da sistemi mobili e distribuiti [30].

### I.3 BACKGROUND E STATO DELL'ARTE

I numerosi progetti realizzati in tutto il mondo per implementare un'infrastruttura in grado di trasformare un semplice ambiente in un ambiente di *ubiquitous computing* evidenziano chiaramente l'ingente impiego di tempo e di risorse che è stato profuso negli ultimi anni.

Ognuna di queste iniziative ha focalizzato la propria attenzione su uno specifico contesto con determinati applicazioni e servizi (ad esempio abitazioni, uffici, *meeting-room*), essendo ancora irrealizzabile l'idea di sviluppare un'unica infrastruttura di tipo generale capace di essere usata in tutti gli ambienti esistenti.

Di seguito sono analizzati, singolarmente, gli esempi più significativi di sistemi pervasivi implementati, evidenziando, per ciascuno di essi, le caratteristiche principali e i modelli architetturali.

### I.3.1 AURA

Aura [27], il progetto seguito alla Carnieg Mellon University in Pittsburgh, mira a fornire all'utente un ambiente di *computing* "libero da distrazioni" (*distraction free ubiquitous computing*), dove le persone possono accedere ai servizi o svolgere le proprie attività senza interventi sul sistema o sull'ambiente. Aura assume un ruolo proattivo anticipando i bisogni degli utenti.

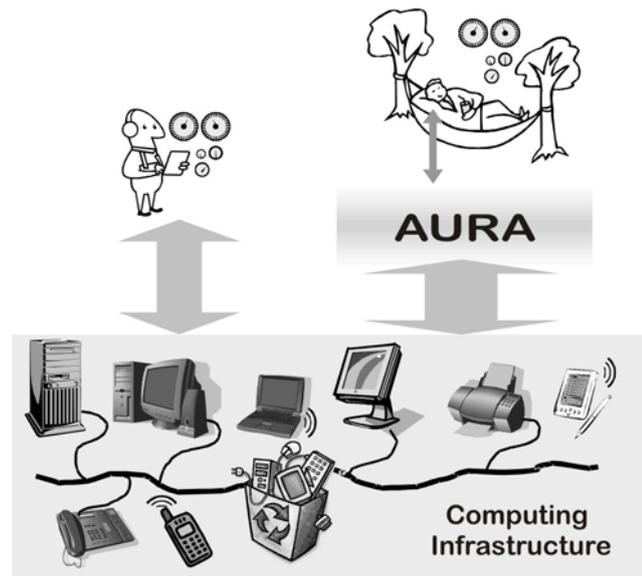


Fig. I.5 AURA: "proxy for people"

Aura utilizza essenzialmente due concetti per perseguire i suoi obiettivi:

*Pro-activity*: rappresenta la capacità, a livello di sistema, di anticipare richieste provenienti da un livello più alto;

*Self-tuning*: rappresenta la capacità di adattarsi osservando le richieste fatte e di modificare le prestazioni e l'utilizzo delle risorse in base ad esse.

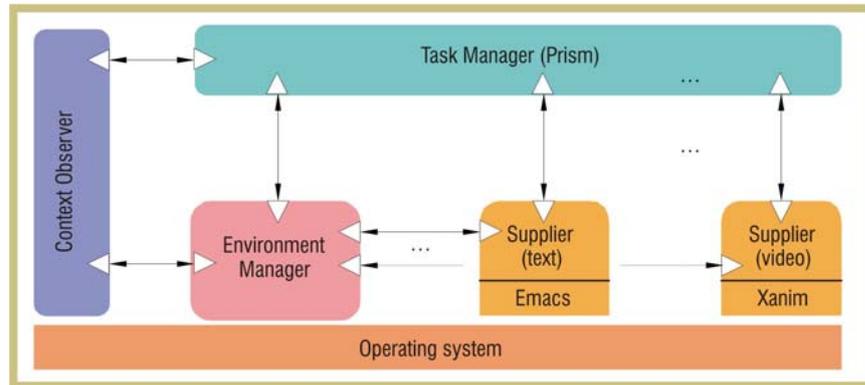
In questa architettura è di fondamentale importanza il concetto di "*user's task*", ovvero l'attività che l'utente vuole svolgere nell'ambiente in cui si trova; ad esempio proiettare una presentazione, stampare, oppure operazioni più complesse [47].

Lo scopo che AURA si prepone è quello di favorire l'utente nell'ottenimento dei suoi obiettivi utilizzando al meglio le risorse presenti nell'ambiente.

Per favorire un'azione del genere c'è bisogno di una struttura architetturale che sia in grado di individuare la natura del task dell'utente, le preferenze personali e le intenzioni.

Tali conoscenze sono la chiave per configurare e monitorare l'ambiente in modo da rendere trasparente all'utente l'eterogeneità degli ambienti informatici e la mutevolezza delle risorse [49].

### I.3.1.1 ARCHITETTURA DI AURA



**Fig. I.6 L'architettura di AURA.**

I componenti principali di AURA sono:

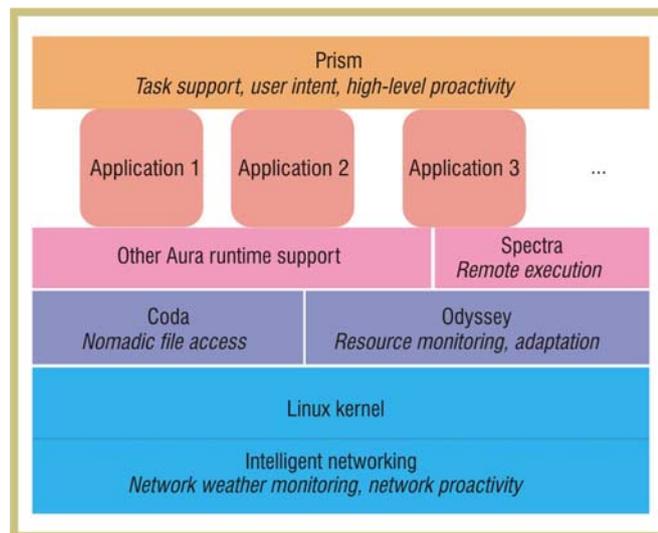
- *Task Manager (Prism)* – Esso ha il compito di rendere l'ambiente libero da distrazioni per l'utente, inoltre si occupa della gestione dei cambiamenti relativi: alla locazione dell'utente; alla conformazione dell'ambiente; alla composizione del task. Quindi Prism fornisce un supporto di alto livello per affrontare gli aspetti di *proactivity* e di *self-tuning*.
- *Environment Manager* – Esso rappresenta la via di accesso all'ambiente. Incapsula i meccanismi per l'accesso a file distribuiti. E' a conoscenza delle risorse disponibili in ogni momento nell'ambiente in quanto, quando un servizio si rende disponibile deve darne comunicazione all'*Environment Manager*. Inoltre, per servizi dotati di risorse condivise limitate, come ad esempio dispositivi input/output, esso tiene traccia anche della capacità disponibile. In aggiunta a meccanismi di discovery di un singolo servizio, un *Environment Manager* più sofisticato è in grado di valutare le possibili configurazioni di un servizio per selezionare quella che combacia meglio con l'esigenza dell'utente.
- *Context Observer* – Offre informazioni riguardo al contesto fisico e notifica gli eventi relativi ad esso a *Environment Manager* e a *Prism*.

Esempi di tali informazioni sono: locazione dell'utente, autenticazione, attività, etc. In ogni ambiente tale componente può avere diversi gradi di complessità, a seconda dei mezzi a disposizione (sensori).

- *Supplier* – Ogni *Supplier* fornisce un servizio astratto. E' realizzato incapsulando le applicazioni esistenti. Ad esempio: Emacs, Microsoft Word e Notepad possono essere incapsulati per diventare il servizio: “*text editing*”.

### I.3.1.2 ARCHITETTURA DI UN AURA CLIENT

Affinché AURA possa offrire i servizi per cui è stato progettato è necessario che anche il client sia dotato di una struttura sofisticata.



**Fig. I.7 Struttura di un AURA client.**

I componenti di un AURA client risultano:

- *Odyssey* – è un'estensione del sistema operativo, che consente al sistema di monitorare le risorse ed alle applicazioni di adattarsi in base alla loro disponibilità;
- *Coda* – è un sistema di gestione di file distribuiti che consente l'accesso continuato ai dati anche in presenza di malfunzionamenti della rete o dei server;
- *Spectra* – è un meccanismo di esecuzione remota ed adattativa che utilizza il contesto per decidere come meglio eseguire le invocazioni remote.

Nello scenario descritto è possibile notare che sia i componenti hardware sia le tecnologie software non sono una novità, infatti Coda e Odyssey sono stati creati prima di Aura, ma sono stati modificati per andare incontro alle esigenze del pervasive computing.

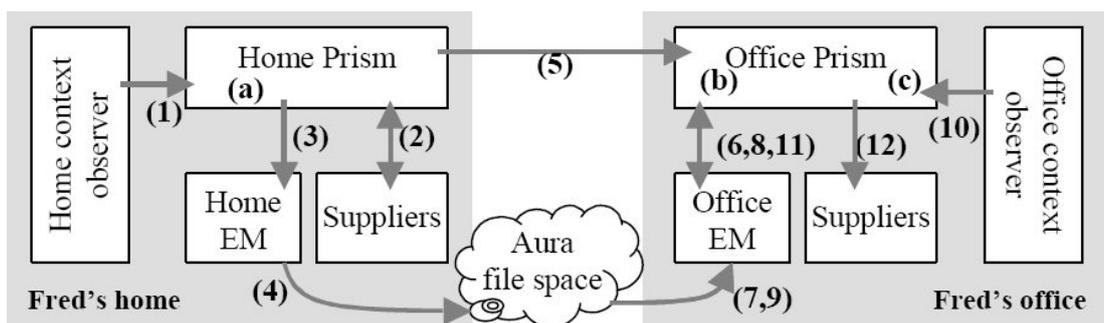
Aura ricicla, in pratica, tecnologie esistenti per creare nuovi sistemi adatti ad essere inseriti in ambienti pervasivi.

### I.3.1.3 AURA A LAVORO

Allo scopo di illustrare come AURA supporta la mobilità dell'utente sarà descritto un semplice scenario.

*Fred sta lavorando a casa all'organizzazione di una conferenza. Egli sta raccogliendo informazioni sulle possibili località e sui costi del servizio di catering attraverso Internet.*

*Fred lascia la casa e si dirige verso il suo ufficio. Quando Fred intende continuare a lavorare all'organizzazione della conferenza, Aura imposta un task in ufficio in modo che Fred possa riprendere il suo lavoro non appena entra in ufficio: un web browser sulle pagine visitate di recente, i video scaricati in pausa allo stesso punto in cui Fred li aveva fermati, e un foglio di lavoro contenente tutte le richieste inoltrate. Da quando sul muro dell'ufficio di Fred è stato montato un grande schermo, egli preferisce visualizzare su di esso i video e le pagine web, lasciando il monitor per la visualizzazione del foglio di lavoro.*



**Fig. I.8 Fred va da casa in ufficio.**

Fred sta lavorando a casa quando l' "Home Context Observer" nota che Fred sta andando via ed informa Prism – interazione (1) in Fig.I.9 – che subisce una transizione di stato (a), e comprende che bisogna sospendere il task in corso a casa.

A questo punto *Prism* richiede lo stato corrente di ogni servizio in uso nel task corrente – interazione (2). Nell’interazione (3), “*Home Prism*” ordina all’ “*Home Environment Manager*” di deallocare le risorse impiegate e memorizzare i file coinvolti in un file server globalmente accessibile – interazione (4).

Dopo la verifica del programma di Fred, *Prism* deduce che probabilmente si dirigerà in ufficio, e (5) trasmette l’informazione all’ “*office Prism*”. Questo provoca la transizione di stato (b) in “*office Prism*”, che richiede all’ “*Office EM*” (6) di recuperare la descrizione aggiornata dei tasks su cui Fred ha lavorato – interazione (7). Dalla descrizione esso capisce quali file servono a Fred per continuare a lavorare e chiede all’ “*Office EM*” di recuperarli – interazione (8). Quest’ultimo esegue l’operazione richiesta – interazione (9).

Non appena l’ “*Office Context Observer*” rileva la presenza di Fred in ufficio informa *Prism* (10) causandone la transizione di stato (c). Quest’ultimo richiede ai suppliers i servizi coinvolti nel task (11) e successivamente ne ripristina lo stato di esecuzione (12).

### **I.3.2 GAIA: UN MIDDLEWARE PER PERVASIVE ENVIRONMENTS**

GAIA ([14], [28],[29]) è un’infrastruttura middleware creata dai ricercatori del Dipartimento di Computer Science nell’Università dell’Illinois, che mira a gestire “Spazi Attivi” (*Active Spaces*).

Nell’ambito di questo progetto di ricerca, vengono definiti i concetti di: “Spazio fisico” (*Physical Space*) una regione geografica con confini fisici limitati e ben definiti, contenente oggetti e dispositivi eterogenei collegati in rete e popolato da utenti che svolgono attività; “Spazio attivo” uno luogo fisico coordinato da un’infrastruttura software sensibile al contesto che consente agli utenti mobili di interagire e configurarsi con l’ambiente fisico e digitale in maniera automatica [29].

L’idea di base è quella di estendere la portata dei sistemi di calcolo tradizionali per includere le apparecchiature e lo spazio fisico che circondano le macchine e permettere, alle entità fisiche e virtuali, di interagire con il sistema: gli Spazi Fisici diventano sistemi interattivi, in altri termini, Spazi Attivi!

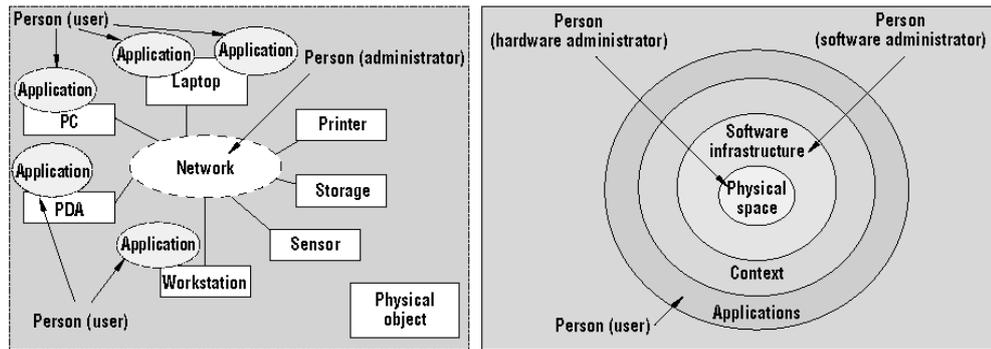


Fig. I.9 Physical Space (sinistra) and Active Space (destra) in GAIA.

Gaia cerca di portare le funzionalità di un *sistema operativo* (come gli eventi, i segnali, il file system, la sicurezza, i processi, i gruppi di processi) in uno spazio attivo, naturalmente estendendo questi concetti ed aggiungendo i nuovi concetti di contesto, ubicazione, dispositivi di calcolo mobili ed attuatori.

L'obiettivo è quindi progettare e implementare un "*sistema operativo middleware*" che gestisca le risorse contenute in uno Spazio Attivo.

Tale sistema deve essere capace di:

- localizzare il dispositivo più adatto;
- rilevare nuovi dispositivi che si aggiungono al sistema;
- adattare il contenuto quando i formati dei dati non sono compatibili con i dispositivi di output.

Il prototipo realizzato opera in spazi fisici usati per insegnare: classi, uffici e stanze per conferenze.

L'ambiente è dotato di attrezzature quali: un sistema audio-sonoro programmabile, cinque pannelli al plasma, HDTV, webcams, bluetooth, ethernet senza fili, apparecchiature di impronta digitale, telefoni intelligenti e la tecnologia di localizzazione Ubisense.

L'architettura di Gaia è formata essenzialmente da quattro componenti:

- *Gaia Kernel*: è un sistema di gestione e sviluppo di oggetti distribuiti (in particolare oggetti CORBA) ed è costituito da un insieme interconnesso di servizi di base di supporto per le applicazioni: *context service*, *event manager*, *presence service*, *security service* e *component repository*;
- *Gaia Application Framework*: modella le applicazioni come collezioni di componenti distribuiti, analizzando le risorse hardware disponibili nello spazio attivo o quelle relative ad un particolare dispositivo; fornisce

funzionalità per alterare la composizione delle applicazioni dinamicamente; è *context-sensitive*; implementa un meccanismo che supporta la creazione di applicazioni indipendenti dallo spazio attivo e fornisce politiche per adattarsi a differenti aspetti delle applicazioni inclusa la mobilità.

- *QoS Service Framework*: si occupa della gestione delle risorse per le applicazioni sensibili alla QoS e adatta dinamicamente tali applicazioni, determinando i nodi appropriati per la loro istanza, in base ad una selezione tra configurazioni in accordo con la disponibilità di risorse, con il servizio di *discovery* e con protocolli di assegnazione di risorse multiple, ossia traducendo i requisiti di alto livello relativi alla QoS in requisiti di sistema;
- *Application Layer*: che contiene le applicazioni e fornisce le funzionalità per registrare, gestire e controllarle attraverso i servizi del Kernel di Gaia.

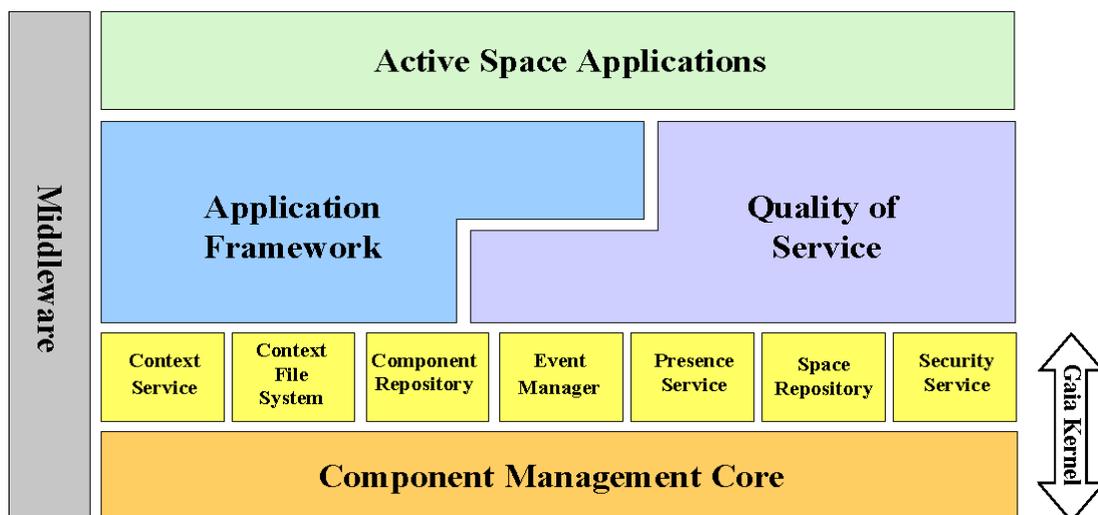


Fig. I.10 L'infrastruttura di GAIA.

In definitiva, GAIA cerca di portare le funzionalità di un sistema operativo in uno spazio attivo (come gli eventi, i segnali, il file system, al sicurezza, i processi, gruppi di processi), naturalmente estendendo questi concetti ed aggiungendo il nuovo concetto di contesto.

### I.3.3 COOLTOWN

Il progetto realizzato presso i laboratori della Hewlett-Packard, Cooltown [10], offre un modello di sistema di *pervasive computing* che combina tecnologie web, reti wireless e dispositivi portatili per creare un ponte virtuale tra utenti mobili e entità fisiche e servizi elettronici [20].

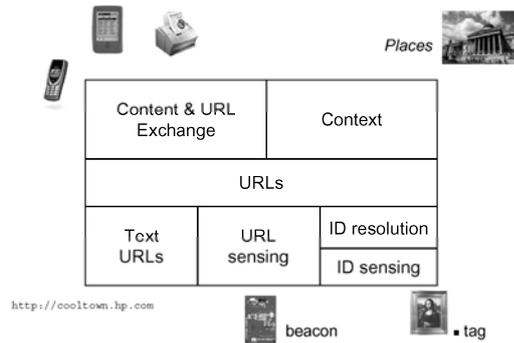
La ragione per cui HP ha scelto il web come tecnologia alla base del suo prototipo di ambiente pervasivo è dovuta alla sua facile adozione, essendo l'ambiente di *distributed computing* più diffuso, rispetto ad una qualsiasi altra tecnologia da sviluppare ex novo. Inoltre il web offre la possibilità di accesso dovunque, possiede requisiti software leggeri, permette di operare sia localmente che su scala globale .

E' un progetto focalizzato soprattutto sul livello dell'infrastruttura necessaria per supportare utenti mobili con device portatili e non tanto sul livello dei device e della comunicazione wireless. Si lavora al livello "web server" con lo scopo di *estendere i modelli del web a nuove aree*.

Nella visione di Cooltown, ogni entità, oggetto, posto, persona, è dotata di una rappresentazione web (web presence). Un oggetto fornisce la propria URL, questa punta alla homepage, localizzata su un web server dedicato, attraverso la quale può essere controllato dall'utente: si può fare "click" sulle cose.

Un posto è una collezione di oggetti che hanno presenza web. Una persona è rappresentata da una web page con links che rappresentano servizi cui possono accedere altri individui per comunicare con lei.

In Cooltown, dove il protocollo di comunicazione è HTTP, è possibile comunicare con i dispositivi anche se non si è direttamente agganciati ad una rete globale adoperando ad esempio tecnologie come Bluetooth o IrDA, che risultano sufficienti per individuare, ad esempio, una stampante ed avviare il processo di stampa di un documento.



**Fig. I.11 La visione di Cooltown.**

L'infrastruttura per supportare tale web presence è articolata in tre livelli:

- *Acquisizione URL (Bottom layer)*: tale livello permette all'utente di acquisire URLs dall'ambiente e da entità fisiche nei dintorni. L'acquisizione dell'URL può avvenire in diversi modi:
  - *Text URLs* : l'utente digita l'URL di un oggetto o posto;
  - *Auto Service Discovery*: si usano protocolli di service discovery;
  - *URL Sensing* : i PDA hanno almeno un dispositivo short-range, con il quale, puntandolo o posizionandolo opportunamente, si possono leggere URL ed altri identificatori associati a posti ed oggetti grazie a: segnali *IR & RF*, *Codici a barre*, *Etichette elettroniche*. Il *sensing* può avvenire in maniera diretta: il segnalatore o etichetta presenta direttamente l'URL di una risorsa web; oppure in maniera indiretta: viene presentato un identificatore come ad es. ISBN o un UPC barcode. Poi per ottenere un URL si usa un *resolver*, un servizio che ritorna un URL a partire da un dato identificatore.
- *Content exchange (Middle layer)*. Lo scambio di Contenuti è opposto al browsing: l'utente nomade può inserire del contenuto nell'infrastruttura pervasiva piuttosto che estrarne soltanto. L'inserimento (pushing o 'posting') di contenuto lavora tra un source (sorgente) di contenuto e un sink (pozzo) di contenuto. Ad esempio: *un utente entra in una sala riunione con una fotocamera (source), fotografa il contenuto di una lavagna, stampa l'immagine su una stampante (sink) della stanza*. L'operazione di *post* può essere fatta sia in modo diretto che in modo indiretto. Nel primo caso si tratta di una semplice operazione di inserimento: la sorgente apre una connessione verso il sink e scrive

l'effettivo contenuto. L'interazione richiede dei metadati relativi ai dati del contenuto. Il formato dei metadati deve essere concordato e stabile nel tempo: in tale approccio si usa un formato XML con codifica ASCII trasportato in una MIME entity. Nel secondo caso il contenuto inserito contiene links ipertestuali ad altri contenuti, quindi si ha lo scambio di URL invece che di contenuti diretti .

- *Context and Physical discovery(Top layer)*. A questo livello viene preso in considerazione il contesto dell'utente: i servizi differiscono a seconda del posto (place) da cui sono acceduti. La rappresentazione di un posto fornisce un contesto: un insieme di risorse correlate. L'implementazione del contesto di un posto è un web server con l'appropriato contenuto che descrive il posto. Nella web page del posto ci sono i link agli oggetti del posto dotati di presenza web. Un *place manager* fornisce accesso e configurazione per i dati associati ad uno o più posti:

- mantiene directories di risorse di un posto e offre un'interfaccia per aggiungere, richiedere, rimuovere risorse;
- agisce da resolver cercando risorse a partire dai loro identificatori.

Da quanto detto è evidente che una caratteristica fondamentale dell'architettura in esame è la "*location awareness*", cioè la consapevolezza della locazione sia delle risorse che dell'utente.

Con XML si possono descrivere degli attributi semantici associati al link della locazione (location representation). In questo modo possono essere offerti servizi personalizzati ai clienti sulla base della loro locazione.

Il servizio web associato all'oggetto conosce la locazione fisica dell'oggetto e può aiutare l'utente a localizzare il servizio richiesto. Però, eseguire un'operazione di tracking dell'utente non è sempre attuabile in quanto:

- a) Il mapping inverso da locazione semantica a locazione fisica non porta ad un'unica locazione, quindi il servizio non può usare le informazioni sulla locazione semantica per localizzare i client;
- b) L'utente non deve necessariamente presentarsi al posto dove c'è l'oggetto fisico per poter accedere al servizio.

Le caratteristiche dei dispositivi dell'ambiente HP sono descritte in una web-form indipendente dal tipo di dispositivo stesso; ciò consente una semplice interazione tra

dispositivi, mediante semplici URL, senza la necessità di dover installare driver per ogni tipo di dispositivo incontrato.

Tuttavia, l'interazione macchina-macchina tra dispositivi sconosciuti è un aspetto ancora irrisolto poiché le web-interface sono essenzialmente rivolte all'uomo e non sono adatte ad una elaborazione automatica.

La sicurezza in Cooltown è gestita da un servizio di autorizzazione centrale che applica una certa politica di accesso a gruppi di servizi. A tale proposito è possibile individuare tre fasi:

- *Setup Phase*: il gestore genera delle chiavi e le comunica ai servizi di un dato gruppo. Tali chiavi hanno lo scopo di descrivere le credenziali che devono essere fornite dall'utente per accedere a tali servizi.
- *User Registration Phase*: quando l'utente entra nell'ambiente il gestore determina il gruppo di servizi ai quali egli può accedere. Genera un documento criptato di credenziali e solo i servizi appartenenti al gruppo accessibile dall'utente saranno in grado di decifrarlo. Le credenziali sono memorizzate sul dispositivo utente e sono richieste ogni qual volta si renda necessaria l'identificazione dell'utente, cioè quando egli richiede l'uso di un particolare servizio.
- *User Access Phase*: L'utente presenta le sue credenziali al servizio a cui vuole accedere. Il servizio decifra le credenziali per determinare la chiave di autenticazione. Inoltre il servizio deve verificare, sempre a partire dalle credenziali, se esso è il servizio effettivamente richiesto dall'utente.

### **I.3.4 CENTAURUS**

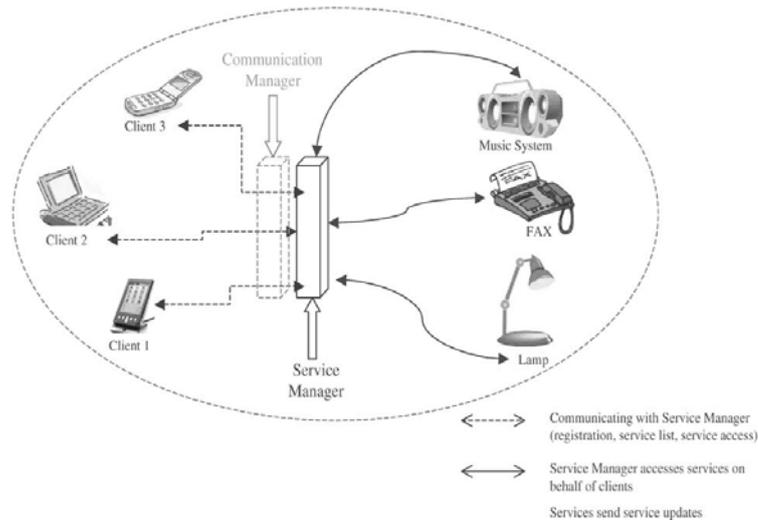
Centaurus [19] definisce un'infrastruttura ed un protocollo di comunicazione per servizi software ed hardware eterogenei che devono essere resi disponibili agli utenti ovunque ne abbiano bisogno. Tali servizi intelligenti devono essere capaci di comprendere le necessità dell'utente per fornirgli migliore supporto ed agevolarne le attività. Tale architettura si comporta da proxy attivo eseguendo servizi per conto dei client che li richiedono.

L'architettura di Centaurus consiste di quattro componenti:

- *Communication Manager*: è responsabile della comunicazione tra il client ed altri componenti Centaurus. Quando riceve informazioni da un client le

invia al Service Manager; quando riceve dati dal Service Manager li valida e analizza l'header per decidere a quale client inoltrarli. Può usare diversi moduli di comunicazione con il client a seconda del mezzo trasmissivo: Bluetooth, IR, CPDP,... Per la comunicazione si usa un linguaggio proprietario basato su XML, il *Centaurus Communication Markup Language* (CCML). Tale linguaggio è integrabile con linguaggi semantici come, ad esempio DAML+OIL

- *Service Manager*: controlla l'accesso ai servizi ed agisce da "gateway" tra i servizi ed i client. Quando un servizio fa lo start up, si registra presso il Service Manager, mandandogli il suo CCML file. Quando un nuovo cliente entra, il SM gli manda un oggetto "ServiceList", che verrà dinamicamente aggiornato. Attraverso questa lista il client potrà selezionare un servizio e il SM gli invierà la descrizione CCML per quel servizio. In alternativa il client potrà invocare il servizio mandando un nuovo file CCML al SM e questo, se il servizio è disponibile, gli inoltra la richiesta.
- *Services*: sono oggetti che offrono funzionalità ai *client* Centaurus: controllare un interruttore di una lampada, stampare... Ogni servizio si registra presso il SM inviandogli un CCML file, con nome, identificatore, locazione, una breve descrizione, il suo "leasing period". Ogni volta che il suo stato cambia deve informare il Service Manager, o deve rinnovare il suo leasing. Accetta richieste solo dal Service Manager con cui si è registrato. I servizi contengono informazioni necessarie a localizzare il più vicino *Service Manager* ed a registrarsi ad esso. Una volta registrato, un servizio può essere acceduto da qualsiasi *client* attraverso il *Communication Manager*.
- *Clients*: implementano un'interfaccia utente per interagire con i servizi. Un *client* può accedere ai servizi forniti dal più vicino sistema Centaurus che si comporta come un proxy soddisfacendo le sue richieste.



**Fig. I.12 Architettura di Centaurus.**

Tutti i componenti del modello, per la comunicazione usano un linguaggio proprietario, basato su XML, il Centaurus Communication Markup Language (CCML). In questo modo offrono al sistema un'interfaccia uniforme ed adattabile. Si tratta di un protocollo di trasporto efficiente, basato su messaggi.

E' costituito da due livelli: i moduli del livello I sono dipendenti dal mezzo fisico, mentre il livello II è indipendente dal mezzo. E' progettato per essere eseguito su un'ampia gamma di dispositivi di limitate capacità elaborative: non sfrutta caratteristiche di sistemi operativi avanzati come segnali e multithreading, al contrario di TCP che necessita del supporto per il signaling da parte del S.O.

In ogni caso, in presenza di reti wireless con banda limitata ed elevata latenza funziona meglio di TCP.

### **I.3.5 PROGETTO COBRA**

All'Università del Maryland in Baltimora è stato sviluppato un modello architetturale per *pervasive environments* con particolare attenzione alle problematiche di context-aware computing.

Il cuore dell'architettura è caratterizzato da un'entità server specializzata ed intelligente, il Context Broker (da cui il nome CoBrA: Context Broker Architecture), che riceve informazioni legate al contesto dai dispositivi e dagli agenti presenti, le relaziona, definendo un modello centralizzato e condiviso dell'ambiente e di tutto ciò

che si trova ad operare al suo interno, e si preoccupa di mantenerlo, nel tempo, coerente e privo di inconsistenze in seguito a nuove acquisizioni di informazioni.

Un punto chiave nella realizzazione di questa architettura è lo sviluppo e l'utilizzo di una serie di ontologie comuni, attraverso le quali agevolare la comunicazione fra i diversi agenti e rappresentare lo stato del sistema.

Tale architettura è caratterizzata dal fatto di definire ontologie OWL per abilitare agenti a processare e ragionare sul contesto, includere un motore inferenziale per ragionare sull'informazione di contesto e individuare e risolvere inconsistenze, inoltre, applicare un approccio policy-based per controllare come condividere tra gli utenti le informazioni di contesto.

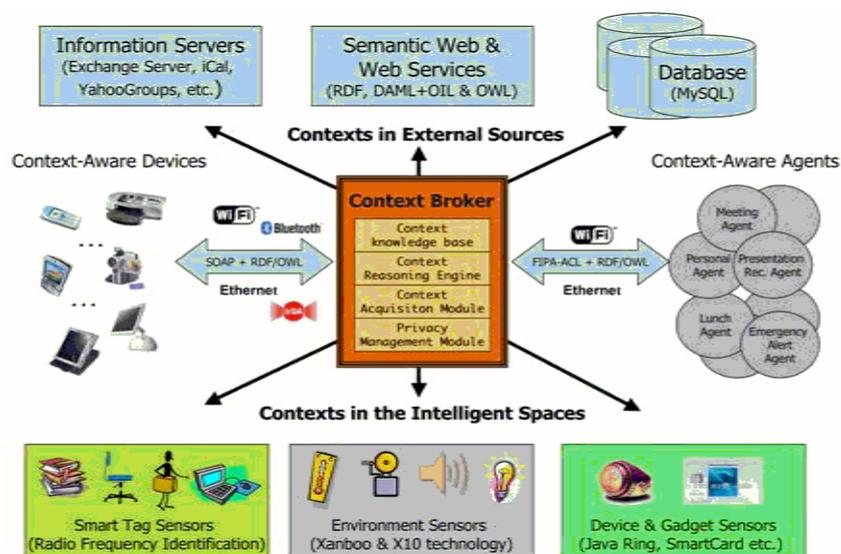


Fig. I.13 Architettura di COBRA.

L'architettura del context broker è costituita da quattro componenti funzionali ([8],[9]):

- *Context Knowledge Base:* immagazzina in maniera persistente la conoscenza del contesto. Fornisce un set di API per consentire agli altri componenti di accedere alle informazioni memorizzate. Contiene anche l'ontologia dello specifico ambiente e le regole euristiche associate ad esso;
- *Context Reasoning Engine:* è un motore che applica algoritmi di inferenza sulle informazioni di contesto immagazzinate. Si possono avere inferenze che usano le ontologie per dedurre conoscenza di contesto e inferenze che usano la conoscenza per rilevare e risolvere inconsistenze;

- *Context Acquisition Module*: è una libreria di procedure che rappresenta una sorta di *middleware* per l'acquisizione di informazioni di contesto;
- *Policy Management Module*: è un insieme di regole di inferenza definite sia per valutare i privilegi di accesso, associati alle differenti entità, sia per condividere particolari frammenti delle informazioni di contesto e ricevere notifiche dei cambi di contesto.

Il modello con broker centralizzato affronta due problemi chiave del pervasive computing: supportare dispositivi con limitate capacità e gestire la *privacy dell'utente*. Grazie ad un context broker che opera su un computer fisso, la complessità di acquisire e ragionare su informazioni contestuali è spostata dai dispositivi mobili limitati verso il broker dotato di elevate risorse, inoltre le complicazioni nel garantire security, trust, and privacy policies sono semplificate dalla presenza di un manager centralizzato.

In base al progetto di tale architettura è stato realizzato il prototipo di una "*intelligent meeting room*" che usa CoBrA.

Tale modello di prova offre servizi e informazioni ai partecipanti *sulla base delle loro necessità correnti* e permette agli utenti di controllare l'uso e la condivisione della loro locazione e del contesto sociale.

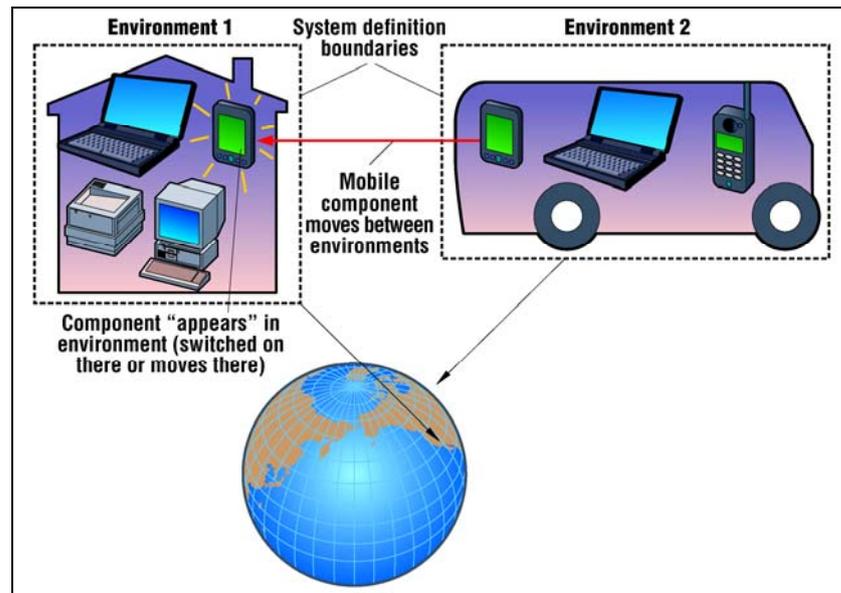
In futuro, gli sviluppatori di questa architettura prevedono di migliorare il meccanismo di inferenza logica, a tal proposito si sta investigando sull'uso del framework Theorist, un metainterprete Prolog: secondo tale approccio le premesse consistono sia di *fatti* (assiomi dati per certi) che di *assunzioni* (istanze di *possibili ipotesi* che possono essere assunte se sono coerenti con i fatti), in altre parole si affianca il default reasoning all'abduction reasoning.

Tutte le informazioni di contesto acquisite dal Context Broker sono viste come sue osservazioni dell'ambiente. Quando riceve un'osservazione, il Context Broker, prima usa l'abduzione per determinare le possibili cause, poi usa il ragionamento di default per predire cos'altro conseguirà dalla causa.

#### **I.4 CARATTERISTICHE DI UN SISTEMA PERVASIVO**

Nei paradigmi di *ubiquitous computing*, servizi ed informazioni sono virtualmente accessibili dovunque, in ogni istante attraverso qualsiasi dispositivo, ma

considerazioni di carattere amministrativo, territoriale e culturale ci inducono ad analizzare l'*ubiquitous computing* in ambienti discreti, dai confini ben definiti, come per esempio case, uffici, sale convegno, aeroporti, stazioni, musei, etc. In altre parole, è bene considerare il mondo suddiviso in tanti domini pervasivi, piuttosto che vederlo come un unico enorme sistema [21].



**Fig. I.14 Il mondo suddiviso in ambienti pervasivi con confini ben definiti.**

Ogni singolo ambiente è contraddistinto da componenti o unità software che implementano astrazioni di servizi, clienti, risorse o applicazioni ed, in generale, ogni ambiente è costituito da un'infrastruttura fissa e da una serie di elementi mobili che, in maniera del tutto imprevedibile, entrano a far parte del sistema e lo abbandonano continuamente, talvolta migrando fra i diversi domini.

A partire da queste considerazioni e dai progetti presentati nei paragrafi precedenti è possibile individuare una serie di aspetti di carattere generale e indicazioni di fondamentale importanza per l'analisi, lo sviluppo e l'implementazione di un sistema di pervasive computing [35].

#### **I.4.1 CONTEXT INFORMATION E RUN-TIME ADAPTATION**

Una prerogativa di un sistema pervasivo è la capacità di ottenere informazioni sugli utenti e sullo stato dell'ambiente, come ad esempio, la posizione e l'identità dei singoli utenti e la disponibilità delle risorse.

Questo viene realizzato collezionando dati grezzi provenienti da una moltitudine di sorgenti. Tali dati saranno poi processati e trasformati in informazioni di contesto da condividere tra le diverse applicazioni in esecuzione sui vari dispositivi, cercando di mantenere la scalabilità, di garantire la sicurezza delle informazioni, inibendo accessi non autorizzati e rispettando la privacy individuale.

In base a queste informazioni, le applicazioni possono adattare il loro comportamento in maniera diversa a seconda dei casi. E' possibile, a tal proposito, definire due tipi di adattamento: funzionale o strutturale.

Un esempio di adattamento funzionale può essere rappresentato da un'applicazione che fornisce news in un ambiente: il tipo di notizie che vengono fornite potrebbe essere determinato in base a chi si trova nell'ambiente o all'ora della giornata.

Un esempio di adattamento strutturale, invece, può essere rappresentato da un'applicazione musicale: a seconda se l'utente è solo oppure no all'interno dell'ambiente, l'applicazione potrebbe utilizzare il sistema audio del laptop dell'utente, oppure quello dell'ambiente stesso.

Per realizzare quanto appena detto, occorre necessariamente avere una percezione dell'ambiente ragionevolmente accurata, e disporre di meccanismi per il rilevamento e la correzione di informazioni di contesto inattendibili o contrastanti.

#### **I.4.2 TASK RECOGNITION E PRO-ACTIVITY**

Un sistema di *pervasive computing* dovrebbe essere capace, a partire dalle informazioni di contesto collezionate, di elaborare lo stato corrente dell'ambiente e le intenzioni dell'utente e modificare dinamicamente il proprio comportamento per assistere l'utente stesso nelle sue attività.

Diversamente dai sistemi di *computing* convenzionali in cui il comportamento del computer è principalmente composto di risposte all'interazione con l'utente, il *pervasive computing* mira alla realizzazione di un modello in cui i dispositivi sono la parte attiva nell'interazione con l'utente. Quindi se la tecnologia corrente si basa su persone che indicano ai computer ciò che devono fare, la nuova generazione di tecnologie dovrebbe essere basata su computer capaci di comprendere quello che le persone stanno facendo e quello che esse desiderano.

I modelli che tengono conto dell'esperienza, del passato, rappresentano un importante strumento per la caratterizzazione di un sistema pervasivo, perché favoriscono la *pro-activity* del sistema, ossia consentono di determinare, in accordo con i precedenti comportamenti dell'utente, le azioni ottimali che l'utente stesso deve eseguire in determinate situazioni.

#### **I.4.3 RESOURCE ABSTRACTION E DISCOVERY**

Le risorse di un sistema pervasivo dovrebbero essere rappresentate in maniera astratta (magari attraverso le caratteristiche anziché attraverso i nomi) cosicché possano facilmente essere selezionate in base a requisiti di tipo generale oltre che specifico. Potrebbe essere necessario definire alcune convenzioni di nomi per evitare che differenti risorse descrivano se stesse adoperando gli stessi termini offrendo, però, servizi diversi.

Dovrebbero essere previsti anche dei meccanismi per scoprire, interrogare ed interagire con le risorse nell'ambiente, per consentire l'introduzione di nuovi componenti senza onerose operazioni di configurazione e di riconfigurazione dei componenti esistenti.

#### **I.4.4 ETEROGENEITY E SERVICE UI ADAPTATION**

Un ambiente pervasivo è caratterizzato da una moltitudine di dispositivi eterogenei, in numero variabile, come laptop, PDA, telefoni mobili.

Alla riduzione delle dimensioni dei dispositivi dell'ambiente corrisponde una maggiore crescita del numero di dispositivi connessi ed una intensificazione delle interazioni uomo-macchina.

Lo sviluppo tradizionale fornisce servizi tipicamente distribuiti e installati separatamente per ogni classe di dispositivo e famiglia di processore. Gli scenari di *computing* pervasivo, invece, portano alla conclusione che distribuire ed installare servizi per ogni classe e famiglia diventa ingestibile, specialmente in un'area geografica molto estesa.

E' importante, quindi, che i servizi forniscano agli utenti interfacce che possano adattarsi alle caratteristiche del dispositivo *client*, senza stravolgere le funzionalità del servizio stesso [13]. Ad esempio, un servizio adibito al controllo

dell'illuminazione in una stanza è libero di fornire un'interfaccia utente di tipo grafico per un *client* PDA, ma deve necessariamente garantire anche una rappresentazione testuale della stessa UI per un utente che adopera un telefono cellulare, senza, peraltro, dover cambiare l'implementazione del servizio stesso.

#### **I.4.5 SECURITY E PRIVACY**

Un sistema pervasivo generalmente gestisce grosse quantità di informazioni, molte delle quali acquisite tramite sensori e riguardanti gli utenti.

Dal punto di vista dell'utente, è desiderabile che vengano rispettati i principi di privacy e che sia garantita la sicurezza di queste informazioni.

Ad esempio, risulta fastidioso sapere che la stazione di polizia più vicina possa conoscere in quale stanza ci si trovi nella propria casa, attraverso i rilevatori di moto del sistema d'allarme, o quanto alcool si sta consumando, deducendo questa informazione dal sistema che gestisce l'inventario degli alimenti.

D'altra parte, in molte situazioni, una certa perdita di privacy può essere tollerata, come ad esempio in situazioni di pericolo.

Per aggiungere sicurezza alle informazioni, i servizi nell'ambiente non dovrebbero consentire accessi non autorizzati: ad esempio, a casa propria, non dovrebbe essere possibile per un ospite, aumentare il riscaldamento o controllare il funzionamento del forno [4].

#### **I.4.6 FAULT-TOLERANCE E SCALABILITY**

Gli ambienti pervasivi costituiscono sistemi "perennemente" attivi. Pertanto, un componente che subisce un guasto non deve compromettere il funzionamento generale dell'intero sistema, né richiedere una complessa tecnica di gestione.

I componenti che cadono dovrebbero automaticamente ripartire, laddove possibile, magari adoperando, ad esempio, memorie di stato persistenti che consentano di effettuare *resume* rapidi ed efficaci.

Abbiamo visto che gli ambienti pervasivi sono caratterizzati anche da una forte dinamicità: dispositivi possono aggiungersi all'ambiente ed abbandonarlo in qualsiasi momento; alcuni servizi possono cadere e presentarsene altrettanti nuovi; gli stessi utenti possono entrare ed uscire dall'ambiente secondo la propria volontà.

Il sistema deve garantire scalabilità, ossia essere in grado di gestire e assicurare il suo funzionamento anche in seguito all'aggiunta di componenti; allo stesso tempo, i nuovi dispositivi e servizi introdotti nell'ambiente non dovrebbero interferire con quelli esistenti.

## II. TECNOLOGIE PER IL PERVASIVE COMPUTING

### II.1 LA COMUNICAZIONE NEI SISTEMI DISTRIBUITI

Un sistema distribuito consiste in un insieme di computer che comunicano su di una rete per coordinare le azioni e i processi di un'applicazione [25]. Le tecnologie per la realizzazione di sistemi distribuiti hanno suscitato molto interesse negli ultimi anni, anche grazie alla proliferazione dei sistemi e servizi basati sul Web.

Tecnologie consolidate come la comunicazione tra processi e l'invocazione remota, i naming service, la sicurezza e la crittografia, i *file system* distribuiti, la replicazione dei dati e i meccanismi di transazione distribuita, forniscono una solida infrastruttura di *run-time* che supporta le applicazioni di rete odierne.

Il modello dominante è ancora la tradizionale architettura *client-server*, ma lo sviluppo di applicazioni in ambito distribuito si sta basando sempre di più sia sull'impiego di supporti *middleware*, che forniscono astrazioni di alto livello, come oggetti distribuiti condivisi, sia su altri generi di servizi, come per la comunicazione sicura o per l'autenticazione.

Inoltre sia Internet, con i suoi protocolli base, sia il World Wide Web, ad un livello più alto, stanno diventando una piattaforma standard per le applicazioni distribuite. Difatti Internet e le sue risorse possono essere viste come un ambiente globale in cui ha luogo l'elaborazione informatica. Di conseguenza l'attenzione si

focalizza su nuovi standard e protocolli di alto livello, come XML, mentre passano in secondo piano aspetti di basso livello, come ad esempio le peculiarità di un sistema operativo.

## II.2 PARADIGMI DI COMUNICAZIONE

Esistono diversi modi attraverso i quali componenti di applicativi software risidenti su macchine differenti possono comunicare fra loro adoperando una rete. Una tecnica di basso livello è quella di utilizzare direttamente le interfacce offerte dal livello trasporto, come il meccanismo delle *socket*, assieme ad un protocollo di comunicazione pensato ad hoc per l'utilizzo specifico.

Comunque programmare a questo livello di astrazione è consigliabile solo in particolari circostanze poiché demanda completamente al programmatore la gestione di complessi problemi come la sicurezza, l'eterogeneità e la concorrenza.

Nella maggior parte dei casi, invece, è preferibile scegliere tra una serie di protocolli e ambienti di più alto livello quello che meglio si adatta alle proprie esigenze. Alcuni di questi protocolli sono *self-contained* e possono essere usati in qualsiasi programma applicativo con un *overhead* addizionale basso o addirittura nullo. Altri protocolli ed ambienti, invece, sono vincolati a specifici linguaggi di programmazione o a particolari piattaforme di esecuzione [25].

### II.2.1 REMOTE PROCEDURE CALL

Un classico schema di comunicazione, che ben si addice al modello *client-server*, è la chiamata a procedure remote (RPC). In questo modello, un componente agisce da client quando richiede un servizio ad un altro componente, da server quando, invece, è lui a rispondere alle richieste. RPC effettua una chiamata ad una procedura esterna che risiede in un differente nodo della rete quasi con la stessa semplicità con cui invoca una procedura locale. Argomenti e valori di ritorno sono automaticamente impacchettati in un formato definito dall'architettura e spediti tra procedure locali e remote.

Per ogni procedura remota, il *framework* RPC sottostante ha bisogno di una procedura stub dal lato client (che agisce da *proxy*) e di un oggetto simile lato server.

Il ruolo dello stub è prendere i parametri passati attraverso una regolare procedura locale ed inviarli al sistema RPC (che deve risiedere su entrambi i nodi). Dietro le quinte, il sistema RPC coopera con gli stub di ambo i lati per trasferire argomenti e valori di ritorno sulla rete.

Per facilitare la creazione di stub, sono stati realizzati speciali tool. Il programmatore fornisce i dettagli di una chiamata RPC in forma di specifiche, espresse attraverso l'*Interface Definition Language* (IDL), poi viene utilizzato un compilatore IDL che genera, a partire da tali specifiche, gli stub in maniera automatica.

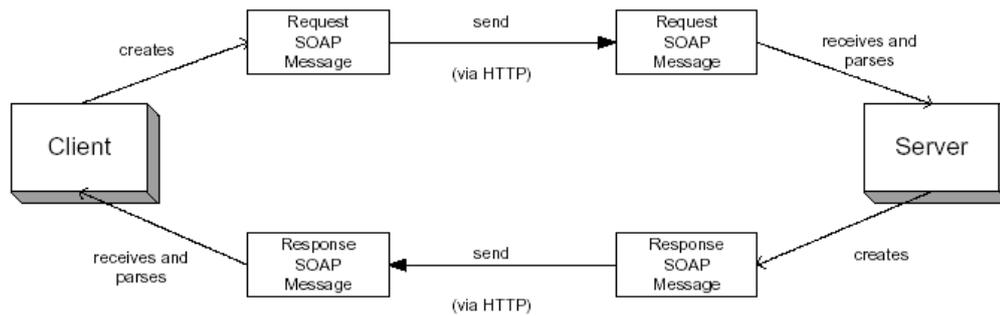
I framework RPC, anche se sono tipicamente invisibili ai programmatori, sono diventati una tecnica consolidata poiché rappresentano i meccanismi di trasporto su cui si basano le più generali piattaforme *middleware* di cui si parlerà in seguito.

## II.2.2 XML-BASED RPC

Sebbene i sistemi RPC trattano esplicitamente aspetti di interoperabilità in sistemi aperti, i programmi *client* e *server* che fanno uso di questo principio sono vincolati ad un singolo *framework* RPC. La ragione principale è che ciascun *framework* definisce la propria tecnica di codifica per le strutture dati. Nonostante queste differenze, le semantiche base della maggior parte dei sistemi RPC sono simili poiché si fondano su chiamate a procedure sincrone in un formato espresso in sintassi C-like.

L'idea nuova è stata quella di utilizzare XML per definire la sintassi delle richieste e delle risposte RPC, consentendo a differenti sistemi RPC di poter comunicare tra loro. In pratica XML è usato per definire un sistema di tipi che può essere adoperato per scambiare dati tra *client* e *server*. Questo sistema specifica tipi primitivi, come interi, *floating point*, stringhe di testo e fornisce i meccanismi per aggregare istanze di tipi primitivi in tipi composti per ottenere nuove categorie di dati.

Uno dei primi *framework* RPC basati su XML è stato SOAP (Simple Object Access Protocol) [40] definito inizialmente da un consorzio di compagnie tra le quali figuravano Microsoft, IBM, SAP, ma che oggi è invece divenuto un progetto *open source* in fase di standardizzazione presso il *World Wide Web Consortium* (W3C).



**Fig. II.1** Interazione client-server usando SOAP.

SOAP [40] definisce solo la struttura del messaggio ed alcune regole per elaborarlo, rimanendo quindi ad un livello alto e completamente indipendente dal protocollo di trasporto sottostante. Aspetti chiave di SOAP sono, quindi, la sua estensibilità dovuta all'uso di schemi XML e l'utilizzo del protocollo HTTP come meccanismo di trasporto tra *client* e *server*, e quindi del Web come infrastruttura di comunicazione.

### II.2.3 REMOTE METHOD INVOCATION

Mentre RPC è ragionevolmente ben adattabile al paradigma di programmazione procedurale, non è direttamente applicabile allo stile di programmazione *object oriented* che ha riscosso molta popolarità negli ultimi anni.

In questo contesto fa la sua comparsa RMI, una nuova tecnica di comunicazione remota per sistemi *Java-based*. Simile a RPC, integra il modello ad oggetti distribuiti nel linguaggio Java in modo naturale, lavorando direttamente sugli oggetti esistenti ed evitando di descriverne i metodi in un file di definizione diverso.

In un classico sistema RPC, il codice dello stub lato client deve essere generato e linkato nel client prima che una procedura remota possa essere chiamata; RMI, invece, è più dinamico perché, sfruttando la mobilità del codice Java, gli stub necessari per l'invocazione possono essere scaricati a tempo di compilazione da una locazione remota, per esempio direttamente dal server.

Internamente, RMI fa uso della serializzazione degli oggetti per trasmettere tipi di oggetti arbitrari sulla rete e poiché il codice scaricato può essere dannoso per il sistema, usa un *security manager* per operare dei controlli di attendibilità.

## II.3 MIDDLEWARE ED INFRASTRUTTURE DI COMUNICAZIONE

I middleware e le infrastrutture software per sistemi distribuiti forniscono strumenti per la comunicazione ai componenti applicativi e gestiscono aspetti come l'eterogeneità fra le varie piattaforme, dovuta a differente hardware, software, sistema operativo e linguaggio di programmazione.

Inoltre forniscono un set di servizi standard che in genere sono indispensabili per lo sviluppo di applicazioni distribuite, come servizi di directory, sicurezza e crittografia.

### II.3.1 CORBA

Una delle più usate infrastrutture per sistemi distribuiti basata sul modello *object-oriented* è CORBA (*Common Object Request Broker Architecture*), che è supportata da un esteso consorzio di industrie.

Il primo standard CORBA è stato introdotto nel 1991 ed è stato oggetto di continue e significative revisioni.

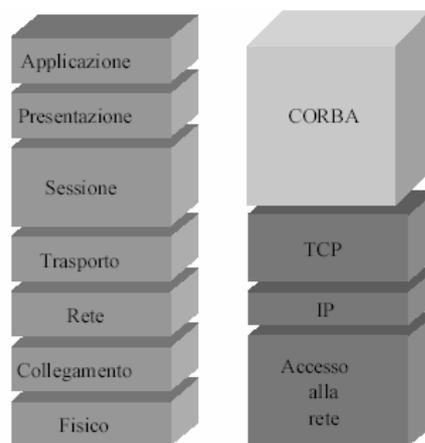
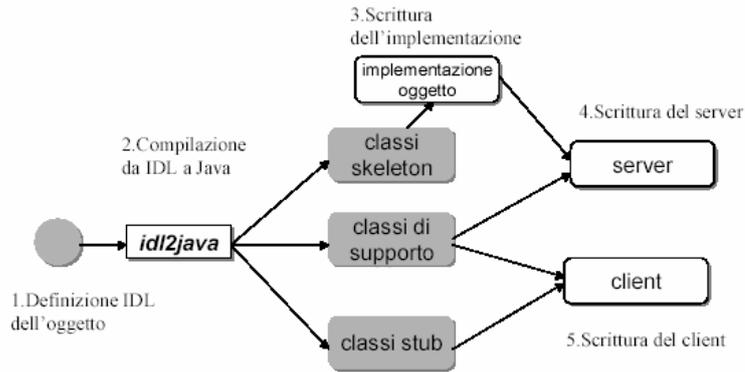


Fig. II.2 CORBA rispetto alla pila OSI.

Una parte delle specifiche descrive l'*Interface Definition Language* (IDL) che deve essere supportato da tutte le implementazioni di CORBA. L'IDL di CORBA si rifà al C++ ed è usata dalle applicazioni per definire i metodi di un oggetto che è possibile utilizzare esternamente.

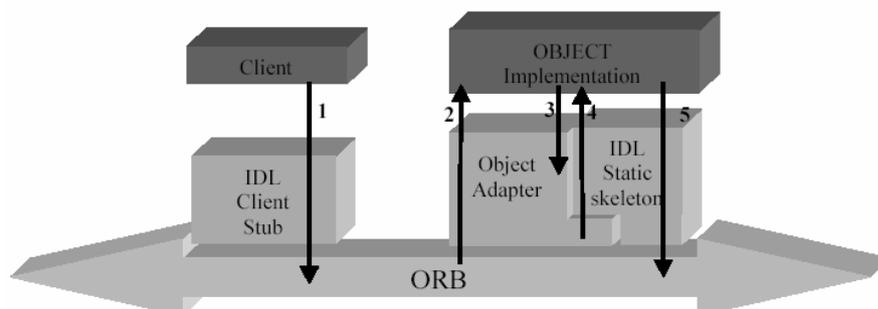


**Fig. II.3 Ciclo di sviluppo di un'applicazione CORBA attraverso Java.**

E' simile all'IDL di RMI. Tramite un compilatore IDL si ottengono file contenenti del codice di supporto per l'aggancio all'ORB, lo *stub* (per il client), lo *skeleton* (per il server). Lo *stub* e *skeleton* realizzano le funzionalità di *marshalling* e di *unmarshalling* dei dati. Lo *skeleton*, inoltre, dal lato server, coopera con un *Object Adapter* per le operazioni che riguardano l'attivazione dell'oggetto.

Il componente centrale di un sistema CORBA è l'*Object Request Broker* (ORB), una sorta di bus software. Esso fornisce un meccanismo per comunicare in maniera trasparente richieste di client alle implementazioni degli oggetti server.

Questo semplifica la programmazione distribuita, disaccoppiando il client dai dettagli di invocazione dei metodi: quando, infatti, un client invoca un'operazione, l'ORB si preoccupa di trovare l'implementazione dell'oggetto, di attivarlo se necessario adoperando l'OA, di inoltrargli la richiesta, e ritornare una eventuale risposta al chiamante.



**Fig. II.4 Interazione client-server in CORBA.**

CORBA prevede sia collegamenti di tipo statico che di tipo dinamico tra client e server. Nel caso statico, codice proveniente dal server (stub del client) viene compilato con il client stesso. Di conseguenza l'interfaccia dell'oggetto server deve essere conosciuta dal client a tempo di compilazione. Nel caso dinamico è introdotta la possibilità di costruire ed invocare richieste su oggetti non conosciuti a tempo di compilazione.

CORBA supporta, quindi, sia la trasparenza di comunicazione, mascherando le chiamate remote come semplici invocazioni locali, sia quella di locazione, nascondendo al client tutti i dettagli sulla localizzazione degli oggetti sulla rete e sul loro linguaggio di implementazione.

La comunicazione tra ORB nella rete avviene tramite i protocolli GIOP e IIOP che implementano il marshalling, l'unmarshalling e la rappresentazione esterna dei dati.

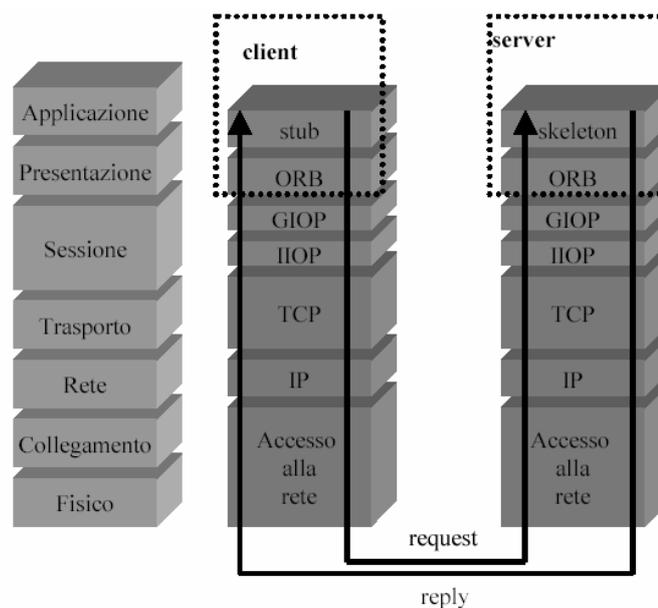


Fig. II.5 Collocazione dei protocolli GIOP-IIOP rispetto alla pila OSI.

GIOP è un protocollo astratto che specifica l'insieme di tipi di messaggi utilizzabili tra un client e un server, una sintassi standard nel trasferimento dati delle interfacce IDL, un formato standard per ogni messaggio utilizzabile. In più GIOP richiede un sistema di trasporto con connessione: IIOP è il mapping specifico di GIOP sul protocollo TCP/IP.

In aggiunta all'ORB, CORBA definisce un framework di oggetti supplementari, limitandosi a fornire le specifiche ma non le implementazioni:

- gli oggetti dei servizi (*Object Services*) costituiscono un insieme di servizi di uso generico per l'implementazione di applicazioni distribuite;
- le facilitazioni comuni (*Common Facilities*) sono un insieme di componenti che forniscono funzionalità applicative di uso comune, come stampa, accesso a DB, accesso a servizi di e-mail;
- le interfacce di dominio (*Domain Interfaces*) specificano componenti che forniscono servizi specifici per particolari domini applicativi;
- gli oggetti delle applicazioni (*Application Interfaces*) corrispondono alla nozione classica di applicazione prodotta da una particolare organizzazione e di conseguenza non sono standardizzati.

Di seguito è fornita una descrizione sommaria di alcuni *object service* particolarmente importanti per i sistemi distribuiti: il *Naming Service*, il *Trading Service*, l'*Event Service* ed il *Notification Service*.

Il *Naming Service* permette ad un client di trovare un oggetto attraverso il suo nome e ad un server di registrare i suoi oggetti dandogli un nome.

Il *Trading Service* è un servizio di locazione di oggetti, così come il *Naming Service*, ma lavora ad un livello più astratto. Nel *Naming Service* gli oggetti vengono ricercati attraverso un nome. Nel *Trading Service* si fanno delle richieste per ricevere liste di oggetti che soddisfano certe caratteristiche o proprietà (inserite nella richiesta) [34].

L' *Event Service* è il servizio che si occupa della gestione di eventi CORBA, dove un evento è definito come una occorrenza di oggetto che è di interesse per uno o più oggetti [44], fornendo in tal modo un supporto per la comunicazione asincrona ad accoppiamento lasco tra oggetti. Per disaccoppiare la comunicazione tra un produttore di eventi (supplier) ed un consumatore (consumer) l'*Event Service* prevede l'astrazione di un canale di eventi (*Event Channel*) che funge da consumer per i supplier e da supplier per i consumer. In tal modo un supplier notifica l'evento senza conoscere i consumers interessati.

La figura II.6 illustra i componenti dell' *Event Service* sopra descritti.



Fig. II.6 Componenti di CORBA Event Service

Il *Notification Service* è una estensione dell' *Event Service* ed affronta le seguenti problematiche:

- strategie di filtraggio eventi ( filtering );
- mantenimento di informazioni sullo stato dell' *Event Channel*;
- definizione di parametri di qualità del servizio di consegna degli eventi;

La figura II.7 [45] illustra i componenti del *Notification Service* e mette in evidenza gli elementi che fanno del *Notification Service* una estensione dell' *Event Service*.

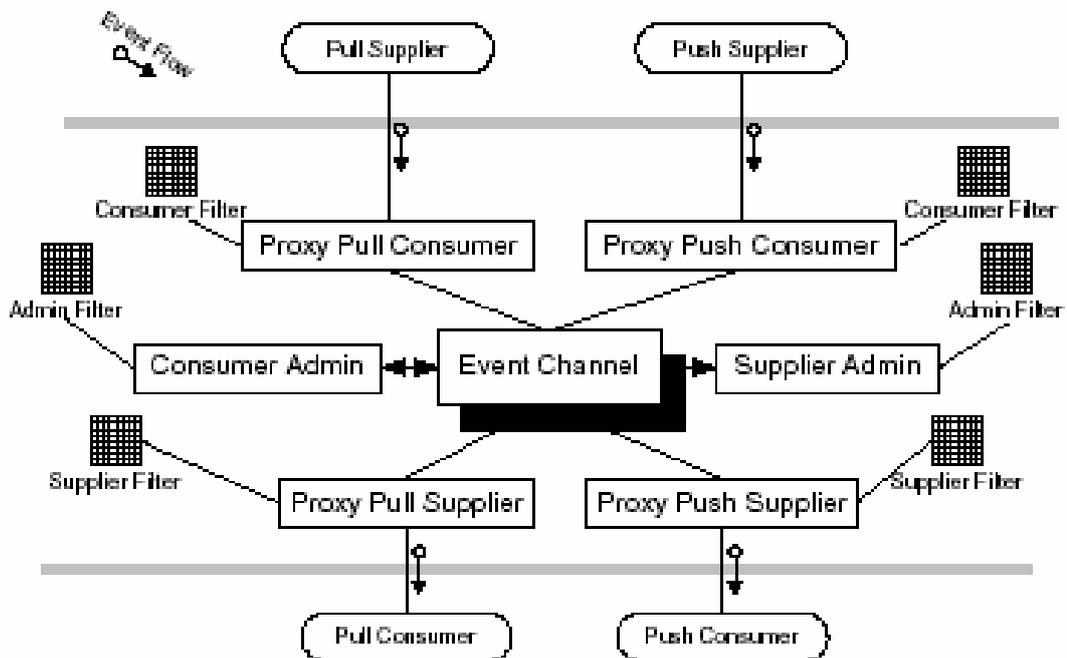


Fig. II.7 Componenti del CORBA Notification Service

CORBA ha riscontrato molto successo nelle industrie e nel campo della ricerca: infatti, implementazioni dello standard sono disponibili presso un gran numero di rivenditori ed esistono persino versioni freeware.

CORBA supporta tutti i maggiori linguaggi di programmazione ed è adattabile per quasi tutte le combinazioni di hardware e sistemi operativi. Tuttavia non è adatto a piccoli dispositivi ed a sistemi altamente dinamici, per i quali sono adottati altri sistemi.

### II.3.2 JINI

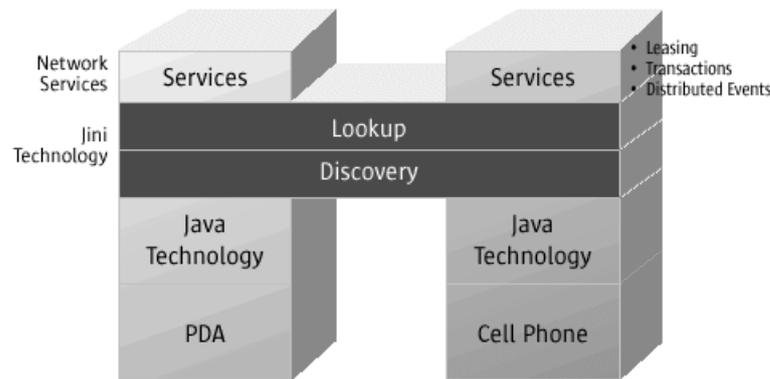
Jini è un'infrastruttura basata sul top di Java ed RMI per creare una federazione fra dispositivi e componenti software che implementano servizi. Jini rende possibile per qualsiasi dispositivo in grado di eseguire una Java Virtual Machine di interoperare con gli altri, offrire ed utilizzare servizi.

Un servizio è definito come un'entità che può essere usata da una persona, da un programma o da un altro servizio. Tipici esempi di servizi sono la stampa di un documento o la traduzione di dati da un formato ad un altro, ma è possibile considerare servizi anche dispositivi con particolari funzionalità hardware.

I client possono usare un particolare servizio senza avere una conoscenza a priori della sua implementazione, scaricando quando necessario un oggetto proxy.

L'abilità di Jini di creare spontaneamente una federazione di servizi che sia robusta e tollerante ai guasti, è basata su una serie di concetti fondamentali:

- *Discovery*: processo attraverso il quale un client o un servizio localizza il sistema di nomi all'interno della rete;
- *Join*: meccanismo che permette di registrare un nuovo servizio sul sistema di nomi locale o remoto;
- *Lookup*: processo attraverso il quale il client interroga il sistema di nomi per ricercare il servizio richiesto.



**Fig. II.8 Architettura di Jini.**

### II.3.2.1 DISCOVERY

I servizi usano un meccanismo standard per registrarsi ed entrare a far parte della federazione: innanzitutto, interrogano la rete locale per localizzare il Lookup Service.

Il Lookup Service è un sistema che tiene traccia dei servizi che hanno notificato la loro presenza sulla rete.

Poiché questo componente è di vitale importanza, è previsto che, per aumentare la robustezza e l'affidabilità del sistema, vi possano essere più Lookup Service in esecuzione contemporaneamente su macchine diverse.

La ricerca avviene, a seconda che si cerchi in una rete locale (LAN) o geografica (WAN), attraverso richieste *multicast* o *unicast*. Il multicast discovery è utilizzato per trovare un Lookup Service che contiene servizi che appartengono alla locale comunità Jini, mentre l'*unicast* discovery permette di localizzare Lookup service che contengono servizi remoti.

### II.3.2.2 JOIN

Quando un servizio ha terminato la fase di Discovery e ha trovato un Lookup Service, ha l'obbligo di registrarsi presso il Lookup Service stesso. Il servizio invia un oggetto proxy (*service object*) e gli attributi ad esso associati. Il service object contiene la descrizione dell'interfaccia per l'utilizzo del servizio, nonché l'implementazione dei metodi che verranno utilizzati dalle applicazioni.

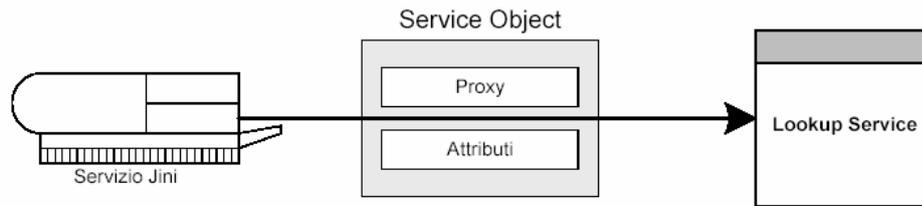


Fig. II.9 Registrazione del servizio.

I servizi sono tenuti a rinnovare la propria registrazione dopo un tempo stabilito al momento della registrazione stessa (*leasing*). Allo scadere del periodo di leasing, se il servizio non rinnova la registrazione, viene cancellato dalle entry del servizio di Lookup.

### II.3.2.3 LOOKUP

Questa fase avviene quando un client ha già effettuato la fase di Discovery con successo e vuole richiedere un servizio Jini che abbia determinate caratteristiche. La richiesta al Lookup Service avviene specificando uno o più dei seguenti campi:

- *attributi*: viene effettuata una comparazione con gli attributi registrati all'atto del Join dei servizi;
- *interfaccia*: si specifica un'interfaccia che il servizio poi implementa. Si può, ad esempio, specificare un'interfaccia stampante per poi scegliere tra i *service provider* che la implementano;
- *ID number*: ad ogni servizio Jini, all'atto della registrazione, è assegnato un identificativo che può essere utilizzato nella fase di Lookup per richiedere un servizio ben preciso.

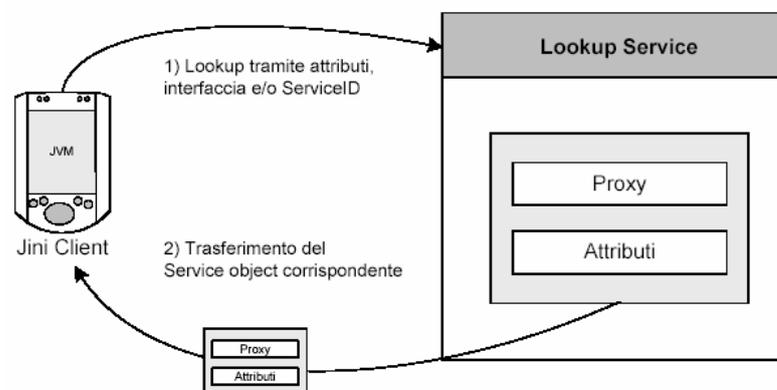


Fig. II.10 Lookup di un servizio.

Se gli attributi specificati dal client corrispondono ad un servizio registrato nel Lookup Service, quest'ultimo restituisce il service object corrispondente che verrà usato per invocare il servizio [23].

La tecnologia Jini, anche se particolarmente adatta per lo sviluppo di sistemi dinamici, presuppone che ogni componente della federazione sia in grado di eseguire una Java Virtual Machine. L'ipotesi non è banale e non è sempre possibile, non solo per mancanza di risorse, ma anche per questioni di conflittualità di interessi: vedi Microsoft contro SUN.

Per questi motivi, sono stati ricercati nuovi modelli basati su tecnologie aperte e popolari, come ad esempio i *Web Services*.

### II.3.3 WEB SERVICES

Il modello dei *Web Services* [41] è un insieme di standard per la pubblicazione, il discovery e la composizione di servizi indipendenti in una rete aperta [6].

Il concetto di web service è molto simile a quello di oggetto, nel significato con cui lo si usa nella programmazione Object-Oriented: un oggetto è un modulo software che offre una serie di funzioni utilizzabili dall'esterno da parte di altro software, tramite una interfaccia di comunicazione dichiarata dall'oggetto stesso.

Così come per CORBA ed RMI, anche un Web Service offre una funzionalità (servizio), ad altri client sulla rete attraverso una interfaccia ben definita. La differenza è che in questo caso il servizio è posto sul web, e l'integrazione con il servizio avviene attraverso lo scambio di messaggi sulla rete.

La forza dei Web Services è di utilizzare un set base di protocolli disponibili ovunque, permettendo l'interoperabilità tra piattaforme molto diverse e mantenendo comunque la possibilità di utilizzare protocolli più avanzati e specializzati per effettuare compiti specifici. I protocolli alla base del modello dei Web Services sono quattro:

- XML è lo standard usato per rappresentare i dati trasportati;
- SOAP è lo standard usato per definire il formato dei messaggi scambiati;
- WSDL è lo standard usato per descrivere il formato dei messaggi da inviare al Web Service, quali sono i metodi esposti, quali sono i parametri ed i valori di ritorno.

- UDDI è lo standard promosso dall'omonimo consorzio, che ha come scopo quello di favorire lo sviluppo, la scoperta e l'interoperabilità dei servizi Web.

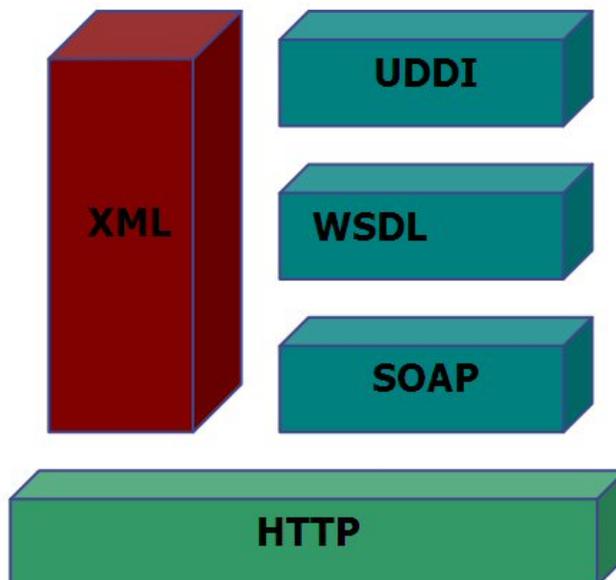


Fig. II.11 Protocolli standard dei Web Services.

### II.3.3.1 WSDL

I web services sono visti come software disponibile sul web, utilizzabile da altri web services o da utenti: di conseguenza essi devono essere descritti cosicché gli altri componenti possano usarli facilmente, disaccoppiando interfaccia ed implementazione.

Il Web Service Description Language [42] è il linguaggio standard per la descrizione delle interfacce dei web services (l'equivalente di IDL per CORBA).

Esso presenta le caratteristiche architettoniche di molti altri linguaggi basati su XML nati di recente: infatti fa riferimento e si integra con standard esistenti, evitando di ridefinire ciò che è già stato definito ed inoltre predilige l'uso di XML Schema per il type system e di SOAP per la definizione dei messaggi.

WSDL si presenta, quindi, come un formato XML per descrivere servizi di rete come un insieme di punti terminali, detti *porte*, operanti su messaggi contenenti informazioni di tipo "documentale" o "procedurale".

Esso opera una chiara distinzione tra i messaggi e le porte: i messaggi, ossia la sintassi e la semantica proprie di un servizio Web, sono sempre astratti, mentre le porte, l'indirizzo di rete grazie al quale è possibile richiamare il servizio Web desiderato, sono sempre concrete.

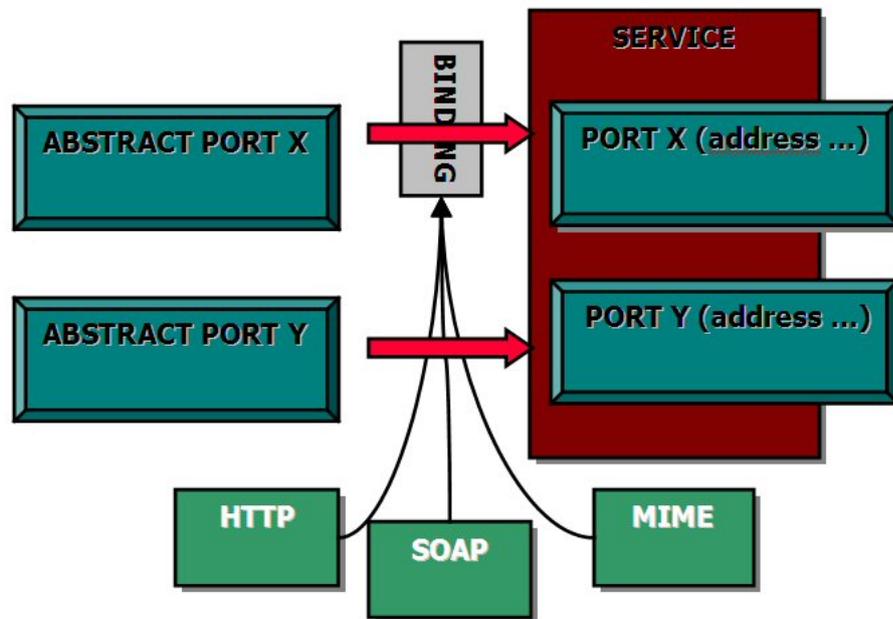


Fig. II.12 Dalla descrizione astratta a quella concreta.

All'utente non viene chiesto di fornire informazioni sulla porta in un file WSDL. Un file WSDL può contenere unicamente informazioni di interfaccia astratte e non può fornire dati di implementazione concreta. Sono questi i requisiti da rispettare affinché un file WSDL sia considerato valido.

WSDL separa, dunque, gli aspetti "astratti" della descrizione di un servizio da quelli "concreti" che lo legano a particolari protocolli di rete o di RPC, consentendo ad uno stesso servizio di poter avere implementazioni diverse, basate sulla stessa descrizione, garantendo in questo modo che i sistemi possano comunicare tra di loro e favorendo il riutilizzo delle descrizioni astratte per la creazione di nuovi servizi.

### II.3.3.2 UDDI

L'Universal Description, Discovery and Integration (UDDI) [33] Service è un'iniziativa supportata da IBM, Microsoft, e HP, e fornisce ai client un meccanismo per trovare dinamicamente web services.

UDDI è un registro pubblico progettato per contenere informazioni strutturate sulle aziende e i rispettivi servizi. Con UDDI, è possibile pubblicare e individuare informazioni su un'azienda e i servizi Web da essa proposti.

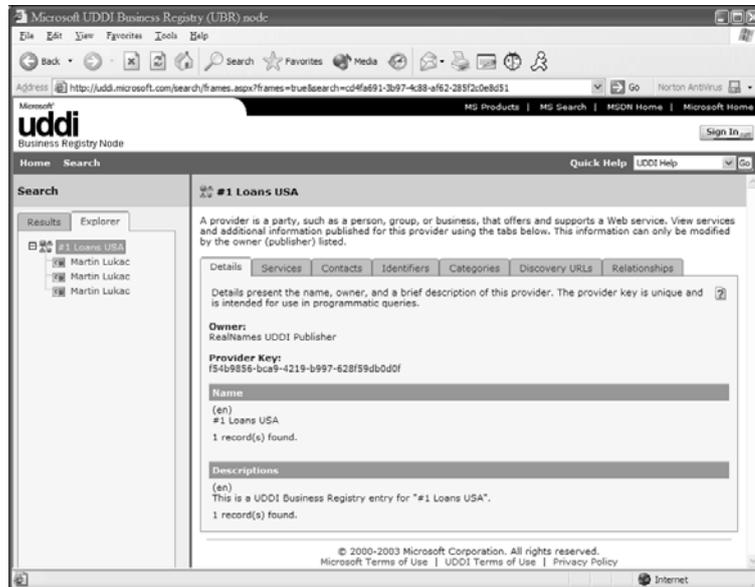


Fig. II.13 Esempio di registro UDDI della Microsoft.

Questi dati possono essere classificati mediante tassonomie standard, in modo da facilitare la ricerca delle informazioni per categoria.

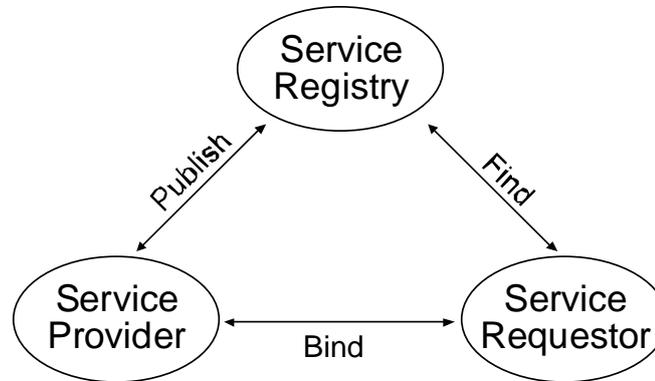
Particolare di primaria importanza, in UDDI sono disponibili informazioni sulle interfacce tecniche dei servizi delle aziende. Mediante un gruppo di chiamate di API XML basate su SOAP, è possibile interagire con UDDI, sia in fase di progettazione che in fase di esecuzione, per rilevare dati tecnici che consentono l'attivazione e l'utilizzo di tali servizi.

Utilizzato in questo modo, UDDI funge da infrastruttura per una configurazione software basata su servizi Web.

Lo scenario nel quale trova pieno utilizzo l'UDDI può essere ricondotto a tre fasi fondamentali:

- pubblicazione (*publishing*): il fornitore del servizio per renderlo pubblico contatta il *service registry* che provvede ad inserirlo nel registry tramite UDDI;
- ricerca (*finding*): alla richiesta di un servizio, il *service registry* provvede a cercare quelli che meglio rispondono alle esigenze del richiedente;

- collegamento (*binding*): nel momento in cui il service registry fornisce la risposta può stabilirsi il collegamento tra il richiedente del servizio web e il fornitore.



**Fig. II.14 Scenario di utilizzo di UDDI.**

La pubblicazione sul registro UDDI è un processo relativamente semplice e consiste nel raccogliere alcune informazioni di base relative al modo in cui si desidera modellare in UDDI la voce che definisce l'azienda e i relativi servizi, dopo di che si passa alla registrazione vera e propria, adoperando un'interfaccia utente basata su Web o tramite un programma.

Le informazioni alla base della registrazione sono così classificate:

- *white pages*, che forniscono il nome dell'azienda, una breve descrizione, se necessario, in più lingue, le persone da contattare per ottenere informazioni sui servizi Web offerti ed altri identificativi;
- *yellow pages*, che classificano azienda e servizi attraverso tassonomie standard. Le tassonomie attualmente supportate sono NAICS (*North American Industry Classification System*), UNSPSC (*Universal Standard Products and Services Codes*), ISO 3166, SIC (*Standard Industry Classification*) e GeoWeb Geographic Classification;
- *green pages*, che contengono informazioni tecniche circa i web service che sono esposti attraverso il concetto di tModel. La struttura tModel, abbreviazione di "Technology Model", rappresenta le tracce tecniche, le interfacce e i tipi astratti dei metadati. I file WSDL sono esempi perfetti dei tModel UDDI.

Per ricercare il servizio sono utilizzate una serie di API che vengono esposte per fare browsing su un repository site e per prelevare le informazioni che occorrono

riguardo al servizio desiderato. Esistono anche API, destinate a chi pubblica nuovi web services, per l'inserimento di nuove informazioni in questo repository.

Infine, una volta trovato il web service di cui si aveva bisogno, viene effettuato il binding tra interfaccia e implementazione. Anche UDDI, infatti, opera una distinzione simile a quella fatta da WSDL tra astrattezza e implementazione e lo fa attraverso il concetto di tModel. A supporto dei tModel, vi sono i modelli di binding, ossia le implementazioni concrete di uno o più tModel. E' nel modello di binding che si registra il punto di accesso per un'implementazione specifica di un tModel.

Il registro UDDI e il linguaggio WSDL fungono da specifiche complementari per la definizione di configurazioni software basate sui servizi Web.

Il linguaggio WSDL fornisce un metodo formale, non legato a produttori specifici, per definire i servizi Web, rendendo possibile l'utilizzo delle chiamate di procedure remote dell'ultima generazione, mentre UDDI fornisce una vasta infrastruttura standardizzata che consente agli utenti di descrivere e individuare i servizi Web. L'utilizzo combinato dei due standard consente la nascita di un vero ecosistema di servizi Web.

## II.4 LA COMUNICAZIONE IN AMBIENTI PERVASIVI

Gli ambienti pervasivi o ubiqui sono caratterizzati da un gran numero di entità autonome dette *agenti*, che contribuiscono a trasformare uno spazio fisico in un ambiente intelligente e computazionalmente attivo. Queste entità possono essere dispositivi, servizi, utenti.

Nonostante siano stati sviluppati vari tipi di *middleware* ed infrastrutture per rendere possibile la comunicazione tra agenti, tra cui quelli presentati precedentemente, nessuno di questi prevede meccanismi per supportare l'interoperabilità semantica tra essi [26].

Nei sistemi distribuiti, lo scambio di messaggi rappresenta, come si è visto, un aspetto fondamentale. I messaggi, che contengono descrizioni di entità, servizi, eventi ed altri concetti, possono essere raggruppati essenzialmente in tre categorie:

- *Advertisement*: corrisponde all'invio di descrizioni di servizi offerti, di dispositivi, interfacce per registri.

- *Notification*: corrisponde all'invio di descrizioni di eventi, ad esempio l'arrivo di un'entità nell'ambiente.
- *Query*: corrisponde alla richiesta di servizi o entità che rispondono a certi requisiti e alla ricezione delle descrizioni di servizi ed entità che li soddisfano.

Per consentire lo scambio di tali messaggi tra due agenti occorre necessariamente che il loro formato sia comune ed inoltre che siano comuni interfacce e protocolli. E' difficile, poi, aspettarsi che le diverse entità comprendano la semantica dell'ambiente e delle altre entità quando interagiscono fra loro. Deve essere quindi conosciuta o scoperta anche la semantica dei messaggi, ossia il vocabolario dei messaggi, che include i nomi e i valori validi degli elementi.

Mentre per sistemi semplici o chiusi, tutti gli schemi richiesti per interpretare il contenuto dei messaggi sono compilati nei componenti del sistema stesso, in un sistema aperto le parti che comunicano sono autonome, eterogenee ed evolvono nel tempo.

Occorre, quindi, un modello aperto che renda possibile la scoperta e l'utilizzo degli schemi quando occorrono e mentre il sistema è in esecuzione.

#### **II.4.1 L'INTEROPERABILITÀ: ESIGENZA DI NUOVI STRUMENTI**

I registri di oggetti, come il CORBA Naming Service, forniscono un meccanismo base per la ricerca di servizi della cui esistenza si è già a conoscenza. Brokers, come il CORBA Trading Service, forniscono la capacità di localizzare servizi in base ad alcuni attributi degli stessi.

Altri servizi che forniscono caratteristiche simili sono, ad esempio, JINI, LDAP, Microsoft's Registry. Questi standard definiscono interfacce e formati per le descrizioni ma non ne definiscono il contenuto, la semantica. Ad esempio, il CORBA Trading Service è un'interfaccia standard per un broker, ma non definisce né le proprietà dei servizi registrati né i valori legali di tali proprietà. Le regole di matching sono limitate e applicabili solo agli attributi definiti esplicitamente. Inoltre, mentre per le query esiste una sintassi, non esiste un linguaggio dichiarativo per definire le tipologie di servizi e le loro istanze.

Allo stesso modo, Jini definisce architettura, protocolli e interfacce per lo scambio di tutte le tipologie di messaggi viste precedentemente. I servizi e le entità sono

descritte dai Service Entry, ma come nel caso di CORBA, la loro definizione è lasciata agli applicativi. Non sono definiti, inoltre, né un linguaggio standard per la definizione di schemi né meccanismi per la gestione e la validazione delle Service Entry.

I Web Services, sebbene siano nati soprattutto per superare i problemi di interoperabilità tra piattaforme, stanno cercando di risolvere anche questi problemi di service discovery e di gestione di descrizioni provenienti da entità autonome. Tuttavia anche per i Web Services non è ancora stato definito il livello semantico, ossia non sono stati definiti standard per la definizione, validazione e scambio di schemi per le descrizioni di entità e servizi e dei loro modelli tecnici. Il Web Semantico è destinato a definire questi aspetti.

## II.5 WEB SEMANTICO

Il termine Web Semantico [7] fu coniato per la prima volta da Tim Berners-Lee, l'ideatore del WWW.

Si tratta di un insieme di standard aperti, indipendenti dalle tecnologie, nati per consentire lo scambio di descrizioni di entità e relazioni allo scopo di migliorare le capacità di ricerca nel Web. Comunque questi standard sono ben adattabili ad alcuni dei requisiti di un sistema pervasivo.

Il Web Semantico rappresenta un'estensione del Web attualmente esistente che, in aggiunta, mira alla cooperazione tra uomo e calcolatori.

Gran parte del contenuto di Internet è progettato per essere letto da esseri umani e non per essere trattato da programmi ed è ben lontano dal poter fornire una solida piattaforma che renda possibile un'interpretazione e una comprensione semantica da parte di agenti automatici.

Questo è ciò che viene fornito dal Web Semantico e che può essere riassunto come segue:

- l'informazione deve essere *machine-readable*, ossia non più pensata per essere letta direttamente dall'uomo, ma mirata ad essere in un formato facilmente elaborabile dalla macchina, da agenti intelligenti, servizi specializzati, siti web personalizzati e motori di ricerca potenziati semanticamente;

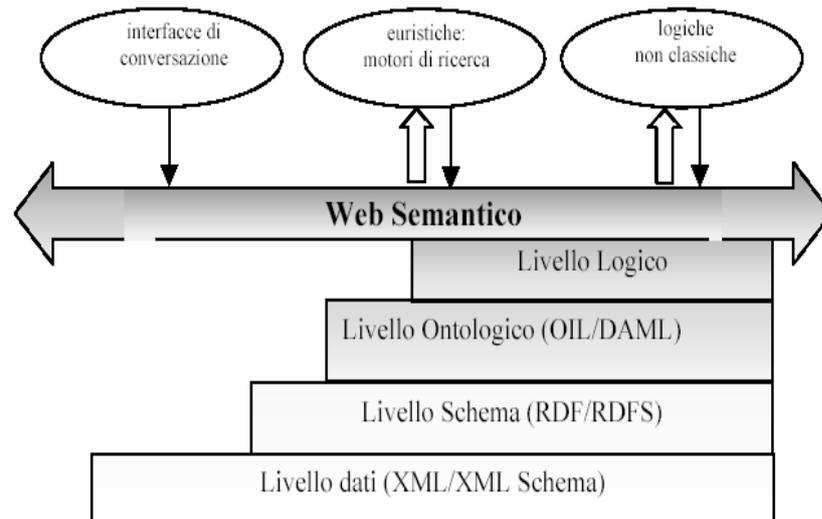
- deve essere fornito un supporto per *l'interoperabilità sintattica*, intesa come la facilità di leggere dati e ottenere una rappresentazione utilizzabile dalle applicazioni;
- deve essere fornito un supporto per *l'interoperabilità a livello semantico*: non sono più sufficienti standard per la forma sintattica dei documenti, ma anche per il loro contenuto semantico; interoperabilità semantica significa definire *mapping* tra termini sconosciuti e termini conosciuti nei dati;
- il formato utilizzato per lo scambio dei dati deve permettere di poter esprimere qualsiasi forma di dati, poiché non è possibile anticiparne tutti i suoi usi potenziali (*potere espressivo universale*). Per raggiungere questo obiettivo, è necessario basarsi su un modello comune di grande generalità. Solo così qualsiasi “prospettiva” può trovare espressione all'interno del modello.

Il Web Semantico, quindi, mira a riportare chiarezza, formalità e organizzazione dei dati, collegando le informazioni a concetti astratti organizzati in una gerarchia, a sua volta descritta in un meta-documento. In questo modo vari agenti automatici hanno la possibilità di cogliere il contesto semantico di una fonte informativa interpretando le varie relazioni esistenti tra le risorse, formulando asserzioni sulle stesse, nonché controllando la loro attendibilità.

### II.5.1 ARCHITETTURA E LINGUAGGI DEL WEB SEMANTICO

Nella visione di Tim Berners-Lee, il Web Semantico è un'architettura strutturata su almeno quattro livelli:

- il livello dei dati (un semplice modello dei dati e una sintassi per i metadati);
- il livello schema (una base per la definizione di un vocabolario);
- il livello ontologico (per la definizione delle ontologie);
- il livello logico (supporto al ragionamento).



**Fig. II.15 Architettura del Web Semantico.**

Il Web Semantico, a differenza del Web che si fonda su documenti, si fonda su risorse. Esse sono descritte adoperando i linguaggi RDF e RDF Schema, entrambi basati su tecnologie XML. Questo tuttavia non è sufficiente perché occorre catturare la semantica delle risorse descritte e delle relazioni tra esse e per fare ciò occorre introdurre vocabolari ontologici.

Una volta che le risorse sono state ben definite, non solo sintatticamente ma anche semanticamente, è possibile estendere questi vocabolari ontologici con logiche che supportano ragionamenti automatici.

### II.5.2 XML E XML SCHEMA

XML (eXtension Markup Language) [37] è un linguaggio di markup destinato alla descrizione di documenti, strutturati in maniera arbitraria. A differenza di HTML, che è utilizzato per descrivere la modalità con cui documenti ipertestuali con strutture fisse sono visualizzati, XML separa il contenuto del documento dalla sua modalità di visualizzazione. E' dunque uno standard elaborato per garantire l'interoperabilità sintattica, ossia per fare in modo che le informazioni non siano semplicemente formattate per facilitarne il reperimento per un utente umano, ma anche facilmente elaborabili da agenti software.

```
<?xml version="1.0" ?>
<person>
```

```
<name>Marco</name>
<personal_info>
  <office_number>18</office_number>
  <phone_number>1024</phone_number>
</personal_info>
</person>
```

**Fig. II.16 Esempio di file XML.**

XML definisce una struttura ad albero per i documenti dove ciascun nodo individua un tag ben definito (*elemento*) mediante il quale è possibile in qualche modo interpretare le informazioni che esso racchiude (*attributi*).

E' possibile imporre delle restrizioni con cui i tag XML possono essere usati e quali annidamenti di tali tag sono permessi mediante l'uso di due tecnologie:

- **DTD (*Document Type Definition*):** Contengono le regole che definiscono i tag usati nel documento XML, in altre parole ne definiscono la struttura. Questi possono essere dei file esterni o specificati direttamente all'interno del documento.
- **XML Schema:** Spesso i documenti condividono una struttura specifica per un certo dominio, per consentire sia al software sia agli umani di riconoscere il contenuto dell'XML si richiede che tali strutture siano documentate in un formato comprensibile ad entrambi. A tal scopo è stato introdotto XML Schema, il quale permette di definire un vocabolario utilizzabile per descrivere documenti XML facendo uso della loro stessa sintassi. Le sue specifiche assumono che si faccia uso di almeno due documenti, un'istanza ed uno schema. Il primo contiene le informazioni che interessano realmente, mentre il secondo descrive struttura e tipo del precedente.

XML, tuttavia, non gestisce la semantica dei contenuti: essa non è specificata in modo esplicito, ma è "incorporata" nei nomi dei tag, ossia il vocabolario degli elementi e le loro combinazioni non sono prefissati, ma possono essere definiti ad hoc per ogni applicazione. Tale semantica, quindi, non è definita formalmente e può risultare eventualmente comprensibile solo all'uomo e non alla macchina: un'entità software riconosce i contenuti, ma non è in grado di attribuire loro un significato.

XML, quindi, fornisce l'interoperabilità sintattica ma non riesce a garantire quella semantica né a fornire meccanismi di classificazione o ragionamento; pertanto, deve necessariamente essere affiancato ad altri linguaggi più potenti.

### II.5.3 RDF E RDF SCHEMA

RDF (Resource Description Framework) [39] è lo standard che consente l'aggiunta di semantica a un documento e quindi si pone ad un livello direttamente superiore rispetto ad XML. RDF è, in un certo senso, un'applicazione di XML: se XML è un'estensione del documento, RDF può essere visto come un'estensione dei dati introdotti da XML.

Il modello base dei dati RDF è composto da:

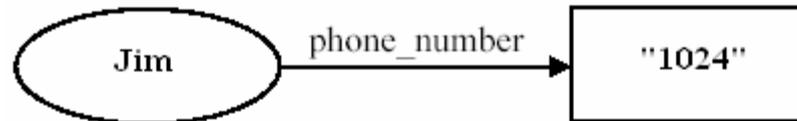
- *Risorse*: con questo termine si intende qualsiasi cosa possa essere descritta. Una risorsa può essere ad esempio una pagina Web oppure un qualsiasi oggetto anche se non direttamente accessibile via Web (ad esempio un libro, una persona). Una risorsa viene identificata univocamente attraverso un URI.
- *Proprietà*: una proprietà è una caratteristica, una relazione che descrive una risorsa. Il significato, l'insieme di valori che può assumere, i tipi di risorse a cui può riferirsi sono tutte informazioni reperibili dallo schema RDF in cui essa è definita.
- *Asserzioni*: un'asserzione è costituita da un soggetto (la risorsa descritta), un predicato (la proprietà) e un oggetto (il valore attribuito alla proprietà), dove l'oggetto può essere una semplice stringa o un'altra asserzione.

Volendo fare un paragone con un database relazionale potremmo dire che una riga di una tabella è una risorsa RDF, il nome di un campo (una colonna) è il nome di una proprietà RDF, il valore del campo è il valore della proprietà.

Un semplice esempio di utilizzo del modello di RDF può essere fornito dalla seguente asserzione:

**Jim's phone\_number is 1024**

che è stato graficamente schematizzato in figura II.17.



**Fig. II.17** Semplice esempio di un'asserzione RDF.

La precedente asserzione è formalizzata in RDF/XML come segue:

```
<?xml version="1.0" ?>
```

```
<rdf:RDF
```

```
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://www.example.com/props/"
  xmlns:base="http://www.organization.com/people">
```

```
  <rdf:Description rdf:about="Jim">
```

```
    <s:Phone_number>1024</s:Phone_number>
```

```
  </rdf:Description>
```

```
</rdf:RDF>
```

**Fig. II.18** Esempio di documento RDF/XML.

Sintatticamente, i concetti espressi con RDF vengono serializzati mediante XML. RDF definisce, quindi, un semplice modello dei dati per descrivere proprietà e relazioni fra risorse. In RDF però non esistono livelli di astrazione: ci sono le risorse e le loro relazioni, tutte organizzate in un grafo piatto. In altri termini non è possibile definire tipi (o classi) di risorse con loro proprietà specifiche.

Vista l'utilità di poter definire classi di risorse, RDF è stato arricchito con un semplice sistema di tipi detto RDF Schema.

Il sistema di tipi RDF Schema ricorda i sistemi di tipi dei linguaggi di programmazione object-oriented (Java-like). Una risorsa può, per esempio, essere definita come istanza di una classe (o di più classi) e le classi possono essere organizzate in modo gerarchico. RDF Schema utilizza il modello RDF stesso per definire il sistema di tipi RDF, fornendo un insieme di risorse e proprietà predefinite che possono essere utilizzate per definire classi e proprietà a livello utente. E' possibile, inoltre, definire vincoli di dominio e di range sulle proprietà ed alcuni tipi

di relazioni (comprese quelli di sottoclasse di una risorsa e sottotipo di una proprietà). L'insieme di tali elementi è detto vocabolario dell'RDF Schema.

Il linguaggio di specifica RDFS è un linguaggio dichiarativo poco espressivo ma molto semplice da implementare. Si esprime attraverso la sintassi di serializzazione RDF/XML e si avvale del meccanismo dei *namespace* XML per la formulazione delle URI che identificano in modo univoco le risorse (definite nello schema stesso) consentendo il riutilizzo di termini definiti in altri schemi.

Concetti e proprietà già dichiarati per un dominio possono essere impiegati di nuovo o precisati per incontrare le esigenze di una particolare comunità di utenti.

Sebbene, in prima battuta, RDF e RDF Schema sembrano essere uno strumento buono per la definizione di un linguaggio di markup per il Web Semantico (ad esempio, per determinare le relazioni semantiche tra termini differenti), in realtà essi mostrano di non avere sufficiente potere espressivo: non consentono, infatti, di specificare le proprietà delle proprietà, le condizioni necessarie e sufficienti per l'appartenenza alle classi e gli unici vincoli che si possono definire sono quelli di dominio e range delle proprietà.

Inoltre, il loro utilizzo nei sistemi di rappresentazione della conoscenza è limitato da un'altra caratteristica: non permettono di specificare meccanismi di ragionamento, ma rappresentano semplicemente un sistema a frame.

I meccanismi di ragionamento devono essere costruiti, quindi, ad un livello superiore.

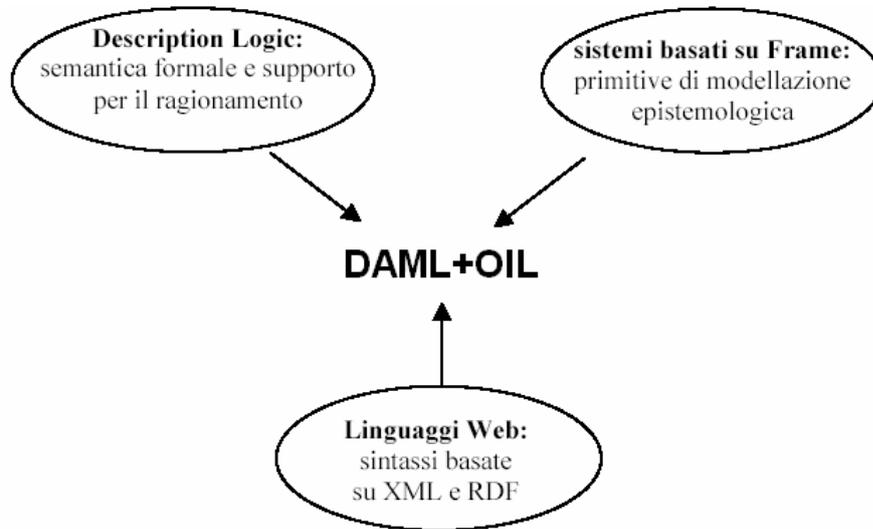
#### **II.5.4 DAML+OIL**

Il linguaggio DAML+OIL [12] è uno standard che consente la rappresentazione delle informazioni in modo che il loro significato sia comprensibile alle macchine.

I due risultati più interessanti degli ultimi anni in fatto di sviluppo verso il Web Semantico, DAML-ONT e OIL, si sono fusi in un unico linguaggio, DAML+OIL appunto, che incorpora caratteristiche provenienti dal lavoro del gruppo americano (DARPA) e del gruppo europeo (progetto On-To-Knowledge, IST).

DAML+OIL riesce a garantire l'interoperabilità sintattica e semantica sfruttando la sintassi e le caratteristiche dei linguaggi che si trovano ai livelli più bassi dello stack del WEB Semantico, ossia XML e RDF, ed in più aggiunge, rispetto a questi, una maggiore forza espressiva ottenuta a partire da primitive di modellazione

ereditate dai sistemi basati su Frame e la capacità di produrre asserzioni in logica formale ed effettuare ragionamenti in modo automatico, traendo spunto dalle Description Logics.



**Fig. II.19 Il modello di DAML+OIL.**

#### II.5.4.1 SISTEMI BASATI SU FRAME

I sistemi basati su *Frame* somigliano agli approcci orientati agli oggetti, ma considerano le cose da un punto di vista differente.

Le primitive di modellazione sono le classi (o frame), ognuna delle quali ha determinate proprietà (o slot), chiamate attributi. Essi non hanno una visibilità globale, ma sono applicabili alle sole classi per cui sono definiti.

Un frame fornisce un certo contesto per la modellazione di un aspetto del dominio in esame. Dai sistemi basati su Frame, DAML+OIL eredita le primitive di modellazione essenziali.

Esso, infatti, è basato sulla nozione di classe, la definizione delle sue proprietà e delle sue superclassi e sottoclassi. Le relazioni possono essere definite come entità indipendenti aventi un certo dominio e intervallo. Come le classi, anche le relazioni possono essere organizzate in una gerarchia.

Ad incrementare il potere espressivo contribuiscono, poi, due aspetti: i costruttori e i tipi di assiomi.

I costruttori permettono la creazione, oltre che di classi intese nel senso classico, con un nome ed associate ad un URI, anche di classi intese come espressioni, cioè come il prodotto di un certo numero di operatori.

Una classe può essere ottenuta dall'unione o dall'intersezione di altre classi, elencandone gli elementi, può essere complementare di un'altra, può essere definita come collezione degli elementi che hanno almeno o al più un certo numero di valori distinti di una proprietà.

Il secondo aspetto è rappresentato dagli assiomi, che permettono la dichiarazione di relazioni di classificazione o equivalenza tra classi e/o proprietà, la disgiunzione tra classi, l'equivalenza o meno di oggetti diversi, le proprietà delle proprietà.

#### II.5.4.2 DESCRIPTION LOGICS

Le *Description Logics* rappresentano una classe di logiche che sono specificatamente designate a modellare vocabolari. Esse descrivono la conoscenza in termini di concetti (paragonabili ai frame) e restrizioni (paragonabili agli slot) che sono utilizzate per derivare automaticamente classificazioni tassonomiche.

DAML+OIL eredita dalle Description Logics la semantica formale e il supporto per il ragionamento, stabilendo una corrispondenza tra concetti logici e tag DAML+OIL che li rappresentano.

A partire da un documento DAML+OIL è possibile, infatti, creare una *Knowledge Base*: una Kb è un database con capacità di ragionare in maniera automatica, in grado non solo di rispondere a query grazie a dei match ma anche effettuando ragionamenti.

Una KB è formata da due componenti:

- *Intensionale*: uno schema che definisce classi, proprietà e relazioni tra classi (Tbox, ossia *Terminological knowledge*);
- *Estensionale*: un'istanza (parziale) dello schema, contenente asserzioni su individui (Abox, ossia *Assertional knowledge*).

Il Tbox è il modello di ciò che può essere vero, l'Abox è il modello di ciò che correntemente è vero. Effettuando ragionamenti sulla KB, è possibile ottenere risposte a questioni importanti come:

- la *satisfiability* di un concetto: se un concetto esiste;
- la *subsumption*: se un concetto è un caso di un altro concetto;
- la *consistence*: se l'intera KB verifica la *satisfiability*;
- l'*instance checking*: se un'asserzione verifica la *satisfiability*.

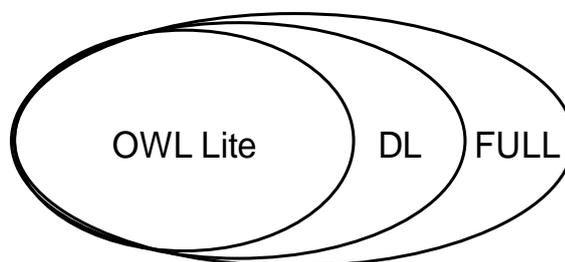
Uno dei motivi principali per cui si usano le DL come supporto per il ragionamento in DAML+OIL è che la *satisfiability* e la *subsumption* possono essere ricavate in modo computazionalmente efficiente, attraverso algoritmi non complessi: essi convertono le asserzioni in una forma normale e poi costruiscono un insieme di vincoli che sono analizzati per verificare se esistono contraddizioni o meno.

### II.5.5 OWL

OWL (Ontology Web Language) è un'estensione di DAML+OIL ottenuta attingendo da esso gli insegnamenti tratti dal suo uso applicativo e, in virtù di ciò, molto simile ad esso. Come DAML+OIL, anche OWL estende RDF ed RDF Schema e usa la sintassi XML. Fornisce tre sottolinguaggi:

- *OWL Full*: consente di combinare OWL con RDF e RDF Schema. E' destinato agli utenti che vogliono la massima espressività e la libertà sintattica di RDF senza alcuna garanzia computazionale. Il suo vantaggio è che è pienamente compatibile con RDF , sia sintatticamente che semanticamente, cioè qualsiasi documento RDF legale è anche un documento OWL Full legale.
- *OWL DL*: inserisce alcuni vincoli sul modo di combinare OWL con RDF Schema. E' destinato agli utenti che vogliono la massima espressività senza perdere l'efficienza e la completezza computazionali, e i benefici dei sistemi che adottano tecniche di reasoning. E' stato realizzato per supportare le Description Logics esistenti.
- *OWL Lite*: è un sottoinsieme di OWL DL che è facilmente utilizzabile ed implementabile. Aggiunge ad RDF Schema molte funzionalità utili per supportare le applicazioni web. Comprende molte delle caratteristiche più usate di OWL ed è destinato agli sviluppatori che vogliono utilizzare OWL

ma vogliono iniziare con un set relativamente semplice di caratteristiche del linguaggio.



**Fig. II.20** Sottolinguaggi di OWL

## III. MODELLO ARCHITETTURALE

### III.1 INTRODUZIONE

Dopo l'analisi effettuata nei primi capitoli, siamo pronti per presentare la nostra idea di modello architetturale per ambienti di *pervasive computing*.

Le principali scelte di progetto sono state prese coerentemente con lo stato dell'arte e si è tentato di non introdurre nuova complessità al panorama esistente, ricorrendo a tecnologie, protocolli e standard già esistenti ed affermati, senza definirne di nuovi. D'ora in avanti il sistema verrà anche chiamato col nome di *UbiSystem*.

### III.2 REQUISITI DEL SISTEMA

L'architettura dovrà essere sufficientemente generale, in modo da poter garantire nuovi sviluppi ed evoluzioni future senza comportare l'esclusione a priori di determinate soluzioni e tecnologie.

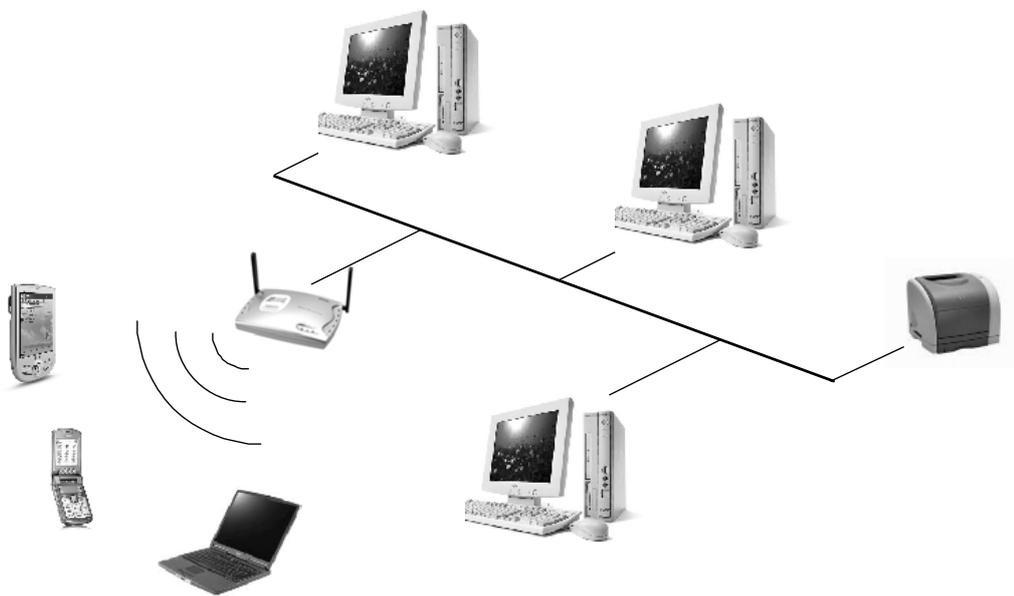
Per quanto visto nei capitoli precedenti, il sistema dovrà garantire i seguenti requisiti funzionali:

- supporto per l'eterogeneità di dispositivi, piattaforme e linguaggi;
- definizione e condivisione dello stato del contesto del sistema;
- discovery semantico dei servizi e supporto per l'integrazione;
- gestione della dinamicità e della disponibilità delle risorse presenti.

Non vanno trascurati, inoltre, tutta una serie di requisiti non funzionali, alcuni dei quali strettamente legati agli aspetti di distributed computing, quali: scalabilità, tolleranza ai guasti, sicurezza ed attendibilità, semplicità, etc.

### III.3 LO SCENARIO E GLI ATTORI

La figura seguente dipinge lo scenario all'interno del quale è stato sviluppato il prototipo. L'ambiente è costituito da una rete LAN con estensioni di tipo wireless: ad una serie di nodi fissi se ne affiancano altri di natura completamente mobile.



**Fig. III.1 Scenario dell'ambiente pervasivo**

Gli attori che interagiscono con *UbiSystem* sono principalmente due:

- Gli utenti (client) - che accedono al sistema mediante dispositivi portatili (palmari, cellulari di ultima generazione, laptop) e che sono interessati ad utilizzare i servizi offerti;
- I fornitori di servizi (service provider) - costituiti da moduli software che offrono servizi e compiono azioni per conto degli utenti.

Dal punto di vista dell'utente è desiderabile poter interagire con il sistema mediante il proprio dispositivo senza dover procedere all'installazione di software aggiuntivo né dover eseguire procedure di configurazione di alcun genere. Per questo motivo l'entry-point degli utenti sarà un'applicazione web e quindi accessibile mediante un comune web browser.

Per quanto riguarda i service provider, è stato stabilito che i servizi offerti all'interno dell'ambiente devono essere preferibilmente dei Web Services.

Questa scelta, adottata coerentemente con la nostra linea progettuale di utilizzare tecnologie e standard web-based, non rappresenta una forte limitazione: di fatti, per riutilizzare moduli già esistenti (pensiamo, ad esempio, a servizi CORBA) è sempre possibile creare appositi wrappers che esponano i servizi come Web Services.

### **III.3.1 PERCHÉ I WEB SERVICES**

La scelta di modellare i servizi all'interno del nostro ambiente come Web Services è stata motivata da una serie di considerazioni che proponiamo di seguito.

Un Web Service costituisce un sistema software disegnato per supportare l'interoperabilità e l'interazione fra macchine attraverso la rete. E' caratterizzato da un'interfaccia descritta mediante formati machine-processable. E' possibile interagire con un servizio Web mediante linguaggi e protocolli standardizzati o in via di standardizzazione quali WSDL, SOAP, HTTP, SMTP, FTP, etc. e quindi secondo tecnologie web-based (cfr. § II.3.3).

Come i Web Services, anche CORBA si propone di garantire interoperabilità multiplatforma e multilinguaggio ma, trattandosi soltanto di specifiche, spesso si riscontrano problemi di incompatibilità fra ORB delle reali implementazioni. Inoltre, la forte limitazione di CORBA è quella di non poter usare Internet come infrastruttura di comunicazione poiché il suo traffico viene sistematicamente bloccato dai firewall presenti sulla rete.

Per quanto riguarda i problemi di Service Discovery e di integrazione di servizi, esistono in letteratura diverse soluzioni. Probabilmente la più famosa è la soluzione Jini proposta dalla SUN (cfr. § II.3.2). Ma come abbiamo visto precedentemente, la tecnologia Jini è strettamente legata alla piattaforma Java e si basa su RMI come middleware.

Tornando ai Web Services, c'è da sottolineare che, accanto alla grande opera di standardizzazione che sta compiendo il World Wide Web Consortium, ingenti sono gli investimenti in ricerca e sviluppo in tal senso da parte delle maggiori industrie e aziende dell'Information Technology, che scommettono su questa nuova ed emergente tecnologia.

Inoltre, insieme a DAML-OIL e OWL (cfr. § II.5.4 e § II.5.5), stanno nascendo ontologie come DAML-S ed OWL-S pensate appositamente per arricchire le descrizioni formali dei Web Services con contenuti di natura semantica.

### III.4 IL MODELLO ARCHITETTURALE

L'ambiente *UbiSystem*, come mostra la figura III.2, offre le seguenti funzionalità:

- ottenere la connessione;
- richiedere la lista dei servizi applicativi presenti nel sistema;
- utilizzare un servizio applicativo presente nel sistema;
- registrare un servizio fornito da un utente;
- gestire i profili degli utenti registrati presso il sistema;
- monitorare l'ambiente.

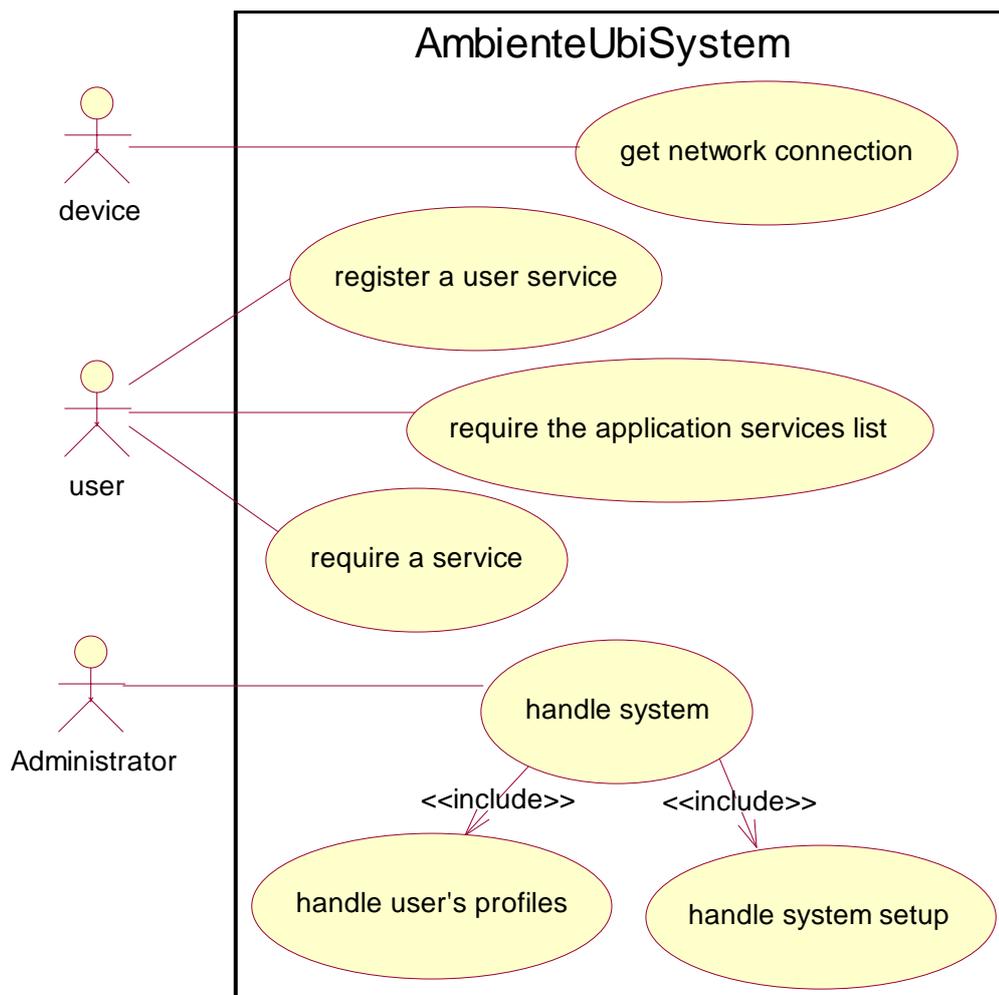


Fig. III.2 Funzionalità offerte dal sistema

Nel definire il framework è stata fatta una distinzione fra i servizi di supporto, finalizzati alla gestione generale del sistema, ed i servizi applicativi di utilità, ovvero i servizi pervasivi.

In figura si mostra l'architettura generale dell'intero sistema.

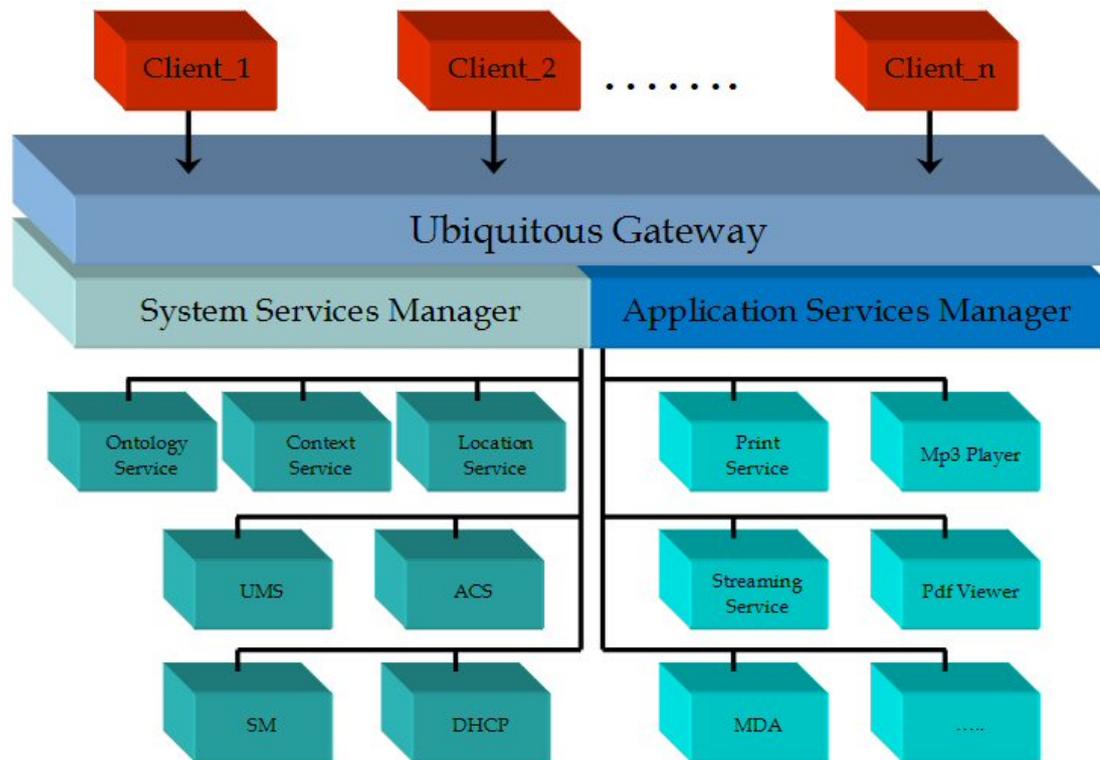


Fig. III.3 Architettura logica del sistema

L'architettura si sviluppa su due piani. Al primo abbiamo:

- *Ubiquitous Gateway*: entry point per gli utenti del sistema pervasivo;
- *System Services Manager*: gestore dei servizi di supporto del sistema;
- *Application Services Manager*: gestore dei servizi pervasivi di utilità.

Al secondo, si trovano i servizi di sistema, che fanno riferimento al *System Service Manager*:

- *DHCP Service*: assegna dinamicamente gli indirizzi della rete e notifica al sistema il momento ed il punto di accesso in cui un dispositivo di utente si aggancia o si sgancia dalla rete, in collaborazione con il *Location Service*;
- *Ontology Service*: gestisce le ontologie dell'ambiente;
- *Asynchronous Communication Service*: gestisce i canali di comunicazione;

- *Context Service*: definisce un modello di contesto e lo condivide con il resto dell'ambiente;
- *Location Service*: estrae informazioni sulla posizione dei dispositivi installati nell'ambiente e del dispositivo utente;
- *Session Manager*: gestisce le sessioni d'utente e d'ambiente;
- *User Manager Service*: svolge servizi per la gestione degli utenti.

Dall'altro lato, invece, si trovano i servizi applicativi, che fanno riferimento all'*Application Service Manager*:

- Servizio di stampa localizzata;
- Servizio di Mp3 JukeBox;
- Servizio di musica d'ambiente;
- Servizio di videoproiezione;
- Servizio di streaming.

L'architettura così disegnata consente di disaccoppiare le funzionalità del sistema ed è facilmente scalabile.

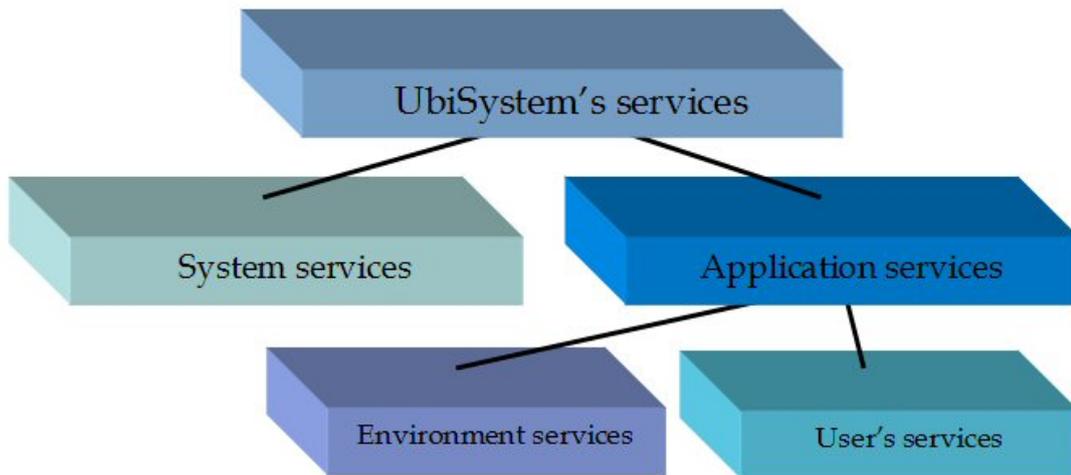
Nella rimanente parte del capitolo verranno approfondite nel dettaglio tutte le funzionalità e le responsabilità di ogni singolo componente.

### **III.5 TASSONOMIA DEI SERVIZI IN UBISYSTEM**

Durante la progettazione dell'ambiente, un'analisi molto approfondita ha permesso di giungere ad una classificazione molto dettagliata dei servizi.

All'interno dei servizi applicativi, a seconda dell'interazione con l'utente, si distinguono:

- *servizi d'utente*: servizi applicativi a cui l'utente può accedere;
- *servizi d'ambiente*: servizi applicativi con quali l'utente non può interagire direttamente (ad esempio, servizio di musica d'ambiente).



**Fig. III.4** Classificazione dei servizi in UbiSystem

Tutti i servizi, inoltre, possono essere soggetti a differenti classificazioni:

- a seconda del tipo di elaborazione fornita si distinguono:
  - *servizi batch*: servizi computazionali;
  - *servizi interattivi*: servizi che richiedono interazione con l'utente.
- a seconda della politica di attivazione applicata si distinguono:
  - *servizi attivi*: servizi che si registrano presso il manager nel momento della loro attivazione, quindi sono su al momento delle richieste provenienti dall'utente;
  - *servizi non attivi*: servizi che vengono attivati automaticamente solo quando richiesti; la registrazione presso il manager viene fatta manualmente dall'amministratore e la descrizione indica il comando da eseguire per portare su il servizio.
- a seconda della dislocazione fisica si distinguono:
  - *servizi centralizzati*: servizi per i quali esiste un unico server;
  - *servizi distribuiti*: servizi per i quali esistono n server su n macchine diverse, in questo caso il manager dei servizi dovrà applicare politiche di load balancing, prima di concedere il servizio.
- a seconda della gestione delle istanze si distinguono:
  - *servizi static*: servizi per i quali esiste una sola istanza utilizzata da tutti i client;
  - *servizi dynamic*: servizi per i quali esiste una istanza per ogni client istanziata dinamicamente da una factory.

### III.6 SYSTEM SERVICES MANAGER

E' il componente presso il quale vengono registrati i servizi di sistema, ovvero tutti quei servizi che sono di supporto per l'ambiente pervasivo. Tali servizi possono essere acceduti da parte dell'*Ubiquitous Gateway* e dell'*Application Services Manager* soltanto attraverso questo componente (ad esempio, immaginiamo un servizio di stampa che prima di avviare il processo necessita dell'autenticazione dell'utente che ne faccia la richiesta). La scelta di introdurre il *System Services Manager* e di distinguerlo dal gestore dei servizi applicativi è stata dettata dall'esigenza di avere a disposizione un meccanismo semplice per aggiungere nuovi servizi di supporto per l'ambiente (ad esempio servizi di policy e sicurezza, o magari di QoS) e, allo stesso tempo, di mantenere distinte le due tipologie di servizi sia per ragioni pratiche che amministrative.

Le funzionalità e le responsabilità del *System Services Manager* sono:

- gestire la pubblicazione dei servizi di sistema presenti nell'ambiente all'atto della loro attivazione;
- mantenere e rendere disponibili le descrizioni di tutti i servizi di sistema attivi;
- offrire meccanismi per il discovery dei servizi di sistema disponibili.

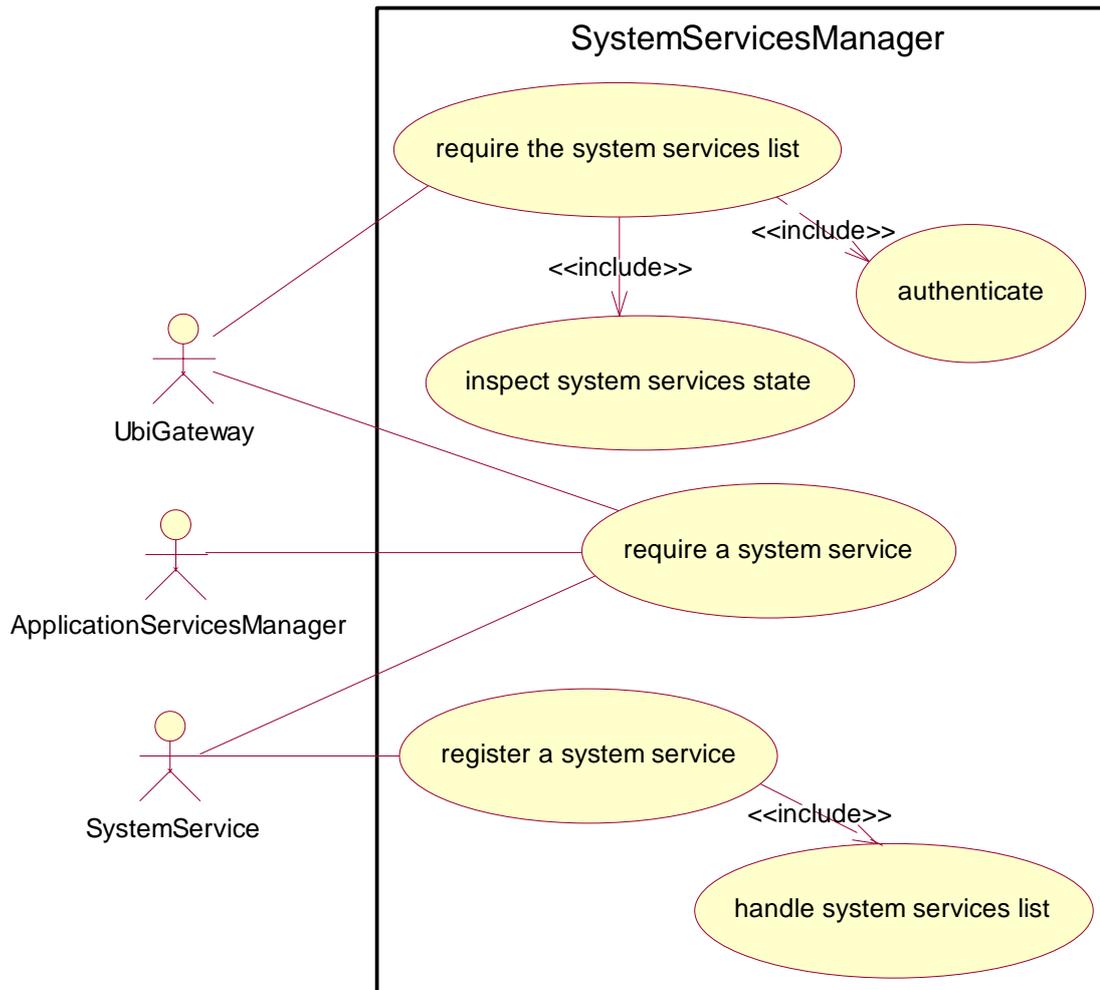


Fig. III.5 Funzionalità offerte dal System Services Manager

### III.6.1 SYSTEM SERVICES

I tipi di servizi necessari per coordinare e gestire l'intero sistema pervasivo possono essere molteplici.

Ne sono stati individuati alcuni che riportiamo di seguito e che si propongono di soddisfare i requisiti di un sistema pervasivo.

#### III.6.1.1 DHCP SERVICE

Le funzioni del *DHCP Service* vanno al di là della semplice assegnazione automatica degli indirizzi ai nuovi elementi che entrano a far parte della rete. Infatti, esso ha la peculiarità aggiuntiva di notificare al sistema il momento e il punto di accesso in cui un dispositivo si aggancia o si sgancia dalla rete.

Questa caratteristica è molto importante per garantire la proprietà di location-awareness e context-awareness all'interno dell'ambiente pervasivo.

### III.6.1.2 ONTOLOGY SERVICE

L'*Ontology Service* è il componente il cui compito è quello di gestire le ontologie utilizzate all'interno dell'ambiente.

Una Ontologia è la descrizione formale di concetti appartenenti ad un determinato dominio applicativo.

L'adozione di modelli ontologici all'interno di sistemi di pervasive computing ha fondamentalmente lo scopo di rendere possibile l'interazione e la collaborazione tra le differenti parti che compongono il sistema.

In particolare, attraverso l'uso di ontologie è possibile:

- definire i termini usati all'interno dell'ambiente, per stabilirne in maniera univoca sintassi e semantica;
- verificare le descrizioni di ciascuna entità affinché rispettino gli schemi definiti;
- intraprendere discovery semantico, per rendere più efficiente la ricerca di oggetti cercando di scoprire tutte e sole le entità rilevanti;
- garantire interoperabilità fra i componenti, fissando definizioni formali ed universali che in particolare descrivono i singoli servizi offerti da ogni risorsa.

Nel progetto, vengono definite ontologie per descrivere i vari aspetti dell'ambiente. Attraverso di esse, viene fissato uno schema logico ed una tassonomia standard interpretabile, oltre che dall'uomo, anche da software automatici.

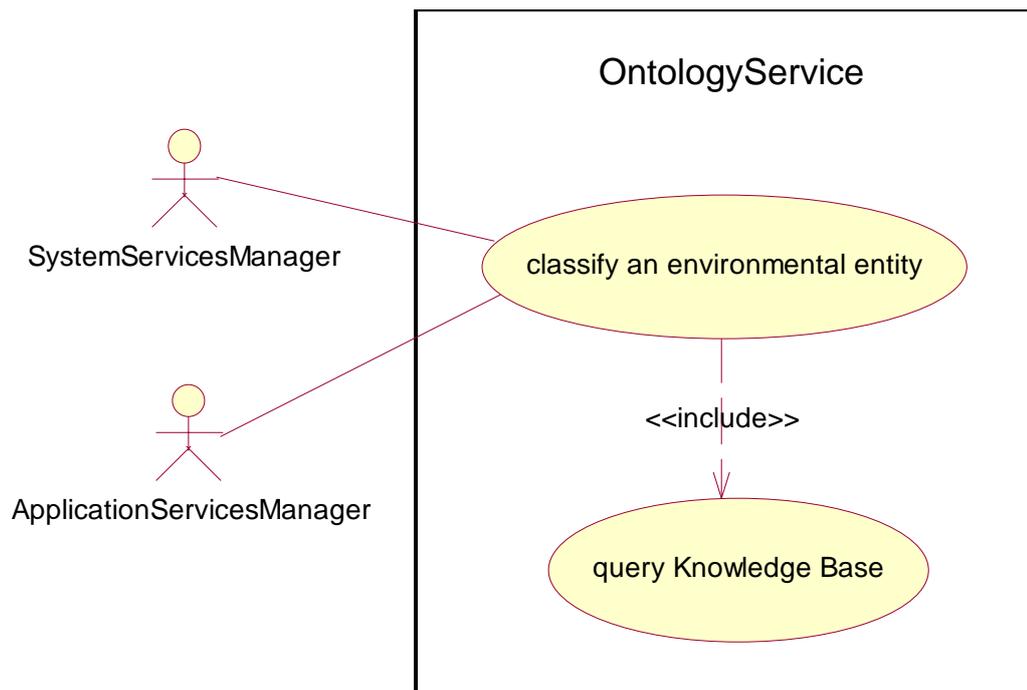
In particolare, sono state sviluppate ontologie di:

- **Contesto:** stabilisce concetti inerenti lo stato dell'ambiente: persone presenti, luminosità, temperature, attività in esecuzione, etc.;
- **Servizio:** stabilisce concetti e caratteristiche che le descrizioni dei servizi disponibili dell'ambiente devono rispettare;
- **Device:** stabilisce un modello per la caratterizzazione e classificazione dei dispositivi presenti nell'ambiente, comprese peculiarità hardware e software;
- **Utente:** stabilisce lo schema concettuale caratterizzante gli utenti degli ambienti pervasivi.

Tutte le ontologie sono sviluppate in linguaggio DAML+OIL (cfr. § II.5.4).

L'*Ontology Service* mette a disposizione un insieme di API (Application Programming Interface) per caricare le ontologie e costruire una Knowledge Base, verificare le descrizioni delle entità, interrogare la Knowledge Base e ricercare definizioni di concetti e relazioni.

La figura riportata di seguito dipinge le funzionalità offerte da questo componente.

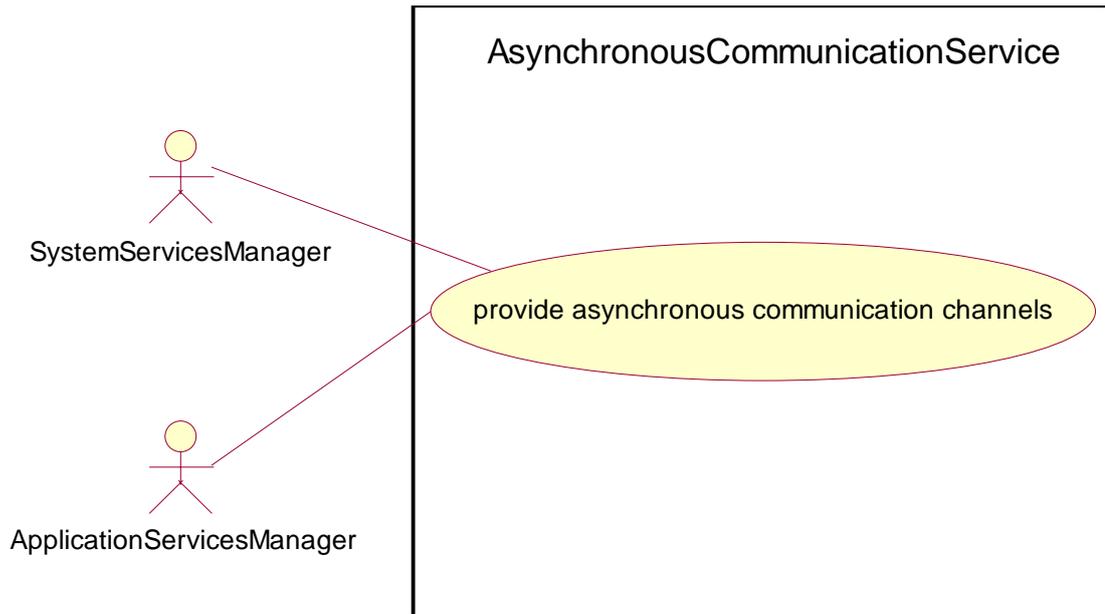


**Fig. III.6** Funzionalità offerte dall'*Ontology Service*

### III.6.1.3 ASYNCHRONOUS COMMUNICATION SERVICE

In ambienti pervasivi non è sempre sufficiente disporre solo di meccanismi di comunicazione basati su paradigmi sincroni. Infatti, la mobilità degli utenti, la dinamicità dei servizi, i problemi legati alla momentanea perdita di connessione inducono a prendere in considerazione anche modelli di comunicazione asincrona.

L'*Asynchronous Communication Service* offre il supporto per questo tipo di comunicazione fornendo canali ad eventi.

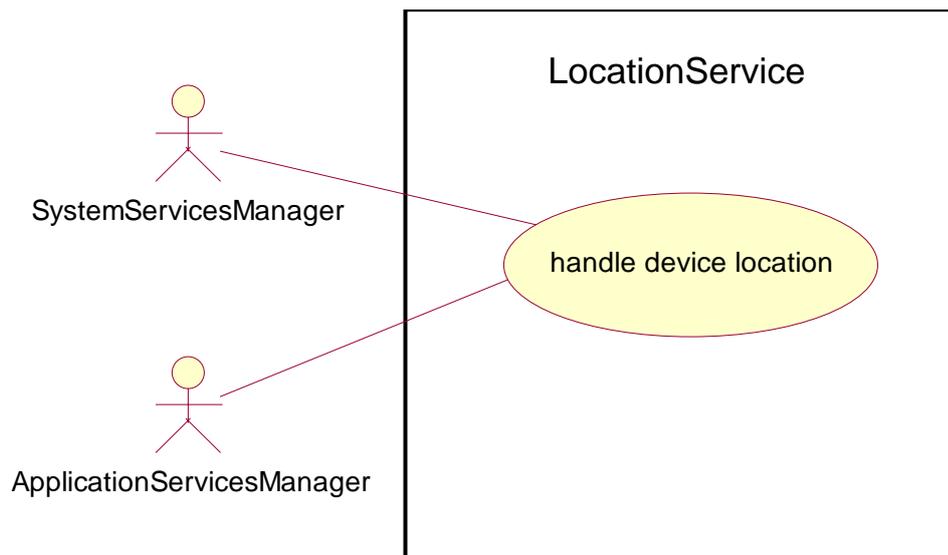


**Fig. III.7 Funzionalità offerte dall'Asynchronous Communication Service**

#### III.6.1.4 LOCATION SERVICE

Il *Location Service* si occupa della localizzazione degli utenti del sistema e della localizzazione dei dispositivi mobili all'interno dell'ambiente pervasivo realizzato. Gli altri servizi di sistema possono, ad esempio, interrogare il *Location Service* sulla locazione di un utente richiedendo l'edificio, la stanza o la locazione interna alla stanza in cui quest'ultimo si trova: il *Location Service* è in grado di operare a diversi livelli di dettaglio nell'estrazione delle informazioni di localizzazione.

Il *Location Service* garantisce la proprietà di location-awareness. Esso si occupa, infatti, anche della diffusione di eventi, generati quando un utente o dispositivo mobile cambia locazione all'interno del sistema pervasivo. In questo modo le altre entità di sistema possono essere avvertite di eventuali cambiamenti di posizione di utenti o di dispositivi mobili.



**Fig. III.8** Funzionalità offerte dal Location Service

#### III.6.1.5 CONTEXT SERVICE

Il contesto e la sensibilità al contesto hanno assunto un ruolo di fondamentale importanza negli ambienti pervasivi. Per “contesto” si intende qualsiasi informazione che può essere usata per caratterizzare il comportamento di una persona o di entità software.

Il *Context Service* ha il compito di costruire e condividere con gli agenti del sistema le informazioni di contesto dell’ambiente.

E’ possibile immaginare diverse tipologie di contesto che possono essere usate dalle applicazioni:

- contesti fisici (posizione e tempo);
- contesti ambientali (aspetti meteorologici, livelli di luce e sonoro);
- contesti informativi (quote azionarie, punteggi sportivi);
- contesti applicativi (e-mail, siti web visitati);
- contesti di sistema (traffico della rete, stato delle stampanti);
- contesti personali (salute, attività, stati d’animo);
- contesti sociali (attività di gruppo, relazioni sociali, con chi si è in una stanza).

Per essere context-aware, le applicazioni necessitano di un’infrastruttura composta di sensori che percepiscono le informazioni dall’ambiente e di entità software che inferiscono nuove relazioni a partire dai dati percepiti.

Si immagini, ad esempio, il seguente scenario: un sensore segnala l'ingresso di una persona nell'ambiente; nel momento in cui la persona entra, è in atto una presentazione PDF; se in un'ontologia è stato definito che durante una presentazione tutte le persone presenti sono dei "partecipanti alla presentazione", si potrebbe concludere che la persona appena entrata è un nuovo partecipante alla presentazione.

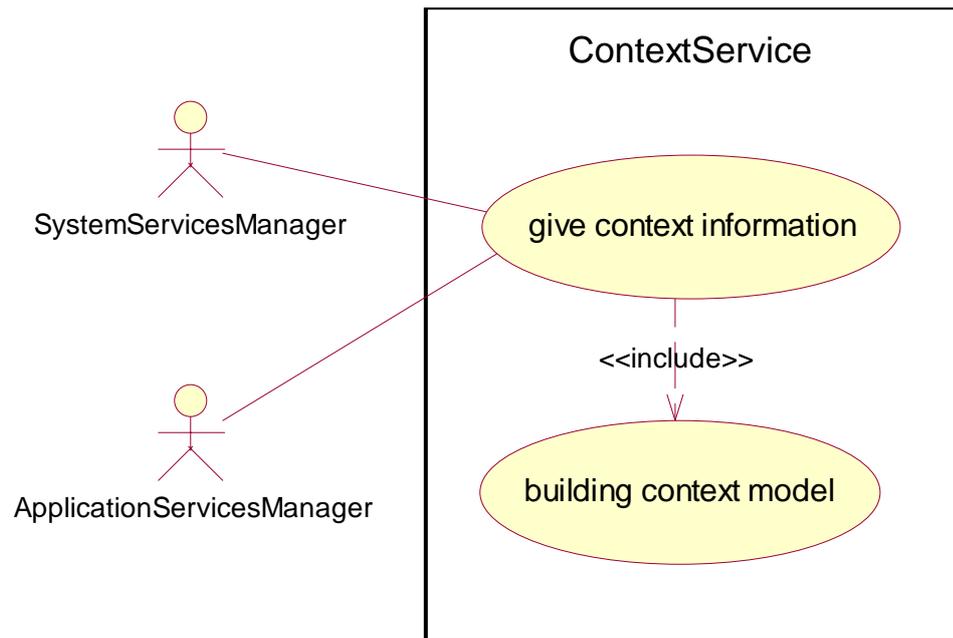
Il *Context Service* definisce il contesto dell'ambiente, sia a partire dai dati raccolti da appositi sensori, sia da quelli provenienti dagli altri componenti del sistema, alcuni dei quali, ad esempio, sono l'*Ubiquitous Gateway*, il *Session Manager* ed il *Location Service*.

In questo modo, il *Context Service* è informato sul numero e sull'identità degli utenti correntemente presenti all'interno dell'ambiente, dei servizi disponibili, dello stato delle applicazioni.

Le applicazioni ed i servizi dell'ambiente, a partire dalle informazioni di contesto ricevute dal *Context Service*, possono adattare il proprio comportamento secondo le diverse situazioni. Per esempio, un'applicazione di gestione dell'illuminazione di un ambiente potrebbe automaticamente accendere le luci regolandone l'intensità secondo le preferenze delle persone presenti all'interno dell'ambiente; oppure, potrebbe regolare l'intensità dell'illuminazione in dipendenza dall'ora del giorno.

Anche in questo caso, l'impiego di ontologie rende più semplice per gli sviluppatori specificare il comportamento dei propri applicativi secondo le diverse esigenze.

La figura riportata di seguito dipinge le funzionalità offerte da questo componente.



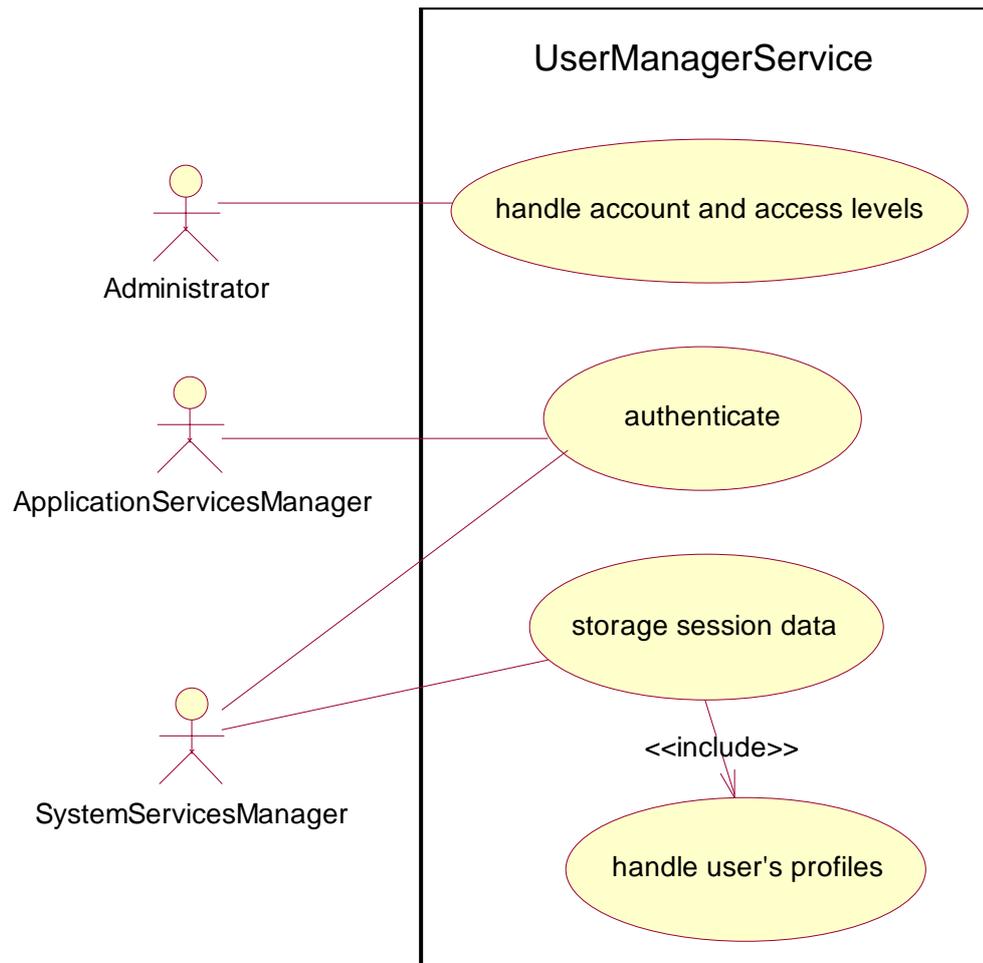
**Fig. III.9** Funzionalità offerte dal Context Service

#### III.6.1.6 USER MANAGER SERVICE

L'*User Manager Service* ha il compito di svolgere tutte le attività legate ad una corretta gestione degli utenti che interagiscono con il sistema. In particolare il servizio deve:

- permettere la gestione dei diritti di accesso ai servizi offerti;
- provvedere all'autenticazione di ogni utente;
- gestire una cronologia delle attività svolte dagli utenti registrati;
- costruire e condividere con gli agenti di sistema le informazioni relative al profilo di ciascun utente registrato;
- gestire il mapping tra l'utente ed i vari dispositivi attraverso i quali l'utente si aggancia alla rete.

La figura seguente dipinge le funzionalità offerte da questo componente.



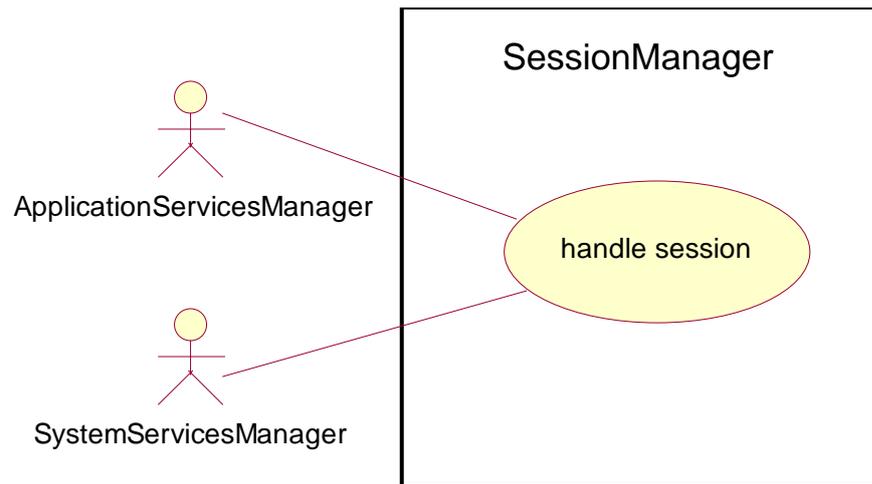
**Fig. III.10** Funzionalità offerte dall'User Manager Service

### III.6.1.7 SESSION MANAGER

Il *Session Manager* rappresenta il componente che si occupa della gestione delle sessioni.

In particolare il servizio deve:

- gestire una sessione per i servizi applicativi d'ambiente attivi nel sistema;
- gestire una sessione di lavoro per ciascun utente presente nell'ambiente;
- verificare la validità delle sessioni d'utente.



**Fig. III.11** Funzionalità offerte dal Session Manager

### III.7 APPLICATION SERVICES MANAGER

Rappresenta il componente referente per i *service providers* che intendono pubblicare un nuovo servizio.

Le funzionalità e le responsabilità dell'*Application Services Manager* sono:

- gestire la pubblicazione dei servizi offerti dall'utente;
- gestire la pubblicazione dei servizi applicativi presenti nell'ambiente all'atto della loro attivazione;
- mantenere e rendere disponibili le descrizioni di tutti i servizi applicativi attivi nel sistema;
- offrire meccanismi per il discovery dei servizi disponibili;
- applicare politiche di bilanciamento del carico prima di consentire l'uso di un servizio.

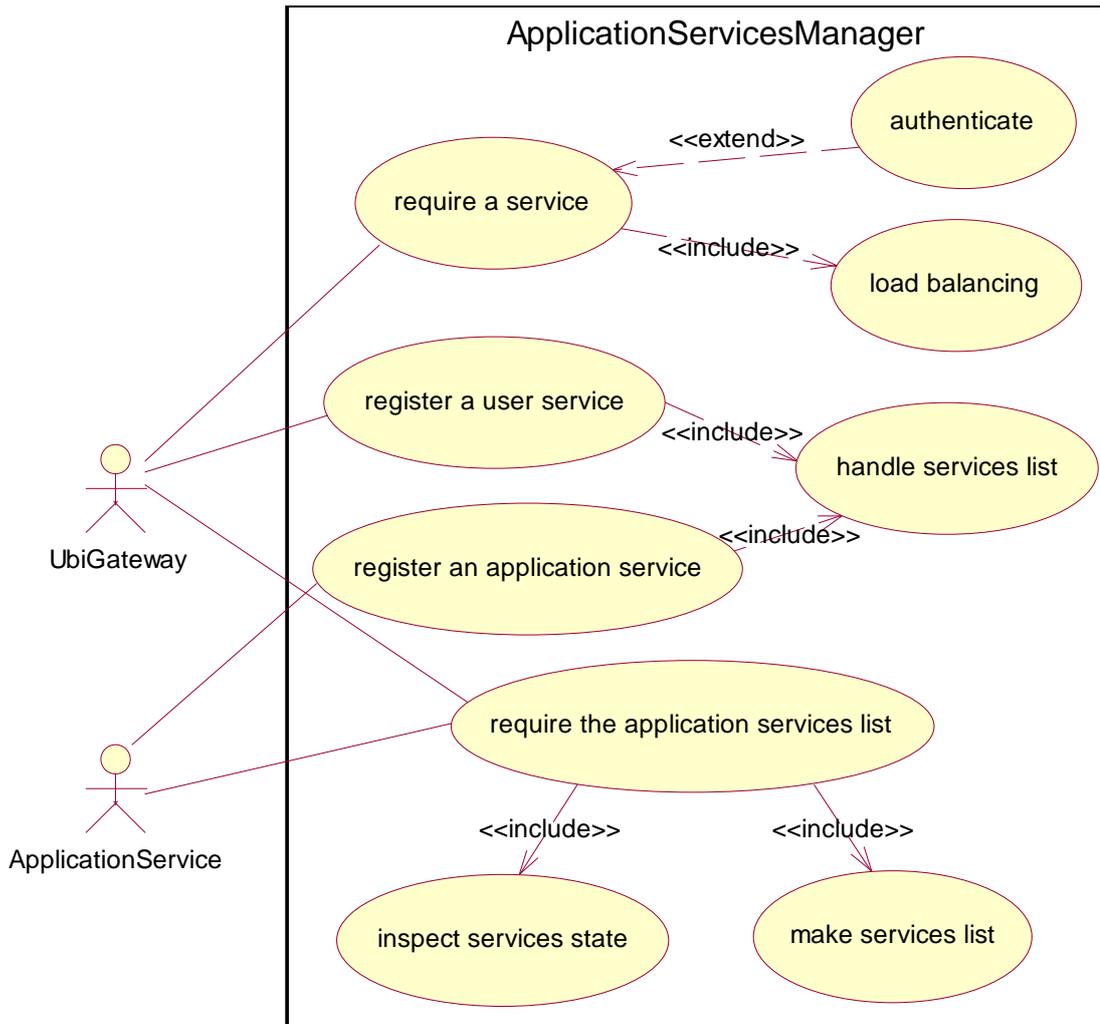


Fig. III.12 Funzionalità offerte dall'Application Services Manager

### III.7.1 APPLICATION SERVICES

Un utente che entra in *UbiSystem* può usufruire di diversi servizi applicativi.

#### III.7.1.1 SERVIZIO DI STAMPA LOCALIZZATA

Il servizio in esame consente agli utenti di scegliere i documenti da stampare e settare alcune preferenze di stampa, in particolare stampa a colori o monocromatica.

Esso avvia in automatico la stampa presso la stampante nella stanza dell'utente oppure chiede all'utente di scegliere esplicitamente la stanza ove gli sia più comodo prelevare i documenti.

Si tratta di un servizio applicativo d'utente e interattivo.

### III.7.1.2 SERVIZIO DI MP3 JUKEBOX

Il servizio consente agli utenti di creare ed ascoltare playlist di brani musicali. Quest'ultima può includere sia brani che si trovano sul server che brani proposti dall'utente stesso.

Si tratta di un servizio applicativo d'utente e interattivo.

### III.7.1.3 SERVIZIO DI MUSICA D'AMBIENTE

Il servizio provvede alla gestione della musica di sottofondo di una determinata stanza a tema in dipendenza della presenza degli utenti.

Il servizio provvede alla gestione della musica di sottofondo di una determinata stanza a tema.

Si tratta di un servizio applicativo d'ambiente e non espone funzionalità accessibili direttamente dagli utenti.

### III.7.1.4 SERVIZIO DI VIDEOPROIEZIONE

Il PdfViewer è un servizio che permette ad un utente di tenere una presentazione per una conferenza. Il relatore attraverso, il suo palmare, una volta avuto accesso al servizio, può inviare il file pdf da visualizzare e comandare la presentazione attraverso il proprio dispositivo.

Si tratta di un servizio applicativo d'utente e interattivo.

### III.7.1.5 SERVIZIO DI STREAMING

Il servizio si occupa di far visualizzare all'utente, sul proprio dispositivo, un filmato diverso a seconda della sua locazione.

Si tratta di un servizio applicativo d'utente e interattivo.

## III.8 UBIQUITOUS GATEWAY

L'*Ubiquitous Gateway* rappresenta l'entry point per gli utenti del sistema verso l'ambiente pervasivo.

Da un punto di vista funzionale, l'*Ubiquitous Gateway* dovrà:

- Registrare e autenticare gli utenti che effettuano l'accesso;

- Reperire la descrizione del dispositivo mediante il quale l'utente interagisce con l'ambiente;
- Offrire meccanismi per ricercare ed utilizzare le risorse disponibili nell'ambiente.

La figura 3.11 dipinge le funzionalità offerte da questo componente.

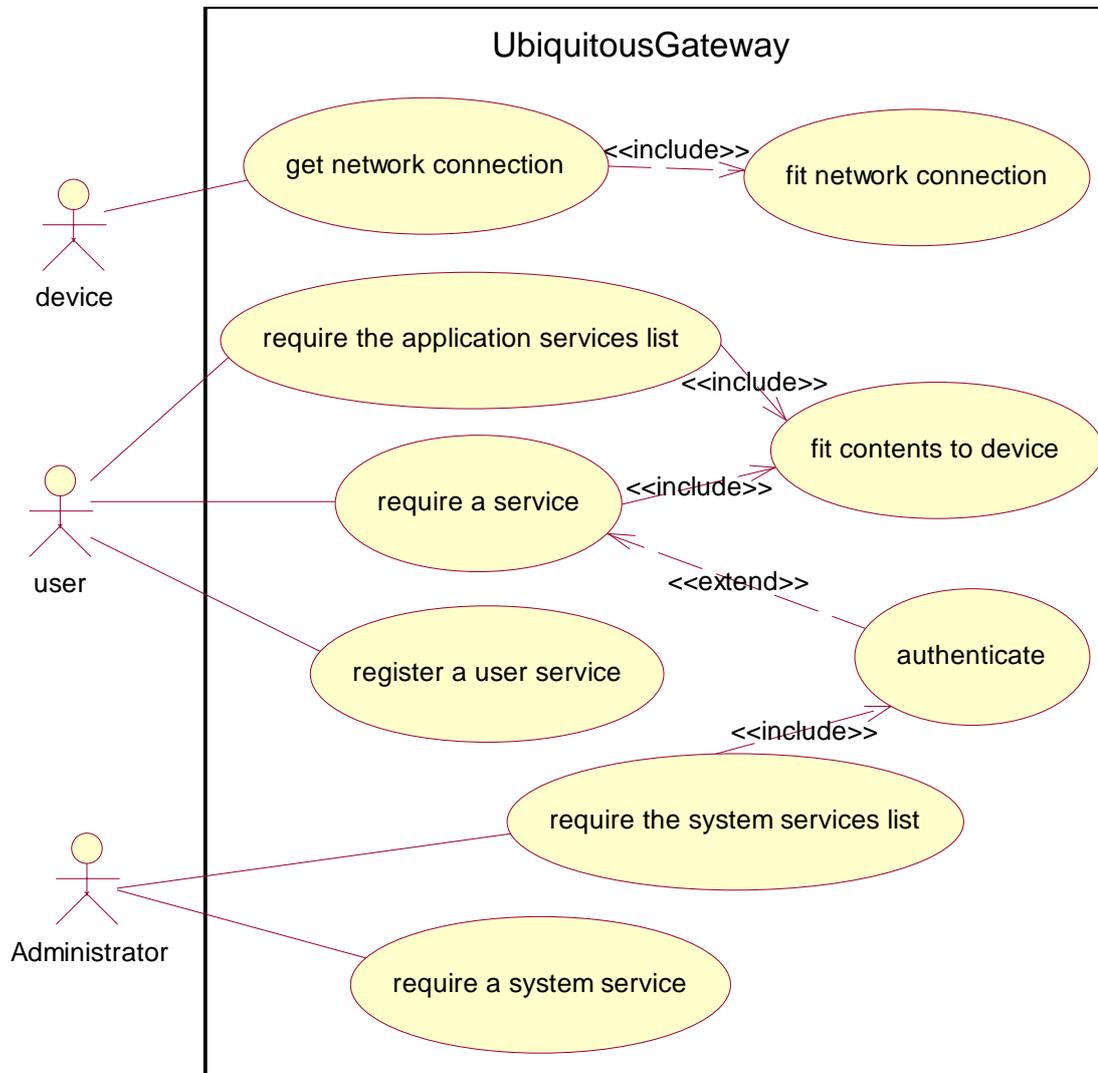


Fig. III.13 Funzionalità offerte dall'Ubiquitous Gateway

## IV. COMPONENTI IMPLEMENTATI

### IV.1 INTRODUZIONE

*UbiSystem* è un'infrastruttura Web-based che mira a gestire e rendere possibile la comunicazione tra entità di un ambiente pervasivo, cercando di garantire tutta una serie di requisiti, quali:

- Supporto per l'eterogeneità di dispositivi, piattaforme e linguaggi;
- Definizione e condivisione dello stato del contesto dell'ambiente;
- Discovery semantico dei servizi e supporto per l'integrazione;
- Gestione della dinamicità e della disponibilità delle risorse presenti.

Esso si preoccupa, fondamentalmente, di consentire ad un utente, che accede al sistema mediante un qualsiasi dispositivo portatile (palmare, cellulare, laptop), di poter usufruire, in seguito ad una fase di autenticazione, di una serie di servizi e risorse disponibili nell'ambiente senza dover procedere all'installazione di alcun software aggiuntivo né di dover eseguire procedure di configurazione di alcuna sorta, ma adoperando un comune internet browser.

I componenti del sistema sono realizzati come servizi CORBA e solo se necessario, ovvero quando espongono funzionalità che devono essere rese accessibili dall'esterno, sono esposti come Web Services.

La scelta di CORBA è dovuta al fatto che si tratta di una tecnologia sicura ed efficiente, ed inoltre anche implementazioni gratuite di piattaforme CORBA offrono servizi, ad esempio l'Event ed il Notification Service, in grado di offrire meccanismi

per la realizzazione di tipologie di comunicazione, tra oggetti, diversa da quella sincrona.

Per la realizzazione dei componenti che saranno descritti nel prosieguo del capitolo si è scelto di utilizzare, come implementazione della infrastruttura CORBA, JacORB.

JacORB è un Java object request broker open source, molto popolare (usato da enti di ricerca e in campo aziendale) ed efficiente che, progettato nel 1995 in un dipartimento della Freie Universität Berlin (FUB), continua ad essere usato per ricerche alla FUB, specialmente nel campo della sicurezza di oggetti distribuiti.

La versione utilizzata è JacORB 2.2, comprensiva di alcuni servizi tra i quali il Name Service ed il Notification Service.

I componenti implementati nell'ambito della presente tesi di laurea sono:

- *Ubiquitous Gateway;*
- *Application Services Manager;*
- *System Service Manager.*

Nel corso del presente capitolo saranno esposti tutti i passi della progettazione fino a giungere ai dettagli implementativi.

## **IV.2 BUSINESS CLASS DIAGRAMS**

Lo scopo di questo tipo di diagramma è l'individuazione dei metodi e degli attributi di cui deve essere dotato ogni componente. A partire dagli Use Case Diagrams si cerca di tradurre le funzionalità esterne individuate in modo da ottenere i metodi necessari all'interazione dei diversi moduli del sistema.

### **IV.2.1 UBIQUITOUS GATEWAY**

L'*Ubiquitous Gateway* rappresenta l'entry point per gli utenti del sistema e per il loro dispositivo verso l'ambiente pervasivo.

Le funzionalità che questo componente deve offrire sono:

- *Get network connection.* Deve permettere, a livello logico, la connessione fisica del dispositivo utente.

- *register a user device.* Quando un utente entra nel sistema lo fa accedendo alla home page di UbiSystem. In questo stesso momento *Ubiquitous Gateway* deve provvedere alla registrazione del dispositivo. Per fare ciò è necessario prelevare le caratteristiche del dispositivo e modellarle in un file scritto in linguaggio DAML+OIL. Tale file dovrà essere inviato all'*Ontology Service* per essere validato e registrato. Il metodo che realizzerà la funzionalità esposta potrebbe essere: *set\_characteristics()*.
- *require the application services list.* Il componente in esame offre all'utente la possibilità di visualizzare la lista dei servizi disponibili nell'ambiente. Se tali servizi sono disponibili l'utente potrà utilizzarli, altrimenti viene indicato il motivo che ne impedisce l'utilizzo. Egli avrà anche la possibilità di autenticarsi per ampliare il numero di servizi che può utilizzare. Il metodo che realizzerà la funzionalità esposta potrebbe essere: *get\_all\_services()*, ma se l'utente si autentica potrebbe invocare direttamente il metodo: *authenticate()*.
- *require a service.* Se il servizio risulta disponibile, il metodo che ne permette l'utilizzo potrebbe essere: *require\_application\_service()*.
- *register a user service.* Tale funzionalità permette ad un utente di poter mettere a disposizione, all'interno dell'ambiente pervasivo, un proprio servizio. Il metodo, allora potrebbe essere: *offer\_service()*.
- *require the system services list.* L'amministratore deve avere la possibilità di visualizzare la lista dei servizi di sistema per poi poterne sfruttare le funzionalità. Per fare questo, però, è necessario che egli si autentichi. Il metodo necessario potrebbe essere: *authenticate()*.
- *require a system service.* Analogamente a quanto visto per i servizi applicativi, *Ubiquitous Gateway* dovrà esporre un metodo per l'utilizzo dei servizi di sistema. Questo potrebbe essere: *require\_system\_service()*.

Oltre alle funzionalità esposte, nel prosieguo potrebbero emergere altre. Ad esempio nasce spontaneo pensare di offrire all'utente un modo per comunicare la sua uscita dal sistema, quindi già da ora conviene inserire un metodo apposito che potrebbe essere *logout()*.

Quanto esposto può essere schematizzato, con l'aiuto di Power Design, come segue:

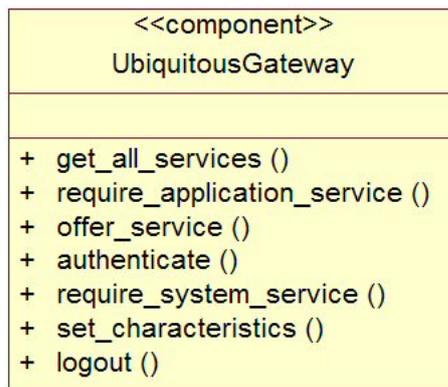


Fig. IV.1 Componente Ubiquitous Gateway

## IV.2.2 APPLICATION SERVICES MANAGER

L'*Application Services Manager* rappresenta il componente referente per i *service providers* che intendono pubblicare un nuovo servizio applicativo. Inoltre, qualsiasi entità che voglia usufruire dei servizi applicativi presenti nell'ambiente può farlo solamente attraverso questo componente.

Le funzionalità che questo componente deve offrire sono:

- *register an application service*. Quando un servizio applicativo si attiva nell'ambiente pervasivo deve notificare la sua presenza. Questo componente offre questa possibilità esponendo un metodo opportuno che potrebbe chiamarsi: *register\_application\_service()*. Dopo aver eseguito questa operazione il sistema è consapevole del nuovo servizio.
- *register a user service*. Tale funzionalità sarà sfruttata dall'*Ubiquitous Gateway* quando un utente vuole pubblicare un proprio servizio nell'ambiente. Il metodo potrebbe chiamarsi: *register\_user\_service()*.

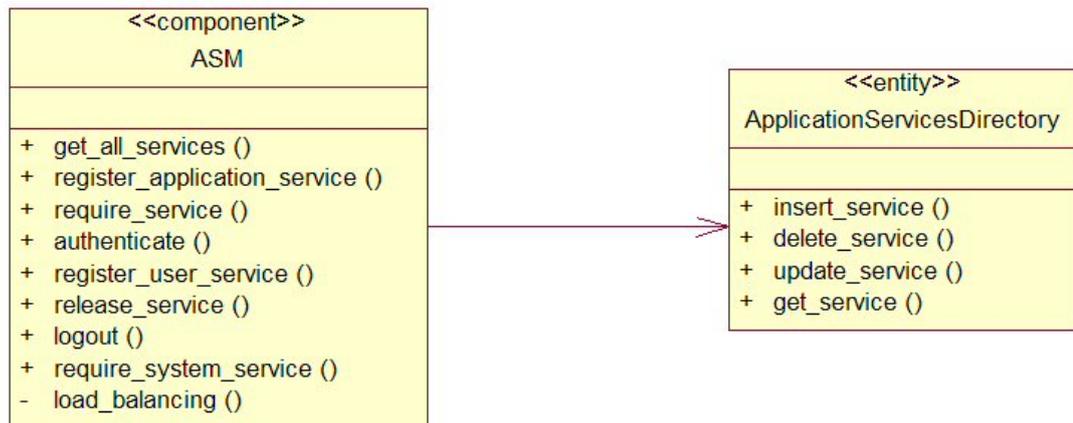
- *require the application services list.* Tale funzionalità sarà sfruttata dall'*Ubiquitous Gateway* nel momento in cui deve fornire all'utente la lista dei servizi. Il metodo necessario potrebbe chiamarsi, analogamente al componente prima analizzato, *get\_all\_services()*. Se l'utente si autentica, la lista sarà fornita dal metodo *authenticate()*.
- *require a service.* Una volta ottenuta la lista dei servizi disponibili l'utente ne richiede l'utilizzo mediante il metodo: *require\_service()*. Tale metodo risulta utile anche nel caso in cui un servizio richieda un altro servizio applicativo per fornire una funzionalità più complessa all'utente. Un servizio potrebbe essere distribuito su più macchine, e quindi avere più istanze, in questo caso è compito dell'*Application Services Manager* applicare delle politiche di bilanciamento del carico prima di concedere il servizio, quindi, risulta necessario prevedere un metodo privato apposito: *load balancing()*.

L'*Application Services Manager* ha anche il compito di mantenere lo stato dei servizi applicativi, per questo deve permettere il rilascio del servizio. In questo modo avrà la possibilità di deallocare le risorse in uso dal servizio stesso. Dunque risulta utile prevedere il metodo: *release\_service()*. Inoltre, potrebbe risultare utile sapere quando un utente abbandona il sistema e per questo prevedere il metodo: *logout()*.

Durante il funzionamento di un servizio applicativo potrebbe presentarsi la necessità di dover sfruttare le funzionalità di un servizio di sistema, quindi dovrà essere l'*Application Services Manager* a permetterne il reperimento. Per questo si è pensato di introdurre il metodo: *require\_system\_service()*.

Per memorizzare i servizi attivi e tutte le informazioni relative si utilizza una struttura denominata *ApplicationServicesDirectory* che permette di inserire, rimuovere, aggiornare e prelevare i servizi in essa contenuti.

Quanto esposto può essere schematizzato, con l'aiuto di Power Design, come in figura IV.2 a pagina seguente.



**Fig. IV.2** Componente Application Services Manager

### IV.2.3 SYSTEM SERVICES MANAGER

Il *System Services Manager* rappresenta il componente referente dove si registrano tutti i servizi si sistema, ovvero quei servizi che sono di supporto per l'ambiente pervasivo. Inoltre, qualsiasi entità che voglia usufruire dei servizi di sistema presenti nell'ambiente può farlo solamente attraverso questo componente.

Le funzionalità che questo componente deve offrire sono:

- *register a system service.* Quando un servizio di sistema si attiva nell'ambiente pervasivo deve notificare la sua presenza. Questo componente offre questa possibilità esponendo un metodo opportuno che potrebbe chiamarsi: `register_system_service()`. Dopo aver eseguito questa operazione il sistema è consapevole del nuovo servizio.
- *require the system services list.* Tale funzionalità sarà sfruttata dall'*Ubiquitous Gateway* nel momento in cui deve fornire la lista dei servizi di sistema. Per ottenere tale lista è necessario che l'amministratore si autentichi, quindi il nome del metodo potrebbe essere: `authenticate()`.
- *require a system service.* Un servizio di sistema può essere richiesto sia dall'amministratore che da un'altra entità nell'ambiente, quindi il metodo necessario sarà: `require_system_service()`. Anche nel caso dei servizi di sistema, un servizio potrebbe essere distribuito su più macchine, e quindi avere più istanze, in questo caso è compito del *System Services Manager*

applicare delle politiche di bilanciamento del carico prima di concedere il servizio, quindi, risulta necessario prevedere un metodo privato apposito: *load balancing()*.

Anche in questo caso risulta utile sapere quando l'amministratore lascia il sistema e per questo è necessario prevedere il metodo: *logout()*.

Per memorizzare i servizi attivi e tutte le informazioni relative si utilizza una struttura denominata *SystemServicesDirectory* che permette di inserire, rimuovere, aggiornare e prelevare i servizi in essa contenuti.

Quanto esposto può essere schematizzato, con l'aiuto di Power Design, come segue:

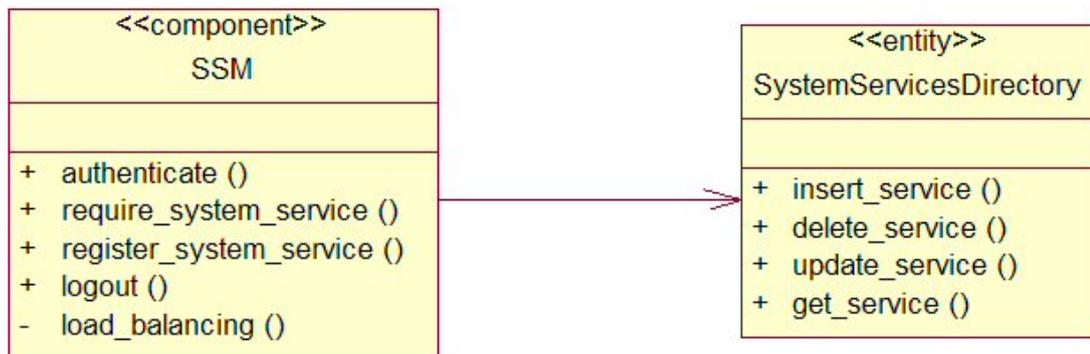


Fig. IV.3 Componente System Services Manager

### IV.3 SEQUENCE DIAGRAMS

Un *Sequence Diagram* rappresenta un'interazione visualizzata rispetto a una sequenza temporale necessaria a portare a termine una procedura. Esso mostra gli oggetti che partecipano all'interazione in termini del loro tempo di vita e dei messaggi che essi si scambiano.

Lo scopo di questo paragrafo è quello di mostrare le interazioni tra i vari moduli del sistema per portare a termine le principali funzionalità che l'intero sistema *UbiSystem* deve offrire. Al termine di questa analisi saranno individuati ulteriori metodi che i vari componenti dovranno implementare per soddisfare i requisiti del sistema.

### IV.3.1 REGISTRAZIONE DI UN SERVIZIO DI SISTEMA

Affinché un servizio di sistema possa registrarsi è necessario che il *System Services Manager* sia attivo e abbia reso noto il proprio riferimento.

Il primo servizio a doversi registrare è l'*Ontology Service*, senza il quale sarebbe impossibile registrare gli altri servizi. In figura si riporta il sequence relativo:

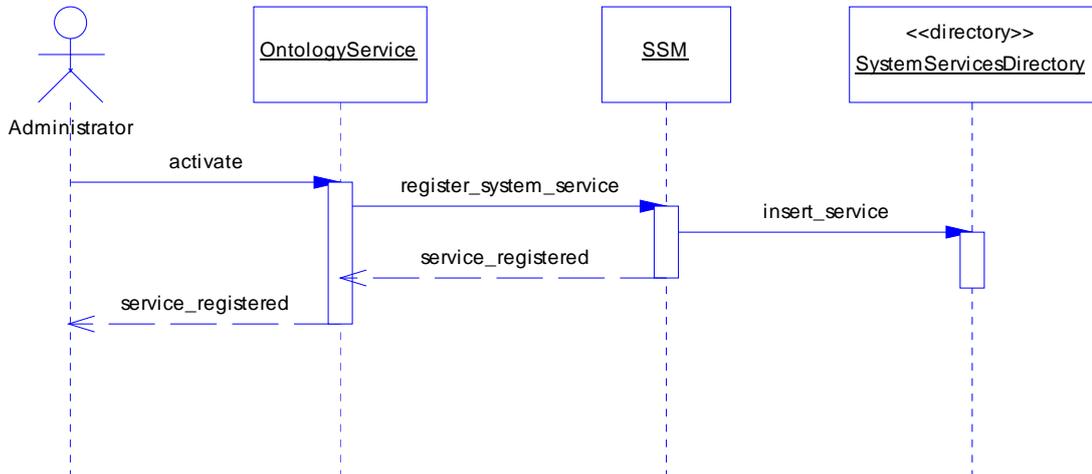
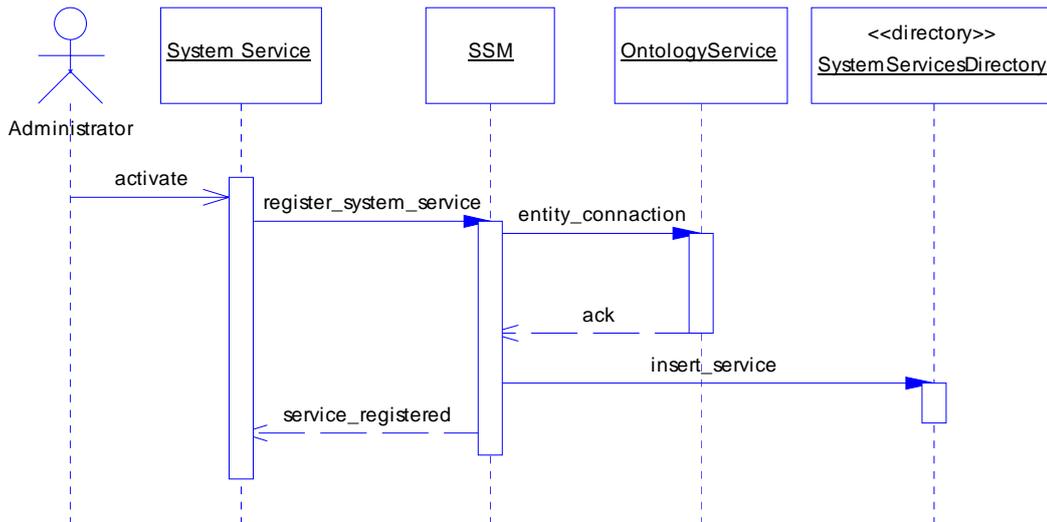


Fig. IV.4 Registrazione dell'Ontology Service.

Quando l'amministratore attiva il servizio questo provvede a registrarsi presso l'*SSM*. Per eseguire la registrazione, il manager avrà bisogno di conoscere almeno il reference dell'*Ontology Service* e la sua URL, nonché il nome simbolico con il quale sarà reperibile nell'ambiente.

Per gli altri servizi di sistema la procedura è leggermente diversa, infatti la registrazione coinvolge anche l'*Ontology Service*. Infatti esso ha il compito di valicare la descrizione del servizio, cioè deve verificare che essa sia coerente con il resto del sistema. Di seguito si riporta il sequence relativo.



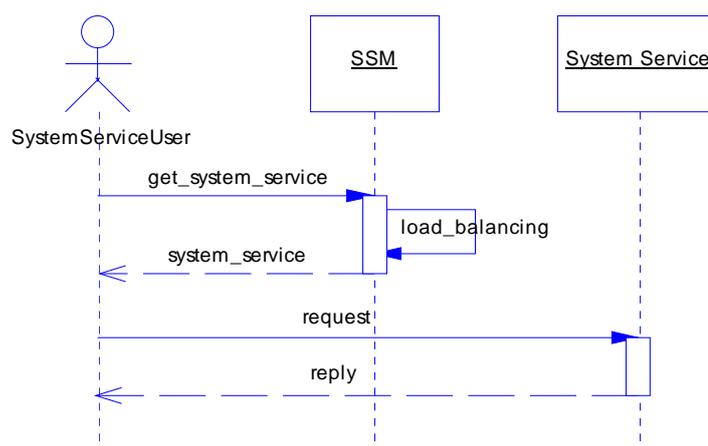
**Fig. IV.5** Registrazione di un servizio di sistema

Al momento dell'attivazione da parte dell'amministratore il servizio reperisce il reference dell'SSM e richiede la propria registrazione. Affinché l'operazione vada a buon fine i dati di cui ha bisogno il manager sono: il file di descrizione del servizio, il suo riferimento e il nome simbolico con il quale sarà registrato.

### IV.3.2 UTILIZZO DI UN SERVIZIO DI SISTEMA

Per utilizzare un servizio di sistema è necessario passare attraverso il manager relativo, infatti solo esso conosce il riferimento e lo stato di tali servizi.

La richiesta di un servizio di sistema può essere schematizzata come segue:



**Fig. IV.6** Utilizzo di un servizio di sistema

Come si può notare, l'interazione con il manager avviene solo in fase iniziale. Una volta ottenuto il riferimento, in base al nome simbolico, è possibile comunicare direttamente con il servizio richiesto.

Nel prosieguo la richiesta del servizio di sistema includerà il nome del servizio stesso e, per semplicità, verrà omessa la chiamata al metodo *load\_balancing*.

### IV.3.3 REGISTRAZIONE DI UN SERVIZIO APPLICATIVO

Affinché un servizio applicativo possa registrarsi è necessario che, oltre all'*SSM* e all'*Ontology Service*, sia già attivo anche l'*Application Services Manager*. I passi relativi a tale operazione sono riportati nella figura seguente:

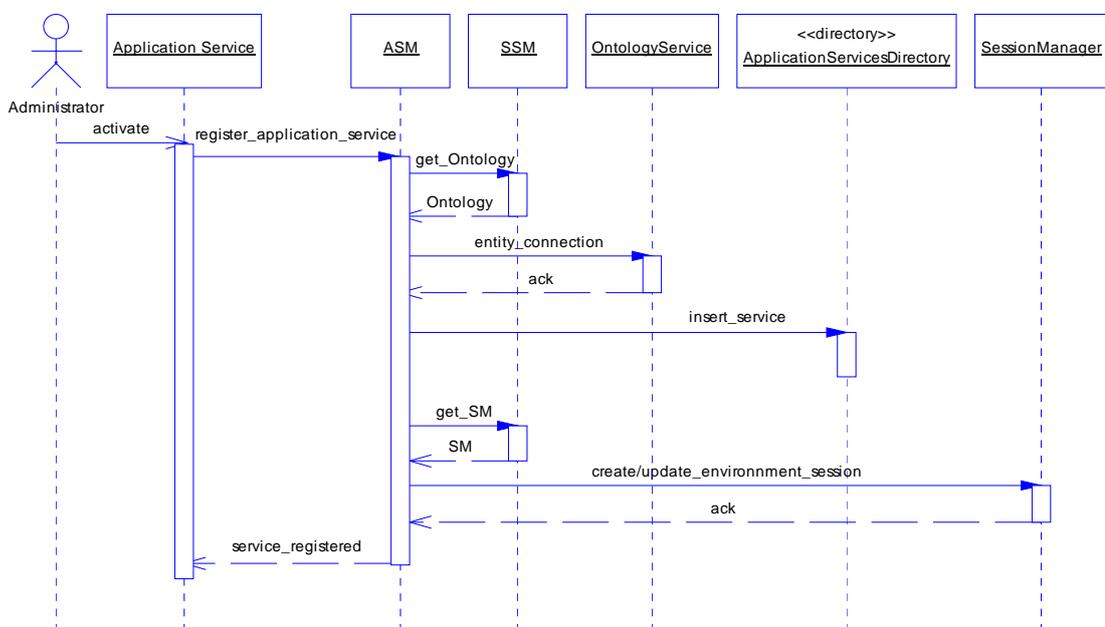


Fig. IV.7 Registrazione di un servizio applicativo

Al momento dell'attivazione, il servizio reperisce il reference dell'*Application Services Manager* ed esegue la richiesta di registrazione inviando il proprio file di descrizione, il reference e il nome simbolico. Successivamente l'*ASM* provvede a richiedere all'*Ontology* la validazione della descrizione del servizio e, se questa va a buon fine, esegue la registrazione presso l'*Application Services Directory*.

In questa fase si evidenzia la differenza esistente tra un servizio applicativo di utente e un servizio applicativo di ambiente. Infatti i servizi appartenenti alla seconda categoria non possono essere invocati direttamente da un utente, pur essendo stati creati per offrirgli delle funzionalità (si pensi alla musica d'ambiente).

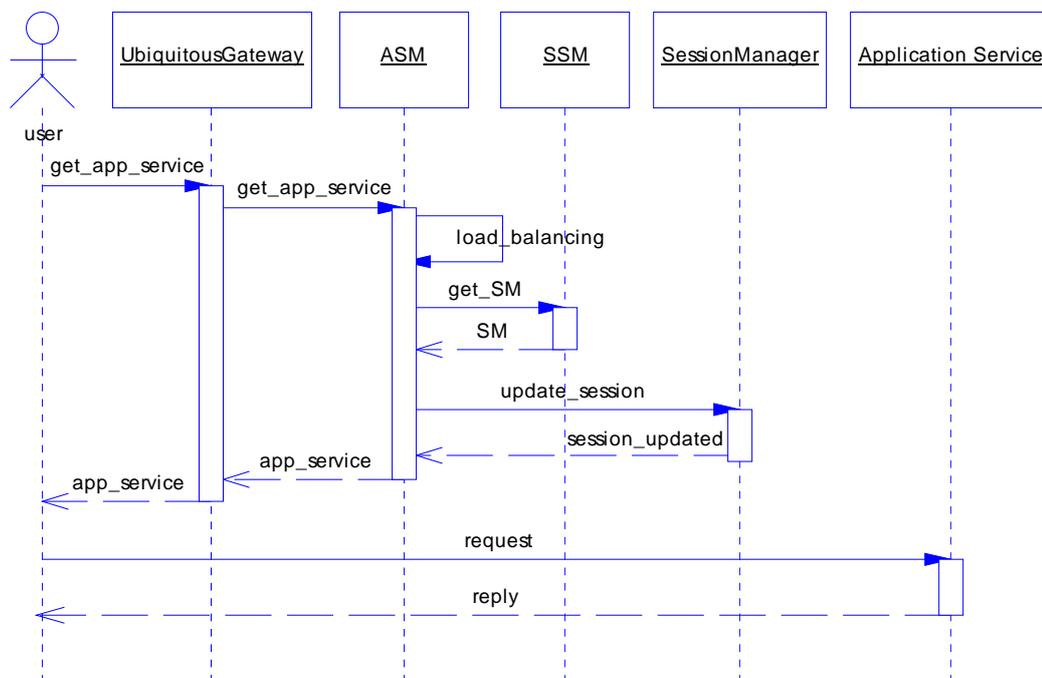
Quando la registrazione viene richiesta da un servizio d'ambiente, l'ASM richiederà al *Session Manager* la creazione o l'aggiornamento della sessione di "Environment". Ovviamente la creazione avverrà in corrispondenza della registrazione del primo servizio d'ambiente.

#### IV.3.4 UTILIZZO DI UN SERVIZIO APPLICATIVO

Analogamente a come visto per i servizi di sistema, per utilizzare un servizio di applicativo è necessario passare attraverso il manager relativo, infatti solo esso conosce il riferimento e lo stato di tali servizi.

La richiesta di un servizio applicativo può provenire sia dall'utente che da un altro servizio applicativo, per realizzare la composizione dei servizi.

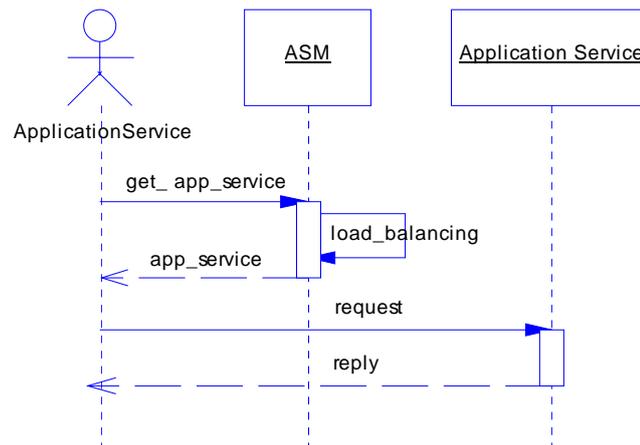
Nel caso in cui il richiedente è l'utente finale le operazioni da compiere risultano:



**Fig. IV.8** Utilizzo di un servizio applicativo da parte dell'utente

Dalla figura è possibile notare il coinvolgimento del *Session Manager*, infatti ogni qualvolta un utente utilizza un servizio, la relativa sessione andrà aggiornata con il riferimento della particolare istanza assegnata all'utente.

Se la richiesta proviene da un altro servizio le operazioni da compiere risultano:



**Fig. IV.9** Composizione di servizi applicativi

### IV.3.5 REGISTRAZIONE DI UN DEVICE

Come tutte le entità presenti in *UbiSystem*, anche un dispositivo utente, quando entra nell'ambiente, dovrà essere registrato.

La caratterizzazione del dispositivo mediante il quale un utente accede alle risorse dell'ambiente è un aspetto molto importante. Infatti da queste dipende il modo in cui il client potrà usufruire dei servizi. Ad esempio, mentre è ragionevole supporre uno streaming audio-video su di un dispositivo PDA, non lo è altrettanto se l'utente lo richiede da un telefono cellulare. Oppure potrebbe cambiare l'interfaccia mediante la quale avviene l'interazione: per un PDA potrebbe essere fornita un'interfaccia di tipo grafica mentre per un cellulare si dovrebbe pensare ad un'interfaccia a caratteri.

Un servizio potrebbe fornire più implementazioni dai requisiti non funzionali diversi (parametri di qualità, banda, ritardi, etc.), quindi, in base alle caratteristiche del dispositivo utente si sceglie l'implementazione più adeguata.

Le caratteristiche del device, sia di carattere hardware che software, vengono descritte in un file DAML+OIL prodotto dall'Ubiquitous Gateway.

L'ingresso dell'utente nel sistema si identifica con la prima connessione alla home page di *UbiSystem*. In questo momento il componente *Ubiquitous Gateway* provvede a verificare se già esiste una sessione relativa a quel device, in caso affermativo vuol dire che l'utente è già entrato nel sistema in precedenza, quindi il device è già stato registrato. In caso contrario, come nella situazione rappresentata nella figura seguente, l'*UG* dovrà provvedere alla costruzione del file di descrizione del device e alla sua registrazione.

Se quest'ultima operazione va a buon fine esso richiederà al *Session Manager* la creazione di una nuova sessione.

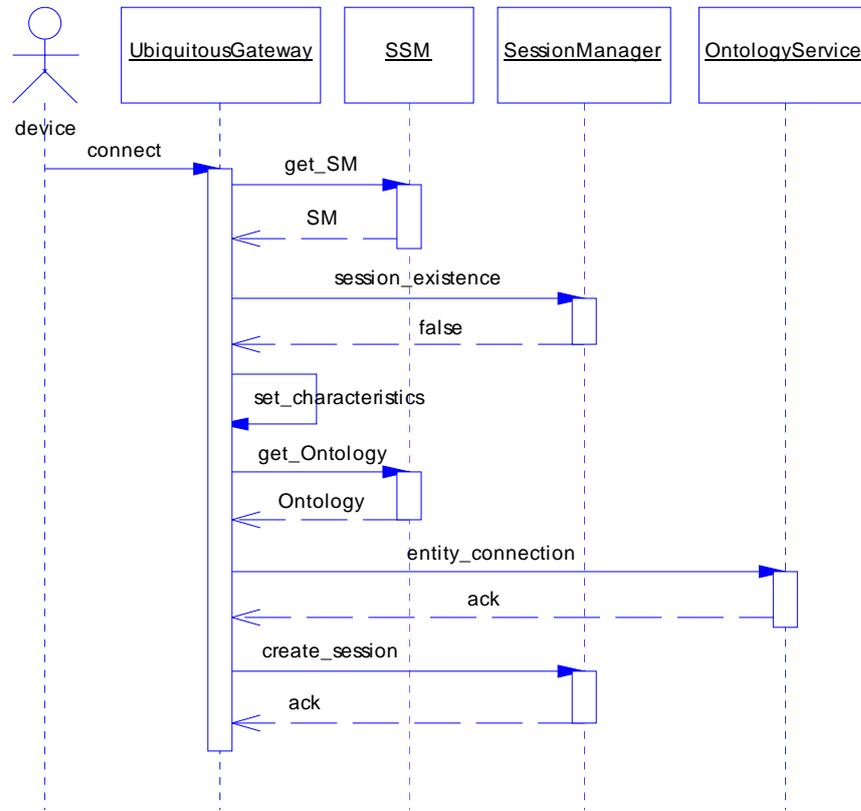


Fig. IV.10 Registrazione di un device

### IV.3.6 REGISTRAZIONE DI UN SERVIZIO OFFERTO DALL'UTENTE

Quando un utente vuole rendere disponibile un proprio servizio all'interno dell'ambiente *UbiSystem* deve fare in modo che questo si registri presso il manager dei servizi applicativi.

L'interazione tra utente e sistema può avvenire solo attraverso *Ubiquitous Gateway*, quindi è questo componente a richiedere la registrazione presso l'*ASM*.

Il servizio deve fornire un file di descrizione, il suo reference e il nome simbolico con cui sarà individuato e richiede all'*UG* la registrazione.

Data l'importanza della funzionalità, si suppone che la possibilità di offrire un servizio sia concessa solamente ad utenti autenticati che abbiano un livello di accesso adeguato. Per questo sarà necessario verificare che colui che richiede di mettere a disposizione un servizio abbia i diritti opportuni.

Nella figura riportata di seguito si mostra la sequenza di operazioni da compiere per registrare un servizio offerto dall'utente.

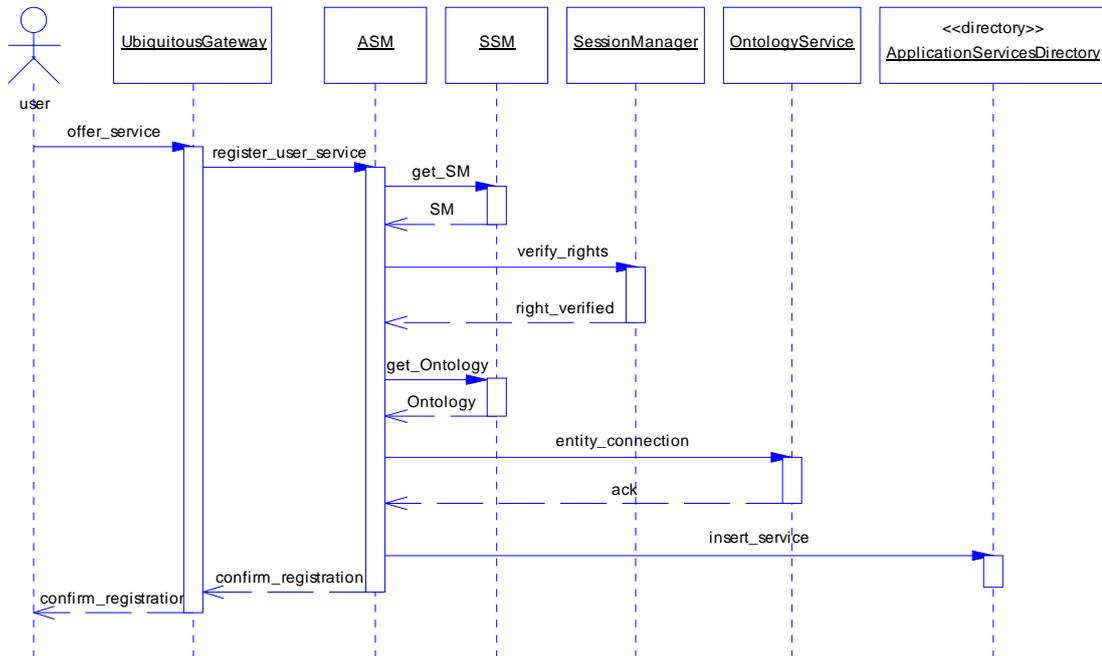


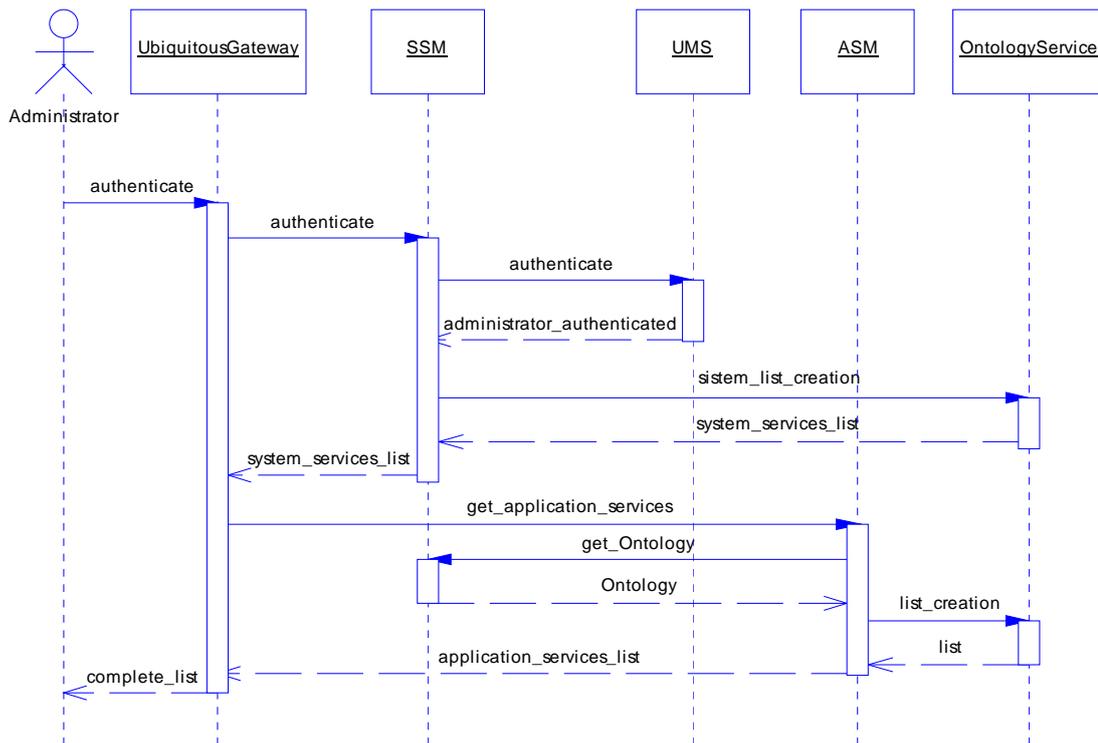
Fig. IV.11 Registrazione di un servizio offerto dall'utente

### IV.3.7 RICHIESTA DELLA LISTA DEI SERVIZI DA PARTE DELL'AMMINISTRATORE

Per poter sapere quali servizi sono attivi nel sistema, ogni utente deve richiederne la lista. Anche l'amministratore può entrare nell'ambiente come un utente, ma se si autentica come amministratore avrà la lista completa di tutti i servizi, sia applicativi che di sistema.

Quando *Ubiquitous Gateway* si accorge che la login è quella dell'amministratore, chiede l'autenticazione al *System Services Manager*, se questa operazione va a buon fine esso ottiene la lista dei servizi di sistema. Successivamente ottiene dall'*ASM* la lista dei servizi applicativi e restituisce all'amministratore la lista completa.

Questa funzionalità è rappresentata graficamente nella figura seguente.



**Fig. IV.12** Lista dei servizi per l'amministratore

In questo modo l'amministratore potrà modificare le impostazioni dei servizi di sistema anche quando il sistema è su. Ad esempio registrare un nuovo utente, modificare i tempi di lease degli indirizzi IP...

#### IV.3.8 RICHIESTA DELLA LISTA DEI SERVIZI APPLICATIVI

La richiesta della lista dei servizi applicativi può essere fatta sia da un utente anonimo, sia da un utente che già si è autenticato o che si autentica per la prima volta.

La lista dei servizi è creata dall'*Ontology Service*, e sarà quest'ultimo a realizzarla tenendo conto dei diritti dell'utente, delle caratteristiche del dispositivo con cui sta interagendo e della sua locazione.

Nel caso in cui la richiesta avviene in concomitanza dell'autenticazione le operazioni eseguite sono quelle mostrate in figura.

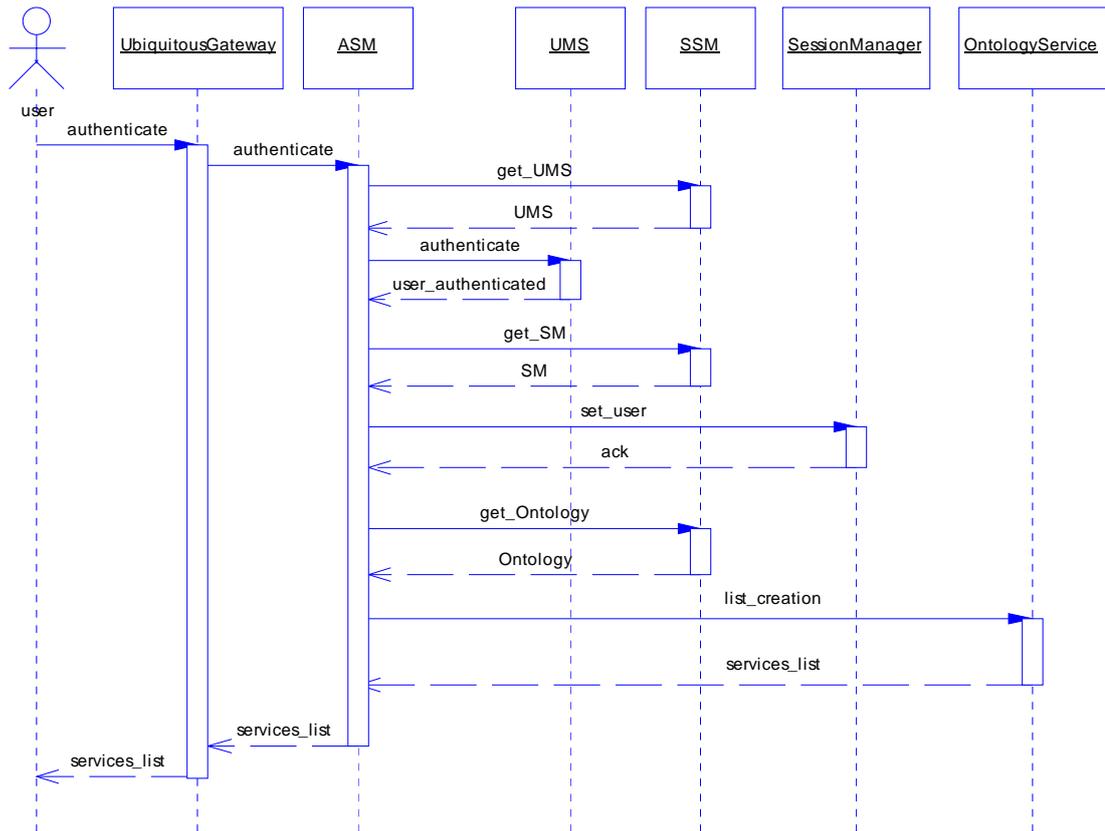


Fig. IV.13 Autenticazione di un utente

Nel caso in cui l'utente richieda direttamente la lista dei servizi, sarà l'*Ontology Service* a verificare i diritti posseduti dall'utente facendone richiesta al *Session Manager* al momento della creazione della lista. Si riporta di seguito il sequenze relativo.

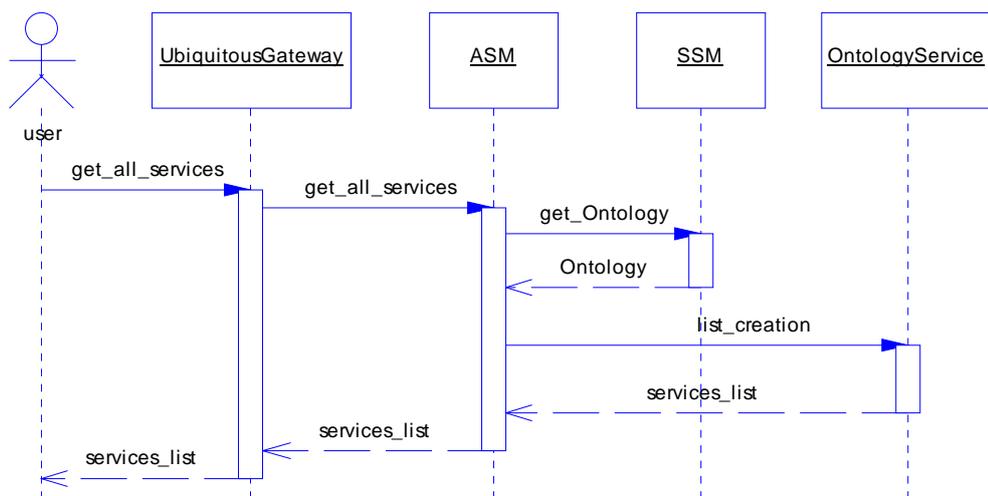


Fig. IV.14 Richiesta lista servizi applicativi

### IV.3.9 RILASCIO DI UN SERVIZIO APPLICATIVO

Nel momento in cui l'utente smette di utilizzare un servizio interattivo, oppure un servizio batch completa la sua elaborazione, il servizio stesso lo dovrà comunicare al relativo gestore.

A tale scopo l'ASM mette a disposizione un metodo per il rilascio del servizio. In seguito ad una comunicazione di questo tipo il manager si dovrà occupare della deallocazione delle risorse relative al servizio e dell'invocazione del suo distruttore.

Di seguito è riportato il Sequence Diagram relativo.

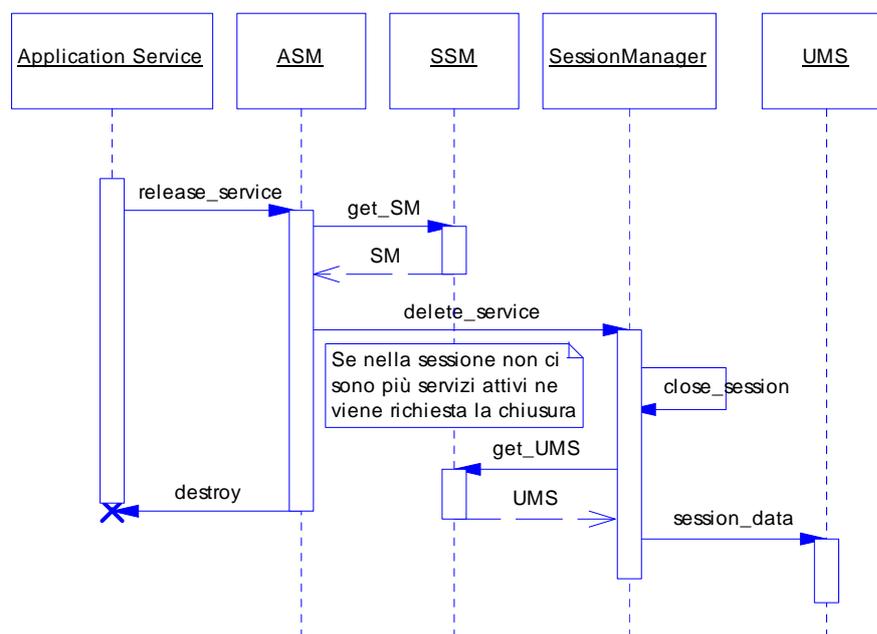


Fig. IV.15 Rilascio di un servizio applicativo

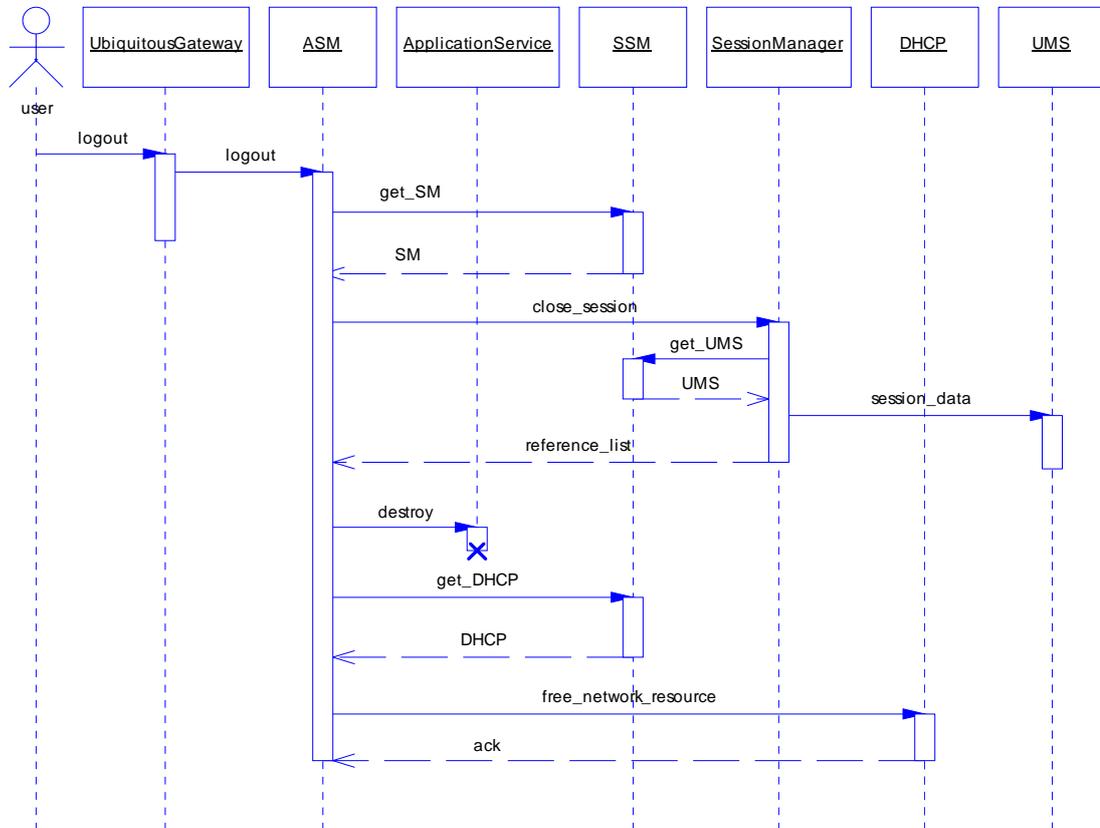
Ogni volta che viene richiesta la rimozione di un servizio applicativo dalla sessione si verifica se ne sono presenti altri, in caso contrario la sessione viene chiusa e, se si tratta di un utente autenticato l'*SM* provvede all'invio dei dati in essa contenuti all'*UMS*. In questo modo l'*User Manager Service* ha la possibilità di eseguire degli studi statistici sulle preferenze dei diversi utenti.

### IV.3.10 LOGOUT

Con il logout l'utente comunica al sistema di voler abbandonare l'ambiente. In tal caso vengono chiusi tutti i servizi applicativi aperti dall'utente compresi quelli computazionali. Di conseguenza devono essere deallocate tutte le risorse relative ai

servizi in uso nella sessione dell'utente e devono essere chiamati i distruttori delle relative istanze.

Le azioni svolte all'interno del sistema si schematizzano come segue:



**Fig. IV.16 Logout**

Nel momento in cui viene chiusa la sessione di un utente autenticato, il *Session Manager* provvede ad inviare all'*User Manager Service* tutti i dati relativi all'attività svolta.

Ovviamente, anche l'amministratore può richiedere il logout. In questo caso *UG* dovrà invocare un metodo analogo presso il gestore dei servizi di sistema e di conseguenza la sessione relativa sarà chiusa.

#### IV.3.11 ABBANDONO DELL'AMBIENTE SENZA LOGOUT

L'utente potrebbe abbandonare il sistema senza dare una comunicazione esplicita.

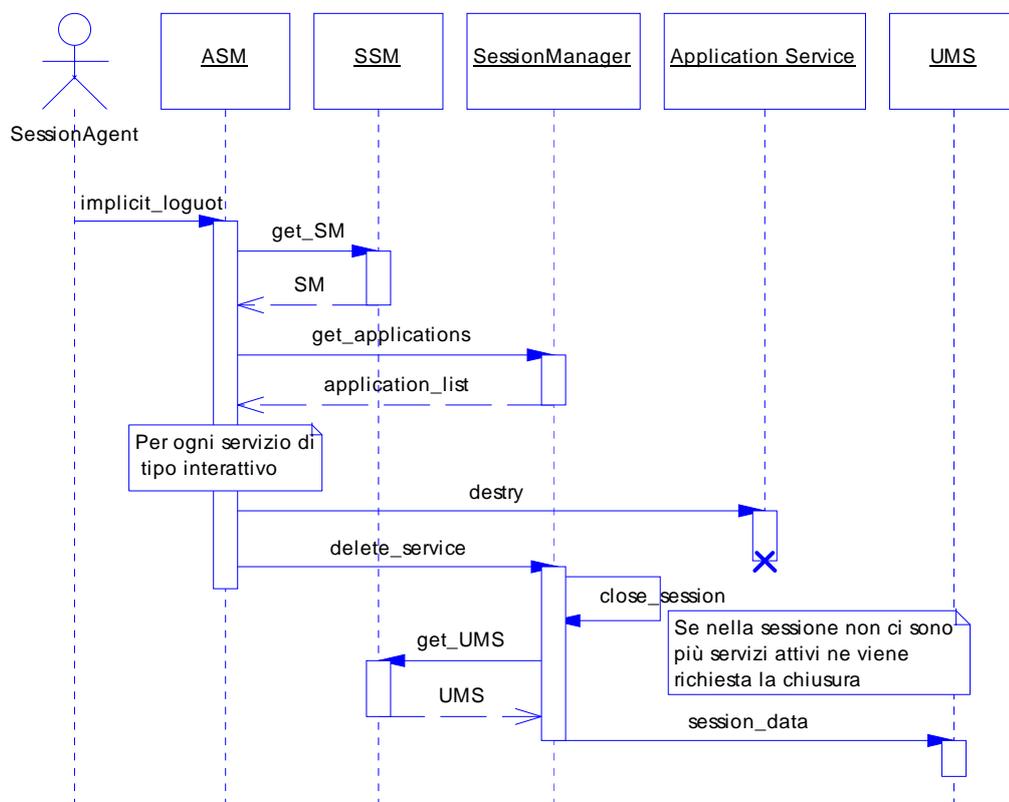
Allo scopo di individuare un evento di questo tipo, ad ogni sessione è associato un agente che, periodicamente, verifica il tempo trascorso dall'ultima interazione con il

sistema. Dopo un certo periodo di inattività esso provvede a notificare la cosa all'ASM.

Si può verificare il caso in cui l'utente abbia richiesto un servizio di tipo batch, e quindi la sessione non andrà chiusa.

Il manager dei servizi applicativi provvede a deallocare le risorse relative ai soli servizi di tipo interattivo e ne richiede la rimozione dalla sessione. Questa sarà chiusa solo nel momento in cui tutti i servizi saranno rimossi.

Il diagramma seguente sintetizza le operazioni svolte per realizzare tale funzionalità.



**Fig. IV.17 Logout implicito**

Nel momento in cui viene chiusa la sessione di un utente autentificato, il *Session Manager* provvede ad inviare all'*User Manager Service* tutti i dati relativi all'attività svolta.

### IV.3.12 COMPLETAMENTO DEI BUSINESS CLASS DIAGRAMS

Dall'analisi dei Sequence Diagrams sorge la necessità di prevedere, per i componenti considerati, ulteriori metodi oltre a quelli individuati a partire dagli Use Case Diagrams.

In particolare, per l'*Application Services Manager*, bisogna introdurre il metodo *implicit\_logout()* per gestire la situazione in cui l'utente va via dall'ambiente senza darne segnalazione.

## IV.4 CLASS DIAGRAMS

Un *Class Diagram* mostra la struttura statica del modello, in altre parole, le cose che esistono (classi e tipi), la loro struttura interna e le relazioni occorrenti tra loro e con altri oggetti. Questi diagrammi, al contrario dei *Sequence Diagrams* analizzati nel precedente paragrafo, non riportano informazioni temporali.

Dopo aver analizzato le iterazioni tra i vari moduli di *UbiSystem* e aver individuato i metodi che ognuno di essi deve esporre è possibile passare ad un livello di astrazione più basso.

Lo scopo di questo paragrafo è di mostrare i diagrammi delle classi dei componenti implementati.

### IV.4.1 UBIQUITOUS GATEWAY

Questo componente rappresenta il punto di ingresso per gli utenti di *UbiSystem* e per l'amministratore.

A seconda di chi inoltra la richiesta il sistema reagisce in maniera diversa. Innanzi tutto risulta utile evidenziare la differenza tra un utente generico e l'amministratore, infatti quest'ultimo può interagire anche con i servizi di sistema e può monitorare i servizi applicativi. Per questo motivo sono presenti due interfacce.

A questo livello sono state individuate delle strutture dati utili per la gestione del sistema che sono usate anche come parametri di ingresso-uscita:

- *Service*. Questa classe raggruppa le caratteristiche principali di un servizio di *UbiSystem* necessarie per far conoscere il servizio all'utente: nome, descrizione testuale e diritti d'accesso.

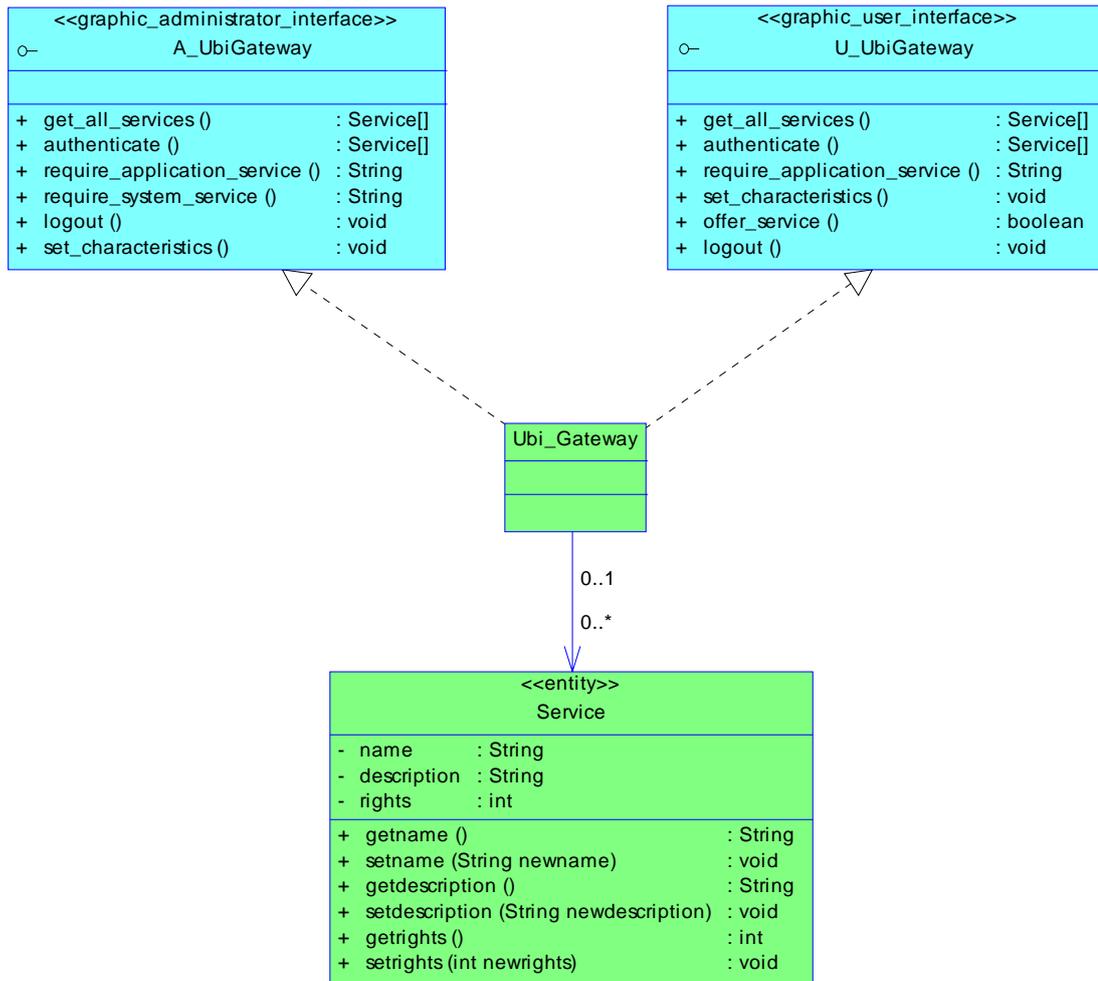
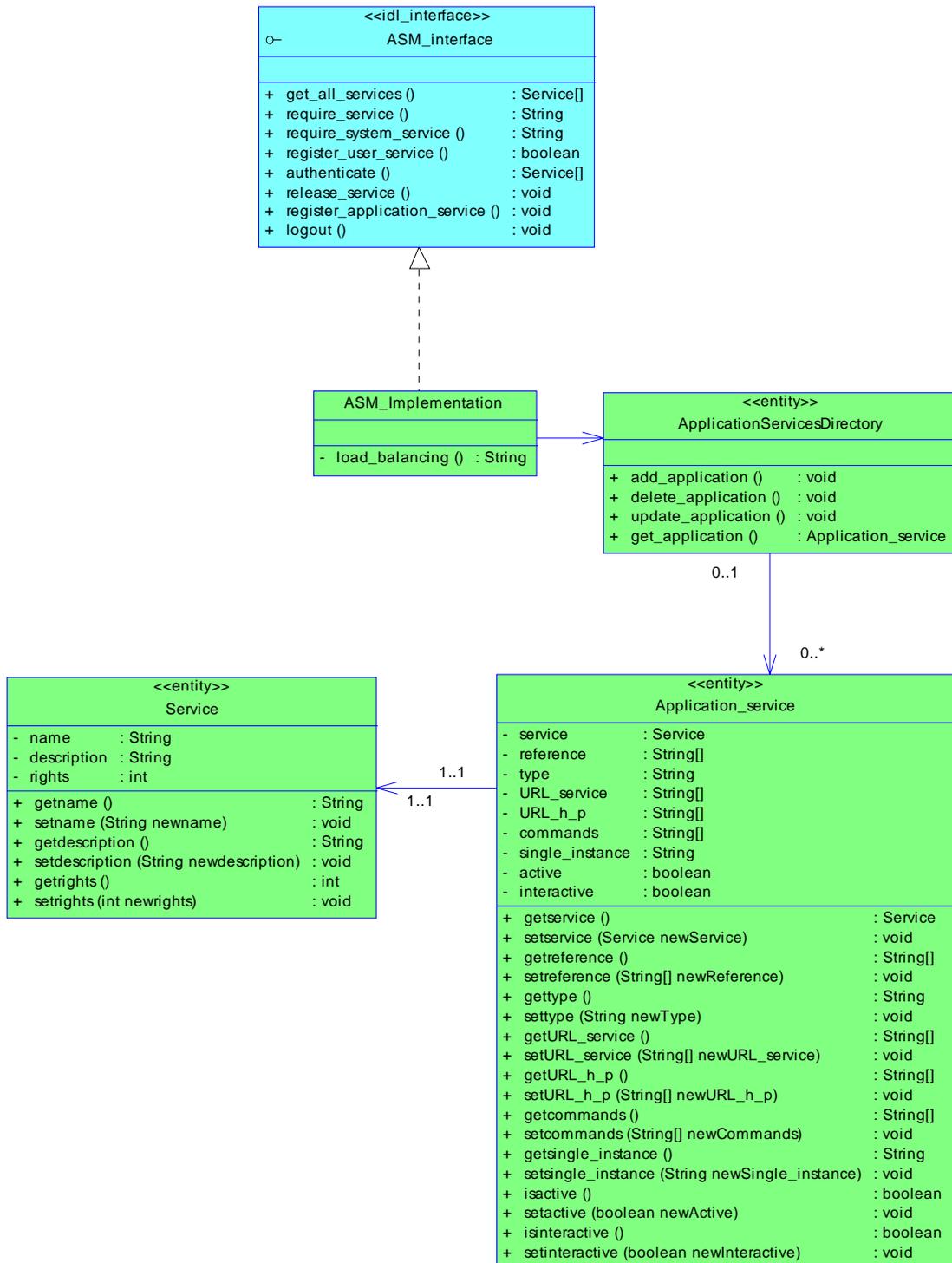


Fig. IV.18 Diagramma delle classi di Ubiquitous Gateway

#### IV.4.2 APPLICATION SERVICES MANAGER

Come per il componente appena analizzato, anche per il manager dei servizi applicativi è possibile individuare un'interfaccia, attraverso la quale è possibile usufruire delle funzionalità da esso offerte.



**Fig. IV.19** Diagramma delle classi di Application Services Manager

Le strutture dati necessarie all'ASM sono le stesse viste per *UG*, ma in più dovrà prevedere la classe *Application\_service* che contiene tutte le informazioni utili per la gestione di un servizio applicativo.

Gli oggetti di questa classe saranno memorizzati nella lista *ApplicationServiceDirectory*.

### IV.4.3 SYSTEM SERVICES MANAGER

La struttura del *System Service Manager* è molto simile a quella del gestore dei servizi applicativi. L'unica differenza sta nella struttura che rappresenta il servizio di sistema data la sua differente natura.

Di seguito è riportato il diagramma delle classi dell'*SSM*.

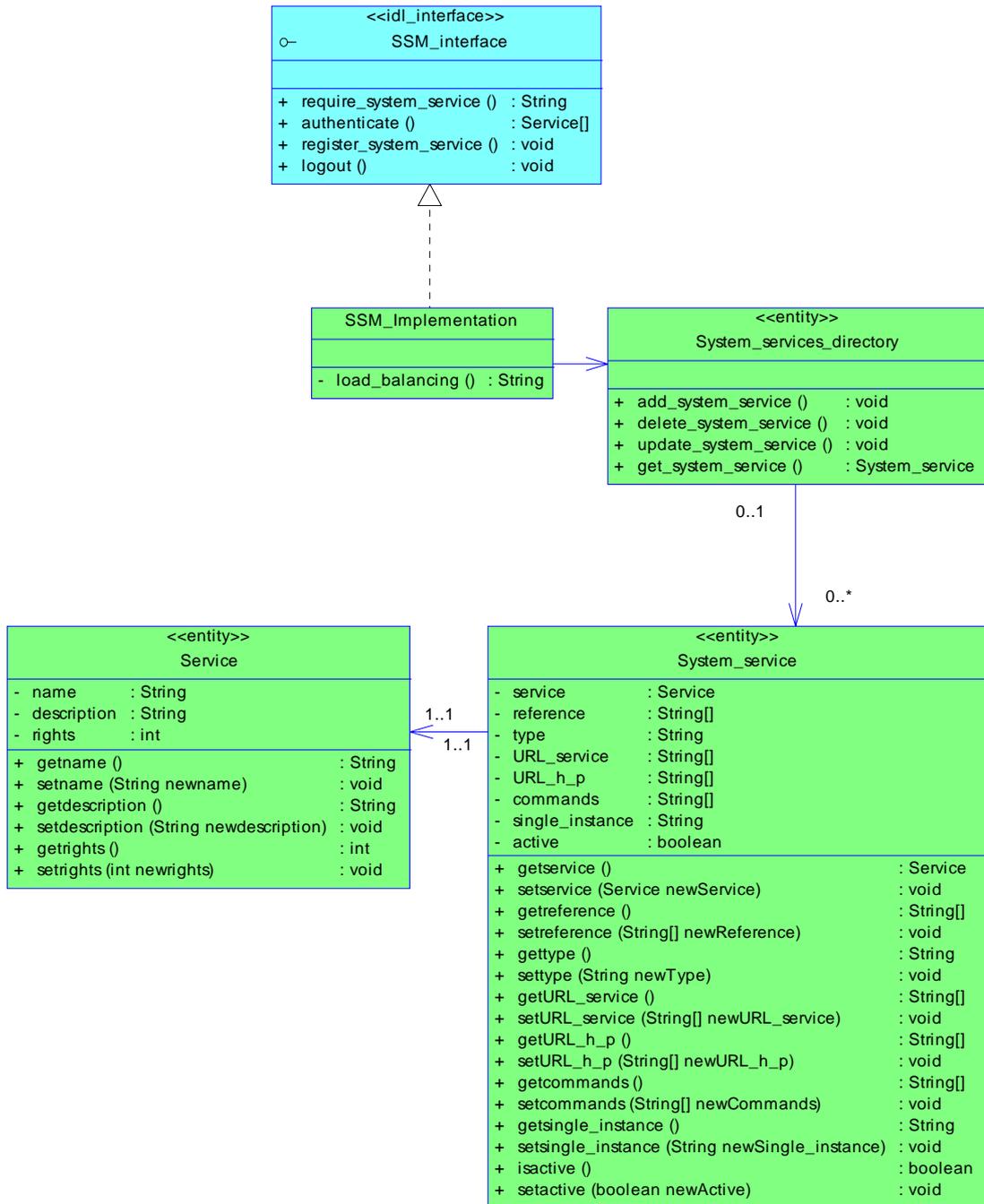


Fig. IV.20 Diagramma delle classi di System Services Manager

## IV.5 DETTAGLI IMPLEMENTATIVI

Al termine della fase progettuale è possibile passare alla fase implementativa.

Tutti i componenti visti sono stati implementati come oggetti CORBA e solamente *Ubiquitous Gateway* è stato esposto come servizio web. Solo in questo modo, infatti, le funzionalità che espone potranno essere accessibili dall'esterno.

### IV.5.1 IMPLEMENTAZIONE DI UN OGGETTO CORBA

La prima cosa da fare per realizzare un oggetto Corba scrivere la sua interfaccia. Questa sarà scritta in un apposito linguaggio per la definizione delle interfacce: *Interface Definition Language (IDL)*.

IDL permette la definizione del nome dell'oggetto Corba, la dichiarazione dei metodi, la costruzione di nuovi tipi di dati, le modalità di passaggio di parametri e la definizione delle eccezioni.

Un compilatore IDL riceve la descrizione dell'interfaccia ed esegue una corrispondenza tra l'IDL ed il linguaggio in cui è realmente implementato l'oggetto.

La compilazione di una interfaccia produce almeno due file: lo *stub* (lato client) e lo *skeleton* (lato server). Lo stub del client rappresenta l'interfaccia attraverso cui un client può richiedere servizi a un server. Cioè contiene tutte le definizioni delle operazioni dell'interfaccia IDL dell'oggetto in forma di definizioni di metodo del linguaggio di programmazione utilizzato. Dal punto di vista del client, rappresenta la parte dell'oggetto CORBA (la più *vicina*) al client con cui esso interagisce. Il client cioè esegue di fatto una invocazione locale allo stub, che "impacchetta" (*marshalling*) i dati dell'invocazione in un messaggio che viene consegnato all'ORB e da questo inviato al server. Di conseguenza allo stub del client è associato a tempo di esecuzione un riferimento remoto all'oggetto (Object Reference), che è il mezzo utilizzato dal client per invocare operazioni sull'oggetto.

Lo skeleton è l'equivalente dello stub dal lato dell'implementazione dell'oggetto. Esso rappresenta il vero oggetto CORBA poiché fornisce lo scheletro dell'oggetto lasciando vuoto lo spazio riservato all'implementazione dei metodi che dovrà essere riempito da un programmatore. Lo skeleton si differenzia dallo stub poiché coopera con un Object Adapter per le operazioni che riguardano l'attivazione dell'oggetto.

Lo skeleton riceve la richiesta dall'Object Adapter, ne estrae i dati (*unmarsahling*) e li passa all'implementazione dell'oggetto.

La forma in cui un oggetto si presenta al client è chiamata *Object Reference* (OR). Un OR è, nel senso più generale del termine, un puntatore all'oggetto, cioè un mezzo per poter accedere all'oggetto senza di fatto dover "utilizzare" l'oggetto vero e proprio. Un OR per un oggetto viene creato insieme all'oggetto stesso e continua a esistere lungo tutto il suo tempo di vita, durante il quale continuerà a riferirsi solamente a quell'oggetto. Cioè l'OR smette di essere valido solo quando l'oggetto viene permanentemente distrutto e da quel momento in poi non sarà più riutilizzabile in seguito per altri oggetti diversi. Un OR è quindi associato ad un solo oggetto, ma non è vero il viceversa, un oggetto cioè può avere diversi OR.

In CORBA è definito un formato standard per gli OR grazie al quale viene garantita l'interoperabilità tra ORB diversi. OR che segue questo formato è chiamato *Interoperable Object Reference* (IOR).

Affinché il client possa ottenere l'OR di un oggetto, prima questo deve essere stato creato e reso disponibile da un server. Questo può accadere in diversi modi:

- Restituendo l'OR come il risultato di un'operazione.
- Utilizzando un servizio CORBA, tipicamente il *Naming Service*
- Convertendo l'OR in una stringa che potrà essere salvata su un file system condiviso o inviata via rete. L'interfaccia dell'ORB fornisce le opportune operazioni per convertire OR a e da stringhe.

Per casi particolari, come ad esempio proprio il *Naming Service* (che è esso stesso un oggetto CORBA), è l'ORB che restituisce l'OR attraverso un'operazione della sua interfaccia.

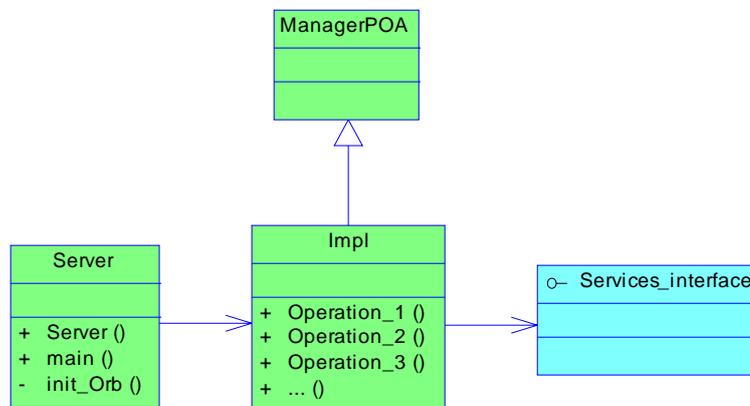
Per rendere più chiaro il meccanismo si considera la realizzazione di un generico oggetto avente le stesse caratteristiche di quelli realmente implementati.

Il primo passo da fare è la scrittura dell'interfaccia IDL salvata nel file: "*Manager.idl*". Questa è data in pasto al compilatore *idl-java* che genera tre gruppi di classi:

- Classi lato Client: *\_ManagerStub.java*, *Manager.java*
- Classi Lato Server: *ManagerPOA.java*, *ManagerPOATie.java*, *ManagerOperations.java*
- Classi di supporto: *ManagerHelper.java*, *ManagerHolder.java*

Successivamente sarà necessario scrivere le classi rappresentanti il server e l'implementazione (*Server.java* e *Impl.java*), dal lato server, e la classe client (*Client.java*) dal lato client.

Il diagramma delle classi dell'oggetto corba in esame risulta:



**Fig. IV.21** Diagramma delle classi di un oggetto Corba

La classe *Impl.java* ha lo scopo di implementare tutte le operazioni definite nell'interfaccia IDL.

La classe *Server.java* costituisce il processo server e contiene le chiamate CORBA necessarie per attivare l'implementazione all'interno dell'ORB. La sequenza di operazioni svolte all'interno del server è questa:

- *Inizializzazione ORB.* Prima di ogni operazione CORBA è necessario inizializzare l'oggetto ORB attraverso l'operazione statica `init()`. In questo modo si ottiene anche il riferimento all'ORB necessario per il successivo utilizzo.
- *Creazione riferimento al POA.* Il riferimento iniziale al POA viene restituito dall'ORB stesso attraverso l'operazione `resolve_initial_reference()`. Questa operazione è in grado di restituire il riferimento di alcuni oggetti speciali che possono essere specificati attraverso il loro nome.
- *Attivazione oggetto nel POA.* Un'istanza dell'implementazione dell'oggetto viene associata dal POA all'oggetto stesso, di cui viene creato e restituito l'Object Reference. L'oggetto a questo punto si manifesta attraverso il suo Object Reference, attraverso il quale i client possono effettuare invocazioni risalendo al servant che lo incarna.

- *Pubblicazione Object Reference.* L'OR dell'oggetto viene reso disponibile ai client registrandolo nel *Naming Service* o trasferendolo in forma di stringa (ad esempio mediante scrittura in un file condiviso o e-mail). La conversione dell'OR a e da stringa avviene attraverso opportune operazioni dell'ORB.
- *Attivazione ORB.* L'ORB viene esplicitamente attivato e posto in uno stato di ascolto delle richieste.

Per quanto riguarda gli oggetti implementati la pubblicazione dell'OR avviene presso il *Naming Service*.

## IV.5.2 UBIQUITOUS GATEWAY

### IV.5.2.1 UG COME OGGETTO CORBA

Si riporta di seguito l'interfaccia IDL dell'*Ubiquitous Gateway*.

```
module UbiSys{
    struct Account {
        string login;
        string password;
    };
    struct Service {
        string name;
        string description;
        long rights;
    };
    //Definizione lista di servizi
    typedef sequence <Service> Services;
    module UbiquitousGateway{
        interface UG {
            //Eccezioni sollevate
            exception UGerror{};

            string require_application_service(in string service_name,
            in string ip) raises (UGerror);

            string require_system_service(in string service_name,
            in string ip) raises (UGerror);

            boolean offer_service(in string description_file,
```

```
        in string ip) raises (UGerror);

Services get_all_services(in string ip)
    raises (UGerror);

Services authenticate(in Account account, in string ip)
    raises (UGerror);

void logout(in string ip)
    raises (UGerror);

void set_characteristics(in string ip, in string os)
    raises (UGerror);

string get_login(in string ip)
    raises (UGerror);

    };//end interface UG
};//end module UbiquitousGateway
};//end module UbiSys
```

Le strutture dati definite sono:

- *Account*. Si tratta di una struttura dati atta a definire l'account di un utente. E' definita in questo componente in quanto il metodo *Authenticate()* prevede un parametro di ingresso di tipo *Account*.
- *Service*. Tale struttura dati definisce le caratteristiche di un servizio presente nell'ambiente. In particolare:
  - il campo *name* rappresenta il nome con cui il servizio è conosciuto nell'ambiente;
  - il campo *description* contiene una breve descrizione testuale riguardante il servizio;
  - il campo *rights* indica il minimo valore del livello di accesso che deve possedere l'utente per poter usufruire delle funzionalità del servizio.
- *Services*. Rappresenta una lista di servizi. In altri termini, un array di *Service*.

I metodi implementati sono:

- *string require\_application\_service(in string service\_name, in string ip)*. Tale metodo riceve in ingresso il nome del servizio applicativo e l'ip del dispositivo da cui proviene la richiesta e restituisce una stringa che rappresenta l'URL della HomePage del servizio. Esso non fa altro che inoltrare la richiesta al gestore dei servizi applicativi.
- *string require\_system\_service(in string service\_name, in string ip)*. Tale metodo riceve in ingresso il nome del servizio di sistema e l'ip del dispositivo da cui proviene la richiesta, cioè il dispositivo con cui l'amministratore interagisce con il sistema, e restituisce una stringa che rappresenta l'URL della HomePage del servizio di sistema. Esso non fa altro che inoltrare la richiesta al gestore dei servizi di sistema.
- *boolean offer\_service(in string description\_file, in string ip)*. Al momento questo metodo non è implementato, ma dovrebbe permettere la registrazione di un servizio offerto dall'utente stesso.
- *Services get\_all\_services(in string ip)*. Tale metodo riceve in ingresso l'ip del dispositivo da cui proviene la richiesta e restituisce la lista dei servizi disponibili in base alle caratteristiche dell'utente. Esso non fa altro che invocare il metodo omonimo del gestore dei servizi applicativi.
- *Services authenticate(in Account account, in string ip)*. Questo metodo riceve in ingresso l'account dell'utente, cioè login e password che egli ha inserito, e restituisce la lista dei servizi disponibili in base alle caratteristiche dell'utente. Se la login è quella dell'amministratore viene invocato il metodo omonimo del *System Services Manager*, altrimenti quello esposto dall'*Application Services Manager*.
- *void logout(in string ip)*. Tale metodo riceve in ingresso l'ip del dispositivo da cui proviene la richiesta e ricava la login dell'utente

relativo. Se la login è quella dell'amministratore invoca il metodo omonimo del gestore dei servizi di sistema, altrimenti invoca il metodo messo a disposizione dal gestore dei servizi applicativi. Con la chiamata a questo metodo l'utente comunica la sua uscita dal sistema.

- *void set\_characteristics(in string ip, in string os)*. Questo metodo riceve in ingresso l'ip ed il sistema operativo del dispositivo da cui proviene la richiesta. Se da un controllo presso il *Session Manager* risulta che la sessione non esiste vuol dire che l'utente è appena entrato nel sistema. In questo caso, allora, il compito dell'*Ubiquitous Gateway* è quello di registrare il dispositivo dell'utente presso l'*Ontology Service* e chiedere al *Session Manager* la creazione della sessione. Affinché la registrazione abbia successo, l'*Ontology Service* ha bisogno di un file di descrizione del device scritto in linguaggio DAML+OIL. Per costruire in maniera corretta tale file, l'*Ubiquitous Gateway* dovrà prima classificare il device in base alle informazioni ottenibili dall'header della richiesta HTTP giunta al Web Service relativo. Per eseguire tutte le operazioni relative alla registrazione del dispositivo l'*Ubiquitous Gateway* si rivolge alla classe "*DeviceProvider*".
- *string get\_login(in string ip)*. Questo metodo restituisce il valore della login in base all'ip del dispositivo da cui proviene la richiesta. Tale informazione è prelevata presso il *Session Manager*. La necessità di introdurre questa funzionalità nasce in fase di implementazione. Infatti, a livello Web\_services saranno effettuati dei controlli sulla login.

#### IV.5.2.2 REGISTRAZIONE DEL DISPOSITIVO CLIENT

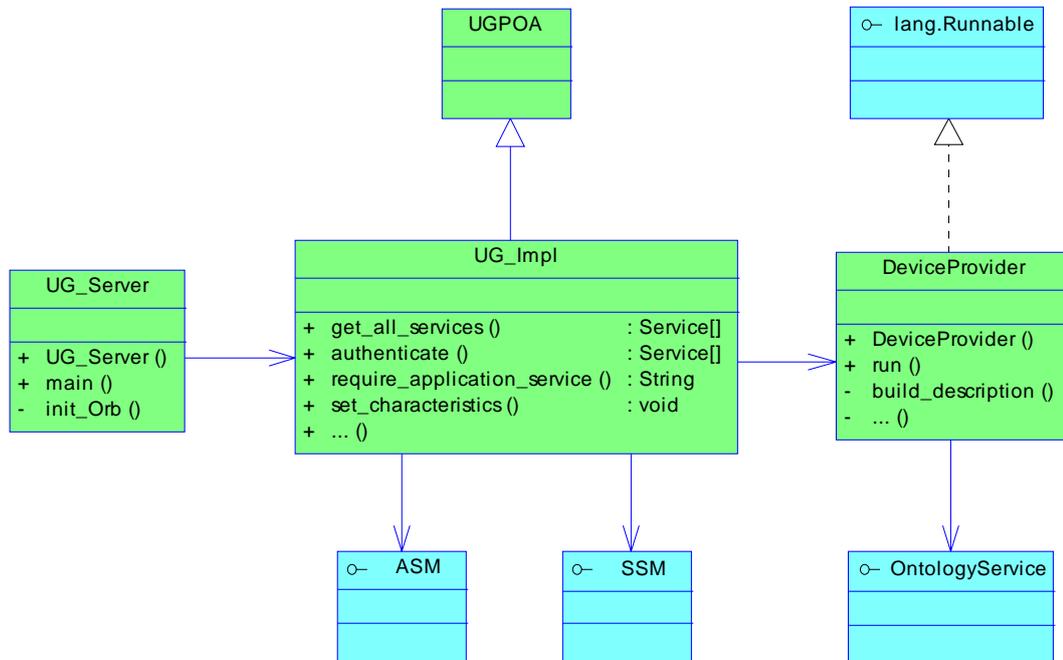
Nel momento in cui un utente si connette ad *UbiSystem* è necessario che il sistema venga a conoscenza delle caratteristiche del dispositivo con cui egli interagisce, quindi bisogna fare in modo che il dispositivo si registri presso l'*Ontology Service*.

Nella realizzazione di *UbiSystem* si cerca di evitare l'installazione di software aggiuntivo per la fruizione dei servizi, in accordo con le scelte progettuali, quindi la registrazione non può essere a carico del dispositivo client.

Visto che il ruolo che *Ubiquitous Gateway* gioca in *UbiSystem* è quello di entry point, sarà suo compito costruire la descrizione del dispositivo a partire dalle informazioni prelevate dall'header della richiesta http giunta ad esso.

*UG* svolge questo ruolo per mezzo della classe *DeviceProvider*.

Il diagramma delle classi di *UG* come oggetto Corba è riportato di seguito:



**Fig. IV.22** Diagramma delle classi di *UG* come oggetto Corba

Si riporta il codice che implementa il server:

```

public class UG_Server {
    public UG_Server(args) {
        initOrb(args);
    }
    public static void main(String[] args) {
        new UG_Server(args);
    }
    private void initOrb (String args[]){
        UG ug;
        try{
            // initializza l'ORB.
            ORB orb = ORB.init( args, null );

            // ottiene il riferimento al Naming Server
            NamingContextExt nc = NamingContextExtHelper.narrow(
                orb.resolve_initial_references("NameService"));
        }
    }
}
  
```

```
        //Ottiene rif POA
        POA rootPOA = POAHelper.narrow
            (orb.resolve_initial_references("RootPOA"));

        //Attiva POA
        rootPOA.the_POAManager().activate();

        //Crea l'obj servant
        UGimpl servant = new UGimpl();

        //registra obj nel POA
        ug = servant._this(orb);

        //Sul naming va il reference del servant
        nc.bind(nc.to_name("UG"), ug);

        //mette l'appl servente in stato di attesa
        orb.run();

    } catch (Exception e){
        e.printStackTrace();
    }
}
```

Il codice della classe DeviceProvider risulta:

```
public class DeviceProvider implements Runnable {

    private int delay = 6000;
    private Thread thread = null;
    private String file = new String();
    private String xmlDescription = new String();
    private OntologyServer MyOntology;
    private boolean device_registered = false;
    private String device_name = new String();
    private String device_id = new String();
    private String os = new String();

    public DeviceProvider() {
        thread = new Thread(this);
        thread.start();
    }
}
```

```
public DeviceProvider(OntologyServer Ontology, String
device_id, String os) {
    thread = new Thread(this);
    thread.start();
    MyOntology = Ontology;
    this.device_id = device_id;
    this.os = os;
}

public static void main(String [] args) throws Exception {
    new DeviceProvider();
}

public void run() {
    // Registra il servizio presso l'SSM
    if (buildDescription())
        subscribeService();
}

private boolean buildDescription() {
    .....
}

public boolean get_result() {
    return this.device_registered;
}

public String get_name() {
    return this.device_name;
}

private void subscribeService() {
    boolean go = true;
    while (go) {
        try {
            System.out.print("Tentativo registrazione del device...");
            device_registered = MyOntology.entityConnection
                                (xmlDescription);
            System.out.print("Registrazione del device terminata!");
            go = false;
        }catch (InvalidOntology e){
            System.out.println("Device not registered, invalid
                                Ontology");
        }catch (OntologyNotLoaded e){
```

```
        System.out.println("Device not registered, Ontology Not
                                Loaded");
    } catch (Exception e) {
        System.out.println("non riuscito.\nProblemi di connessione
                                con l'Ontology Server");
    }
    try {
        thread.sleep(delay);
    } catch (Exception e) {}
    }
}
```

Per chiarezza si riporta un esempio di descrizione di un laptop in ambiente *UbiSystem*.

Descrizione del Device UbiLaptop					
Info Description	Infoname			Massimo	
	Infovendor			Toshiba	
	Infoversion			Satellite 1400-103	
Devicetype				Laptop	
Offers				PdfViewer	
OwnedBy				Massimo Esposito	
Hw Description	Connection Description		InfoDescription	Infoname	Modem 56K
			Bandwidth		56
			Bandwidthunit		Kbit/s
	Connection Description		InfoDescription	Infoname	Lan Ethernet
			Bandwidth		100
			Bandwidthunit		Mbit/s
Cpu Description		InfoDescription	Infoname	Intel Celeron	
		Cpuspeed		1.33	
		Cpuspeedunit		GHz	
Ui Description	Screen Description		Width		28.25
			Height		22
			Unit		Cm
			Color		True
			Resolution Description	Resolutionwidth	1024
	Resolutionheight	768			
	Resolutionunit	Pixel			
	Resolutionbpp	32			
	Resolutiongraphics	True			
	Audio-input		False		
	Audio-output		True		
	Video-input		False		
Memory Description	Maximum	MemoryType Description	Memoryamount	20	
			Memoryunit	Gb	
			Memoryusagetype	Storage	
	Available	MemoryType Description	Memoryamount	6	
			Memoryunit	Gb	
			Memoryusagetype	Storage	
Memory Description	Maximum	MemoryType Description	Memoryamount	128	
			Memoryunit	Mb	
			Memoryusagetype	Application	
Sw Description	Os Description	InfoDescription		Infoname	Windows Xp
				Infovendor	Microsoft
				Infoversion	Home
	Application Description	InfoDescription		Infoname	Windows Media Player
				Infovendor	Microsoft
				Infoversion	8.0

Fig. IV.23 Descrizione di un laptop

## IV.5.2.3 UG COME WEB-SERVICE

Per la realizzazione del servizio web sono state utilizzate le librerie di Apache Axis [2] che mettono a disposizione una serie di toolkit che consentono la realizzazione di Web Services a partire da applicazioni Java.

Il primo passo da eseguire è la creazione di due classi Java che rappresentino rispettivamente l'interfaccia e l'implementazione del servizio: *UG.UG\_W.java* e *UG.UG\_W\_impl.java*.

```
package UbiSys.UG_Wrapping.UG;

public interface UG_W {

    public java.lang.String require_application_service
        (java.lang.String service_name, java.lang.String ip);

    public java.lang.String require_system_service
        (java.lang.String service_name, java.lang.String ip);

    public boolean offer_service
        (java.lang.String description_file, java.lang.String ip);

    public UbiSys.Service[] get_all_services(java.lang.String ip);

    public UbiSys.Service[] authenticate
        (UbiSys.Account account, java.lang.String ip);

    public void logout(java.lang.String ip);

    public void set_characteristics
        (java.lang.String ip , java.lang.String os);

    public java.lang.String get_login(java.lang.String ip);
}
```

La classe *UG\_W\_impl* sarà client dell'oggetto Corba descritto nel precedente paragrafo.

```
package UbiSys.UG_Wrapping.UG;

import java.io.*;
```

```
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import UbiSys.UbiquitousGateway.*;

public class UG_W_impl {

    UG UG_Reference;
    /** Creates a new instance of UG_W_impl */
    public UG_W_impl() {
        //Questa classe sar  "client" di UG_Server
        try{
            // initialize the ORB.
            ORB orb = ORB.init( (String[])null, null );

            NamingContextExt nc = NamingContextExtHelper.narrow(
                orb.resolve_initial_references("NameService"));

            //legge il riferimento di UG dal naming service
            UG_Reference=UGHelper.narrow(nc.resolve(nc.to_name("UG")));

        }catch (Exception e){
            e.printStackTrace();
        }
    }
    public String require_application_service
    (String service_name, String ip)throws java.rmi.RemoteException
    {
        String result;
        try{
            result = UG_Reference.require_application_service
                (service_name, ip);
        }catch(Exception e){
            throw new java.rmi.RemoteException();
        }
        return result;
    }

    public String require_system_service(String service_name, String
    ip)throws java.rmi.RemoteException
    {
        String result;
        try{
            result = UG_Reference.require_system_service
```

```
        (service_name, ip);
    }catch(Exception e){
        throw new java.rmi.RemoteException();
    }
    return result;
}

public boolean offer_service(String description_file, String
    ip)throws java.rmi.RemoteException
{
    boolean result;
    try{
        result = UG_Reference.offer_service(description_file, ip);
    }catch(Exception e){
        throw new java.rmi.RemoteException();
    }
    return result;
}

public UbiSys.Service[] get_all_services(String ip)
    throws java.rmi.RemoteException
{
    UbiSys.Service[] result;
    try{
        result = UG_Reference.get_all_services(ip);
    }catch(Exception e){
        throw new java.rmi.RemoteException();
    }
    return result;
}

public UbiSys.Service[] authenticate(UbiSys.Account account,
    String ip)throws java.rmi.RemoteException
{
    UbiSys.Service[] result;
    try{
        result = UG_Reference.authenticate(account, ip);
    }catch(Exception e){
        throw new java.rmi.RemoteException();
    }
    return result;
}
}
```

```
public void logout(String ip)throws java.rmi.RemoteException{
    try{
        UG_Reference.logout(ip);
    }catch(Exception e){
        throw new java.rmi.RemoteException();
    }
}

public void set_characteristics(java.lang.String ip,
    java.lang.String os)throws java.rmi.RemoteException
{
    try{
        UG_Reference.set_characteristics(ip, os);
    }catch(Exception e){
        throw new java.rmi.RemoteException();
    }
}

public String get_login(java.lang.String ip)
    throws java.rmi.RemoteException
{
    String login = null;
    try{
        login = UG_Reference.get_login(ip);
    }catch(Exception e){
        throw new java.rmi.RemoteException();
    }
    return login;
}
}
```

Una volta definite queste due classi viene generata l'interfaccia WSDL del servizio Web, mediante il tool Java2WSDL di axis:.

I parametri necessari sono:

- il nome del file di output: *ugw.wsdl*;
- l'URL del Web Service: *http://localhost:80/axis/services/UG*;
- il target namespace per la WSDL: *urn: UG*;
- il mapping fra il package java ed il namespace: *UG = urn: UG*
- l'interfaccia java del servizio: *UbiSys.UG\_Wrapping.UG.UG\_W*.

Il comando completo da eseguire risulta:

```
java org.apache.axis.wsdl.Java2WSDL
  -o ugw.wsdl
  -l"http://localhost:80/axis/services/UG"
  -n urn:UG
  -p"UG" urn:UG
  UbiSys.UG_Wrapping.UG.UG_W
```

Una volta eseguito il programma verrà generato il file *ugw.wsdl*.

Il prossimo passo è quello di generare, a partire dalla WSDL del servizio, il codice necessario per effettuare il deploy. Questa volta verrà adoperato il tool WSDL2Java.

I parametri necessari sono:

- la directory base di output: .;
- lo scope del servizio: *Session, Application, Request*;
- la generazione anche del codice server-side: *-s*;
- il package dove andrà il codice generato: *UbiSys.UG\_Wrapping.UG.ws*;
- il file dove reperire la wsdl: *ugw.wsdl* (ma potrebbe esse anche una URL).

Il comando completo risulta allora:

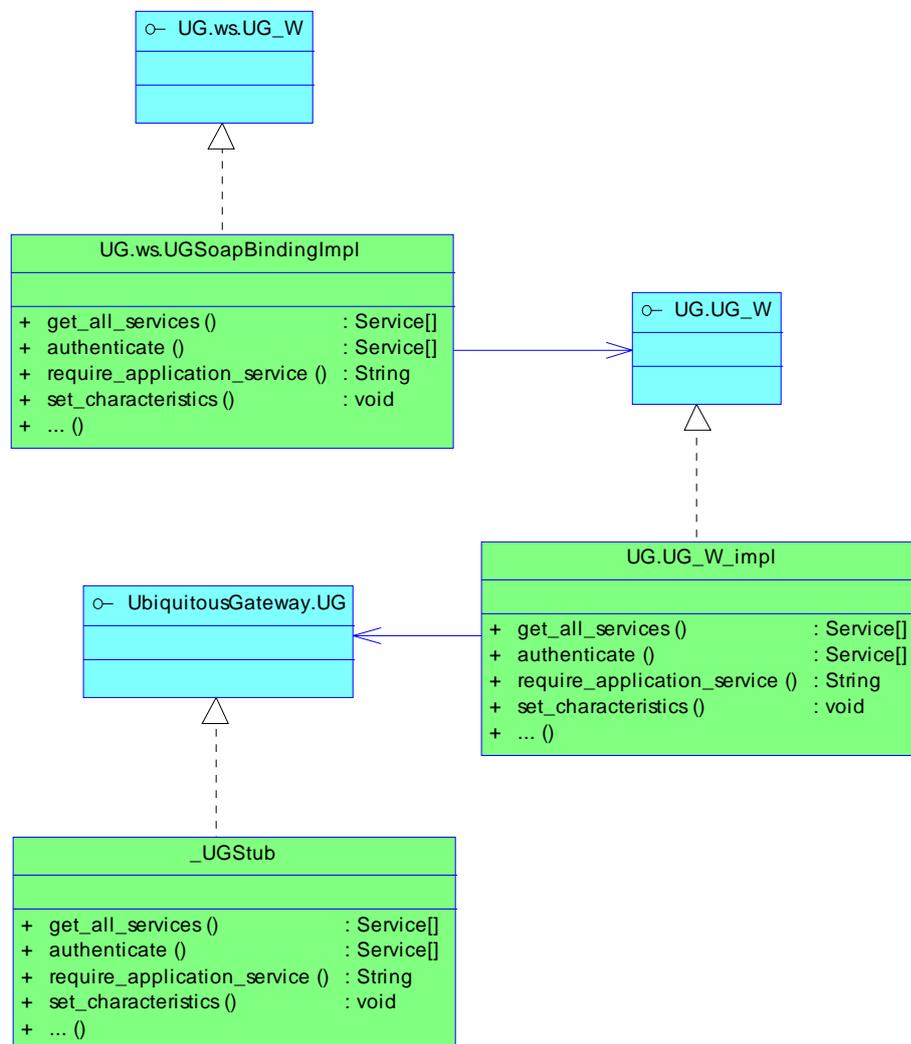
```
java org.apache.axis.wsdl.WSDL2Java
  -o .
  -d Session
  -s
  -p UbiSys.UG_Wrapping.UG.ws ugw.wsdl
```

Nel package *ws* saranno generati i file:

- *UGSoapBindingStub*: client-side stub;
- *UG\_W*: l'interfaccia remota;
- *UG\_WService*: interfaccia di servizio per il Web Service;
- *UG\_WServiceLocator*: implementa *UG\_WService*. E' una Helper Factory che fornisce l'accesso al servizio;
- *Account*: classe che ha lo scopo di fare il wrapping dell'omonima classe utilizzata dall'oggetto corba come tipo per parametri di in-outut;
- *Service*: classe che ha lo scopo di fare il wrapping dell'omonima classe utilizzata dall'oggetto corba come tipo per parametri di in-outut;

- *UGSoapBindingImpl*: la classe il cui codice implementa il Web Service. Sarà necessario apportare delle modifiche per fare in modo che i suoi metodi chiamino i corrispondenti della classe *UG\_W\_impl*. Inoltre si dovrà prevedere il mapping relativo alle classi *Account* e *Service*.
- *deploy.wsdd*: file contenente le informazioni necessarie per effettuare il deploy del Web Service sul sistema Axis;
- *undeploy.wsdd*: file necessario per effettuare l'undeploy del servizio da Axis.

Il seguente diagramma delle classi mostra come sia possibile ottenere un servizio web a partire da un oggetto Corba.



**Fig. IV.24** Creazione del servizio web a partire dal servizio Corba

Il passo successivo da eseguire è quello di generare degli archivi di file che saranno necessari affinché il servizio web funzioni in maniera corretta.

In particolare gli archivi da generare sono:

- *UG.jar*: contiene tutte le classi interne ai package *UG* e *UG.ws*, cioè quelle relative al Web Service;
- *UbiSys.jar*: contiene il server Corba invocato dal Web Service. In realtà contiene le classi relative all'intero sistema Corba.
- *sGn.jar*: contiene il servizio Corba *Ontology Service* invocato per la registrazione del device.

Tali file devono essere copiati nella cartella *webapps\axis\WEB-INF\lib* situata all'interno di *Tomcat*.

I comandi usati sono:

```
jar cvf UG.jar UbiSys/UG_Wrapping/UG/UG_W.class
      UbiSys/UG_Wrapping/UG/UG_W_impl.class
      UbiSys/UG_Wrapping/UG/ws/*.class
jar cvf UbiSys.jar UbiSys\*.*
jar cvf sGn.jar sGn\*.*
```

A questo punto è possibile procedere con la pubblicazione del servizio (*deploy*) utilizzando il comando:

```
org.apache.axis.client.AdminClient
-p 80
UbiSys/UG_Wrapping/UG/ws/deploy.wsdd
```

Quando viene eseguito quest'ultimo comando, *Tomcat* dovrà essere in esecuzione ed in ascolto sulla porta 80 (-p 80).

L'utente finale interagisce con Ubiquitous Gateway e, quindi, con l'intero sistema, mediante pagine *jsp*. Affinché queste possano utilizzare il servizio web è necessario copiare il file *UG.jar* anche nella cartella *WEB-INF\lib* situata all'interno della directory in cui sono locate le pagine in questione.

### IV.5.3 APPLICATION SERVICES MANAGER

Si riporta di seguito l'interfaccia IDL dell'*Application Services Manager*.

```
module UbiSys{
    struct Account {
        string login;
        string password;
    };

    struct Service {
        string name;
        string description;
        long rights;//Info passata a UG per distinguere i
                //servizi disponibili da quelli non disponibili
    };

    typedef sequence<Service> Services;
    typedef sequence<string> References;

    module ApplicationServiceManager{
        interface ASM {
            //Eccezioni generate
            exception ServiceNotAvailable{};
            exception ServicesListNotAvailable{};
            exception DeviceNotClassified{};
            exception ServiceNotRegistered{};

            string require_service(in string service_name,in string ip,
            in boolean flag) raises (ServiceNotAvailable);

            string require_service_reference(in string service_name,
            in string ip) raises (ServiceNotAvailable);

            string require_system_service(in string service_name)
            raises (ServiceNotAvailable);

            Services get_all_services(in string ip)
            raises (ServicesListNotAvailable, DeviceNotClassified);

            boolean register_user_service(in string description_file);

            void register_application_service(in string description_file,
            in string service_name, in long rights,
```

```
        in boolean active, in References reference)
            raises (ServiceNotRegistered);

Services authenticate(in Account account, in string ip)
    raises (ServicesListNotAvailable);

void release_service(in string service_name,
in string reference);

void logout(in string ip);

void implicit_logout(in string ip);

}; //End interface ASM
}; //End module ApplicationServiceManager
}; //End module UbiSys
```

Le strutture dati definite sono:

- *Account*. Si tratta di una struttura dati atta a definire l'account di un utente. E' definita in questo componente in quanto utilizzata come tipo di parametro in alcuni metodi.
- *Service*. Tale struttura dati definisce le caratteristiche di un servizio presente nell'ambiente. In particolare:
  - il campo *name* rappresenta il nome con cui il servizio è conosciuto nell'ambiente;
  - il campo *description* contiene una breve descrizione testuale riguardante il servizio;
  - il campo *rights* indica se il servizio è disponibile o meno.
- *Services*. Rappresenta una lista di servizi. In altri termini, un array di *Service*.
- *References*. Rappresenta una lista di riferimenti. In pratica si tratta di un array di stringhe.

I metodi implementati sono:

- *string require\_service (in string service\_name, in string ip, in boolean flag)*. Tale metodo riceve in ingresso il nome del servizio richiesto, l'ip del dispositivo da cui proviene la richiesta e un flag. Quest'ultimo dovrà assumere il valore "true" se la richiesta proviene da un utente e il valore "false" se la richiesta proviene da un altro servizio applicativo. La stringa restituita rappresenterà l'URL della HomePage del servizio nel primo caso, l'URL del servizio stesso nel secondo. Nel caso in cui la richiesta è fatta da un utente è necessario verificare che questo abbia i diritti opportuni per usufruire del servizio.
- *string require\_service\_reference (in string service\_name, in string ip)*. Tale metodo riceve in ingresso il nome del servizio richiesto e l'ip del dispositivo da cui proviene la richiesta. La stringa restituita rappresenta l'IOR dell'oggetto CORBA del servizio richiesto. All'interno di questo metodo si provvede anche all'aggiornamento della sessione associata all'utente, inserendo in essa l'IOR del servizio concesso all'utente.
- *string require\_system\_service (in string service\_name)*. Questo metodo viene invocato dai servizi applicativi che hanno bisogno di usufruire delle funzionalità offerte da servizi di sistema. Fornendo il nome del servizio di sistema desiderato viene restituito il riferimento. In pratica viene invocato il metodo opportuno esposto dal *System Services Manager*.
- *Services get\_all\_services (in string ip)*. Tale metodo riceve in ingresso l'ip del dispositivo da cui proviene la richiesta ed ha il compito di restituire la lista dei servizi applicativi disponibili per l'utente, in base ai suoi diritti, alla sua locazione e alle caratteristiche del dispositivo con cui è entrato nel sistema. La lista, in realtà, viene creata dall'*Ontology Service*.
- *boolean register\_user\_service (in string description\_file)*. Tale metodo riceve in ingresso la stringa di descrizione del servizio messo a disposizione dall'utente e ne permette la registrazione presso l'*Ontology*

*Service*. La variabile booleana in uscita indica l'avvenuta registrazione. Al momento però questa funzionalità non è implementata.

- *void register\_application\_service(in string description\_file, in string service\_name, in long rights, in boolean active, in References reference)*.

I parametri che il metodo riceve in ingresso sono:

- *description\_file*: stringa di descrizione del servizio;
- *service\_name*: nome con cui sarà registrato il servizio;
- *rights*: minimo livello di accesso che l'utente deve possedere per poter usufruire delle funzionalità del servizio.
- *active*: variabile booleana che indica se il servizio è già attivo, oppure dovrà essere attivato nel momento in cui viene richiesto per la prima volta.
- *reference*: array di stringhe. Se il servizio è su un'unica macchina nella prima locazione dell'array ci sarà l'IOR del servant, oppure l'IOR della factory se il servizio prevede più istanze del servant. Se il servizio è distribuito su più macchine l'array conterrà gli IOR dei diversi servant, oppure delle diverse factory.

Tale metodo viene invocato nel momento in cui si attiva un servizio applicativo e viene chiamato dalla classe *ApplicationServiceProvider*. Il metodo provvede alla registrazione del servizio presso l'*Ontology Service* e in più memorizza le informazioni relative al servizio in una struttura dati interna al manager.

- *Services authenticate (in Account account, in string ip)*. Questo metodo riceve in ingresso l'account dell'utente, cioè login e password che egli ha inserito, e restituisce la lista dei servizi applicativi disponibili in base alle caratteristiche dell'utente e quelle del suo dispositivo.
- *void logout (in string ip)*. Tale metodo riceve in ingresso l'ip del dispositivo da cui proviene la richiesta. Viene invocato nel momento in cui un utente diverso dall'amministratore comunica l'abbandono del

sistema. A questo punto viene richiesta la chiusura della sessione e si deallocano le risorse relative alle applicazioni in uso.

- *void implicit\_logout (in string ip)*. Tale metodo riceve in ingresso l'ip del dispositivo da cui proviene la richiesta. Viene invocato nel momento in cui un utente abbandona il sistema senza darne comunicazione. In questo caso si richiede al *Session Manager* la lista dei servizi in uso nella relativa sessione e si richiede la deallocazione delle risorse relative a tutti i servizi di tipo interattivo. Se l'utente ha lasciato in sospeso dei servizi di tipo batch la sessione non sarà chiusa.
- *void release\_service (in string service\_name, in string reference)*. I parametri che il metodo riceve in ingresso sono il nome del servizio da rilasciare e il suo IOR. In questo modo sarà possibile deallocare un servizio applicativo e le relative risorse. Il servizio in questione può essere sia batch che interattivo. Nel primo caso sarà il servizio stesso ad invocare il metodo nel momento in cui termina l'elaborazione per cui è stato richiesto. Nel secondo caso sarà il manager a richiederlo nel momento in cui l'utente va via senza chiudere la sessione. Ad ogni invocazione si comunica al *Session Manager* che il servizio non è più attivo e se tutti i servizi utilizzati nella sessione relativa sono inattivi questa sarà chiusa.

#### IV.5.4 SYSTEM SERVICES MANAGER

Si riporta di seguito l'interfaccia IDL del System Services manager.

```
module UbiSys {
    struct Service {
        string name;
        string description;
        long rights;
    };
    struct Account {
        string login;
        string password;
    };
};
```

```
typedef sequence<Service> Services;

module SystemServiceManager {
    interface SSM {
        //Eccezioni sollevate
        exception AccountNotFound {};
        exception SystemServiceNotAvailable {};
        exception SystemServiceNotRegistered {};
        exception ServicesListNotAvailable {};

string require_system_service(in string service_name,
    in string ip)raises (SystemServiceNotAvailable);

string get_system_service(in string service_name)
    raises (SystemServiceNotAvailable);

void register_system_service(in string description_file,
    in References references, in string service_name,
    in boolean active) raises (SystemServiceNotRegistered);

Services authenticate(in Account account, in string ip)
    raises (AccountNotFound,ServicesListNotAvailable);

void logout(in string ip);

    };//End interface SSM
};//End module SystemServiceManager
};//End module UbiSys
```

Le strutture dati definite sono:

- *Account*. Si tratta di una struttura dati atta a definire l'account di un utente. E' definita in questo componente in quanto utilizzata come tipo di parametro in alcuni metodi.
- *Service*. Tale struttura dati definisce le caratteristiche di un servizio presente nell'ambiente. In particolare:
  - il campo *name* rappresenta il nome con cui il servizio è conosciuto nell'ambiente;

- il campo *description* contiene una breve descrizione testuale riguardante il servizio;
  - il campo *rights* indica se il servizio è disponibile o meno.
- *Services*. Rappresenta una lista di servizi. In altri termini, un array di *Service*.

I metodi implementati sono:

- *string require\_system\_service(in string service\_name, in string ip)*. I parametri di ingresso sono il nome del servizio e l'ip del dispositivo da cui proviene la richiesta. Tale metodo viene invocato nel momento in cui l'amministratore vuole usufruire delle funzionalità esposte dai servizi di sistema. Esso restituirà l'URL della HomePage del servizio solo se verifica che colui che ha effettuato la richiesta ha i diritti opportuni.
- *string get\_system\_service(in string service\_name)*. Il parametro di ingresso è il nome del servizio. Tale metodo viene invocato quando a voler utilizzare il servizio di sistema sono altri servizi dello stesso genere.
- *void register\_system\_service(in string description\_file, in References references, in string service\_name, in boolean active)*. Questo metodo permette di registrare un servizio di sistema. Viene chiamato dalla classe *SystemServiceProvider*. I parametri di ingresso del metodo in esame sono:
  - *description\_file*: stringa che contiene la descrizione del servizio;
  - *references*: array di stringhe. Se il servizio è su un'unica macchina nella prima locazione dell'array ci sarà l'IOR del servant, oppure l'IOR della factory se il servizio prevede più istanze del servant. Se il servizio è distribuito su più macchine l'array conterrà gli IOR dei diversi servant, oppure delle diverse factory;
  - *service\_name*: nome con cui sarà registrato il servizio;
  - *active*: indica se il servizio è attivo o meno all'atto della registrazione.

Il metodo provvede alla registrazione del servizio presso l'*Ontology Service* e in più memorizza le informazioni relative al servizio in una struttura dati interna al manager.

- *Services authenticate(in Account account, in string ip)*. Questo metodo riceve in ingresso l'account dell'utente, cioè login e password che egli ha inserito, e restituisce sia la lista dei servizi applicativi attivi nel sistema, sia la lista dei servizi di sistema. Esso sarà invocato dall'*Ubiquitous Gateway* solo quando ad autenticarsi è l'amministratore.
- *void logout(in string ip)*. Tale metodo riceve in ingresso l'ip del dispositivo da cui proviene la richiesta. Esso viene invocato nel momento in cui l'amministratore comunica l'abbandono del sistema. A questo punto viene richiesta la chiusura della sessione relativa.

## V. SERVIZI APPLICATIVI

### V.1 INTRODUZIONE

Durante il lavoro di tesi svolto, il sistema *UbiSystem* è stato riprogettato e sono stati realizzati componenti di gestione prima inesistenti. Non per questo il lavoro già esistente è stato tralasciato.

Lo scopo di questo capitolo è di mostrare le modifiche apportate ai servizi applicativi esistenti per la loro integrazione nella nuova architettura.

I servizi considerati sono:

- Mp3 JukeBox;
- PdfViewer;
- Local Print Service;
- Musica d'ambiente;
- Streaming Locale.

Per quest'ultimo non è stata necessaria alcuna modifica in quanto è stato oggetto di una tesi di laurea sviluppata in contemporanea alla presente.

I servizi sono stati implementati con tecnologia CORBA e poi, generato l'apposito wrapper, sono stati esposti all'interno dell'ambiente sottoforma di servizi WEB.

## V.2 DESCRIZIONE FUNZIONALE DEI SERVIZI

Lo scopo di questo paragrafo è quello di descrivere le funzionalità esposte dai servizi applicativi presenti in *UbiSystem*.

### V.2.1 MP3 JUKEBOX

L'Mp3 JukeBox, in esecuzione nell'ambiente, consente agli utenti di creare ed ascoltare playlist di brani musicali.

Il servizio è stato implementato con tecnologia Corba e poi, generato l'apposito wrapper, è stato esposto all'interno dell'ambiente sottoforma di servizio WEB.

Di seguito è riportata l'interfaccia IDL del servizio di *Mp3 JukeBox*.

Attraverso il metodo *getList()* è possibile conoscere la lista delle canzoni attualmente disponibili sul server; con il metodo *setPlayList()* viene definita la sequenza di canzoni da suonare; tale elenco potrà essere recuperato tramite il metodo *getPlayList()*. Nel caso un utente volesse ascoltare una propria canzone può effettuare l'upload del file musicale e poi incorporarlo in una playlist. I restanti metodi sono i tradizionali controlli di esecuzioni dei brani (*play, stop, next, increase volume, decrease volume, etc.*).

```
/* IDL Mp3 Server */
module Mp3 {
    typedef sequence<string> mp3List;
    interface Mp3Server {
        // Controlli esecuzione brano
        void play ();
        void pause();
        void rewind();
        void stop();
        void next();
        void previous();
        // Controlli volume
        void increase();
        void decrease();
        void mute();
        // Funzione di upload file sul server
        void upload (in string filename);
        // Funzioni di gestione playlist
        mp3List getPlayList();
    };
};
```

```

        void setPlayList(in mp3List list);
        mp3List getList();
    };
};

```

### V.2.2 PDFVIEWER

Il PdfViewer è un servizio in esecuzione su di un nodo della rete presso il quale è, presumibilmente, disponibile un video proiettore. Immaginiamo che nell'ambiente si debba tenere una presentazione per una conferenza. Il relatore attraverso, il suo palmare, una volta avuto accesso al servizio, può inviare il file pdf da visualizzare e comandare la presentazione attraverso il proprio dispositivo.

Il servizio è stato realizzato mediante le librerie reperibili presso il sito della Adobe e contenute nel package `adobe.jar`; le funzionalità messe a disposizione sono quelle di un comune visualizzatore pdf:

- Apertura e chiusura del file (*view(fileURL)* e *close()*);
- Metodi per lo scorrimento delle pagine (*firstpage()*, *lastpage()*, *pageup()*, *pagedown()*, *lineup()*, *linedown()*);
- Metodi per l'impostazione della visualizzazione delle pagine (*fitwidth()*, *fitheight()*, *singlepage()*);
- Metodi per l'impostazione della presentazione a pieno schermo (*fullscreen()*, *fromfullscreen()*).

Naturalmente, la prima operazione da effettuare è quella di avviare la presentazione attraverso il metodo *view(fileURL)*: il metodo ha per parametro l'URL del file pdf da visualizzare. Il file viene trasferito sulla macchina dove risiede il servizio con un download mediante protocollo HTTP.

Si riporta di seguito l'interfaccia IDL del servizio in esame.

```

/* IDL Pdf Viewer */
module pdfviewer{
    module Pdf {
        typedef sequence<string> pdfList;
        interface PdfViewer {

            void viewLocalFile(in string urlPdfFile);
            void viewRemoteFile(in string filename);
            void setDir(in string dir);
        }
    }
}

```

```
void nextpage();
void prevpage();
void pageup();
void pagedown();
void close();
pdfList getList();
void firstpage();
void lastpage ();
void fitpage ();
void fitheight ();
void fitwidth ();
void fullscreen ();
void fromfullscreen ();
void linedown ();
void lineup ();
void singlepage ();
};
};
};
```

### V.2.3 LOCAL PRINT SERVICE

Il servizio di stampa locale, in esecuzione nell'ambiente, consente agli utenti, che si sono agganciati alla rete, di scegliere i documenti da stampare, settare alcune preferenze di stampa, in particolare stampa a colori o monocromatica.

Il servizio avvia in automatico la stampa presso la stampante nella stanza dell'utente oppure chiede all'utente di scegliere esplicitamente la stanza ove gli sia più comodo prelevare i documenti. Quest'ultimo scenario si verifica quando nessuna stampante è presente nella stanza ove si trova l'utente o quando le stampanti presenti nella stanza dell'utente non sono in grado di soddisfare le preferenze di stampa dell'utente.

Il servizio necessita di informazioni di contesto per stabilire quali sono le stampanti a cui l'utente possa accedere più agevolmente.

Il servizio non necessita di un monitoraggio continuo delle informazioni di contesto. La richiesta delle informazioni di contesto avviene solo quando necessario, ovvero quando l'utente fa esplicita richiesta di stampa. Il servizio comunica sempre in modo sincrono con *UbiSystem* attraverso le apposite funzionalità offerte dal Context Service.

Il servizio è stato implementato con tecnologia CORBA e poi, generato l'apposito wrapper, giacché il servizio espone funzionalità che devono essere rese accessibili dall'esterno, è stato esposto all'interno dell'ambiente sottoforma di servizio WEB.

Di seguito è riportata l'interfaccia IDL del servizio. Il metodo *print* è il metodo attraverso il quale il servizio avvia la stampa per l'utente, identificato dall'IP del dispositivo con cui si è agganciato alla rete, che ne fa richiesta. Attraverso l'invocazione del metodo *printwith* il servizio invia invece la stampa su una specifica stampante.

I metodi *setFile* e *setColor* sono metodi attraverso i quali l'utente specifica il documento da stampare ed il tipo di stampa inteso come a colori o monocromatica. Il metodo *getStatus* infine consente di ricavare informazioni di controllo sul servizio.

```
/* IDL localPrintService */
module Print {
    interface StampaService {
        void print (in string userIP);
        void printwith (in string printerName);
        long getStatus();
        void setFile (in string filename);
        void setColor (in boolean colorFlag);
    };
};
```

#### V.2.4 MUSICA D'AMBIENTE

Il servizio di musica di ambiente è un servizio sviluppato con lo scopo di mettere in evidenza come UbiSystem sia effettivamente sensibile ai cambiamenti di contesto.

Il servizio provvede alla gestione della musica di sottofondo di una determinata stanza a tema.

Si immagini a riguardo, in un museo, una stanza dedicata alla cultura egiziana, all'interno della quale, se presenti visitatori, viene eseguita una specifica musica a tema.

Nella concreta implementazione del servizio: se nella stanza è presente almeno un visitatore il servizio provvede alla esecuzione della musica di sottofondo mentre provvede alla sua interruzione quando la stanza si svuota. La musica di sottofondo verrà poi ripresa quando verrà notificato al servizio la presenza di un nuovo

visitatore. Il servizio inoltre ha una funzionalità aggiuntiva: ad ogni nuovo individuo che giunge nella stanza da il benvenuto tramite un apposito messaggio vocale.

Affinché il servizio possa offrire tali funzionalità è necessario che invochi l'apposito metodo del *Context Service* affinché quest'ultimo lo tenga continuamente aggiornato sugli individui presenti nella stanza.

Il servizio di musica di ambiente, pertanto, a differenza di quanto avviene con il servizio di stampa in locale, utilizza e sperimenta le funzionalità che richiedono un monitoraggio continuo sulle informazioni di contesto che *UbiSystem* fornisce mediante il *Context Service*.

Il servizio di musica di ambiente non è esposto come Web Service in quanto non espone metodi che devono essere resi accessibili dall'esterno.

### V.3 IMPLEMENTAZIONE DEI SERVIZI

I servizi applicativi sono stati implementati come oggetti Corba ed in seguito sono stati wrappati ed esposti nell'ambiente come servizi web.

Affinché un utente possa accedere alle funzionalità da essi esposte è necessario che i servizi si siano registrati presso il manager dei servizi applicativi.

Di seguito, si mostra il procedimento seguito per la realizzazione di un generico servizio applicativo evidenziando le modifiche apportate al codice rispetto all'implementazione precedente.

#### V.3.1 IMPLEMENTAZIONE DI UN OGGETTO CORBA

Analogamente a come fatto nel capitolo precedente, si considera la realizzazione di un generico servizio applicativo: *Service*.

Il primo passo da fare è la scrittura dell'interfaccia IDL salvata nel file: "*Service.idl*". Questa è data in pasto al compilatore *idl-java* che genera tre gruppi di classi:

- Classi lato Client: *\_ServiceStub.java*, *Service.java*
- Classi Lato Server: *ServicePOA.java*, *ServicePOATie.java*, *ServiceOperations.java*
- Classi di supporto: *ServiceHelper.java*, *ServiceHolder.java*.

Successivamente si dovrà scrivere il codice relativo alle classi *Server*, *Impl* e *Client*, che hanno struttura simile a quelle relative all'oggetto *Manager* analizzato nel capitolo precedente.

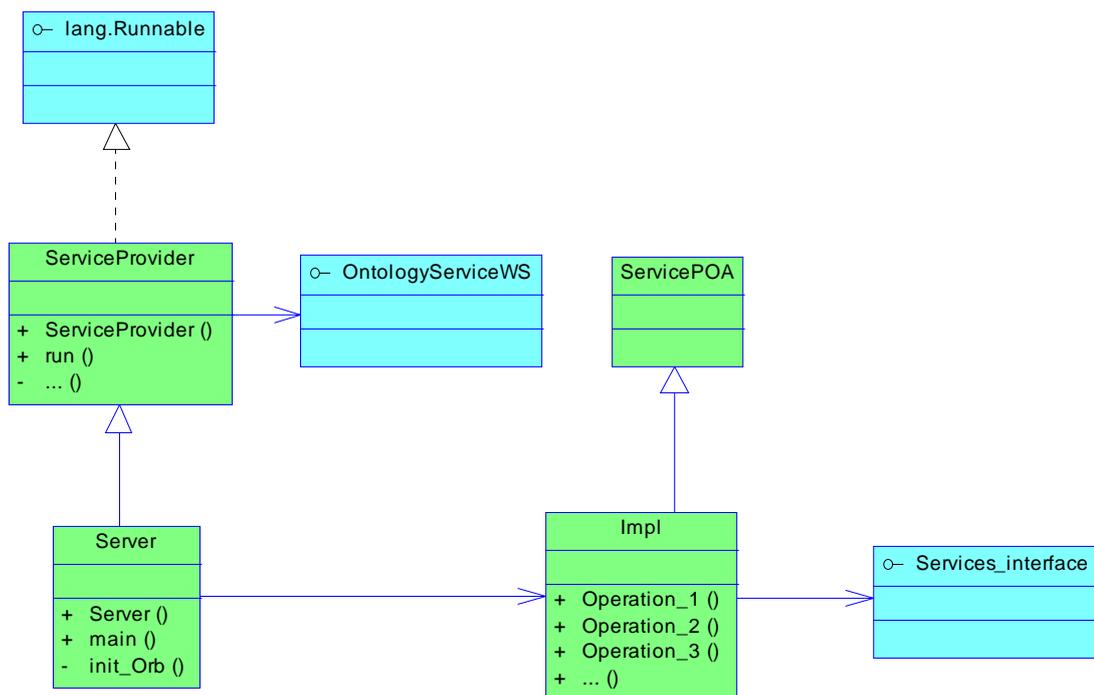
### V.3.1.1 REGISTRAZIONE DI UN SERVIZIO APPLICATIVO

Una delle modifiche apportate consiste nella variazione del meccanismo di registrazione di un servizio e di pubblicazione dell'IOR.

Nella precedente implementazione non erano presenti i manager dei servizi, quindi la registrazione veniva fatta direttamente presso l'*Ontology Service*, esposto come servizio web, per mezzo della classe *ServiceProvider* estesa dal server.

Il server pubblicava il riferimento dell'oggetto presso il *Naming Service*, quindi il client lo doveva leggere da questo.

Il diagramma delle classi del servizio prima delle modifiche risulta:



**Fig. V.1 Diagramma delle classi di un servizio applicativo**

In questo caso particolare la pubblicazione dell'IOR avviene mediante la registrazione del servizio.

Il codice che implementa il server è riportato di seguito.

```
public class Server extends ServiceProvider {
    public Server(args) {
        super(); //Chiamata al costruttore della ServiceProvider
        initOrb(args);
    }
    public static void main(String[] args) {
        new Server(args);
    }
    private void initOrb (String args[]){
        Service service;
        try{
            // initializza l'ORB.
            ORB orb = ORB.init( args, null );

            // ottiene il riferimento al Naming Server
            NamingContextExt nc = NamingContextExtHelper.narrow(
            orb.resolve_initial_references("NameService"));

            //Ottiene rif POA
            POA rootPOA = POAHelper.narrow
                (orb.resolve_initial_references("RootPOA"));
            //Attiva POA
            rootPOA.the_POAManager().activate();

            //Crea l'obj servant
            Impl servant=new Impl();

            //registra obj nel POA
            service = servant._this(orb);

            //Sul naming va il reference del servant
            nc.bind(nc.to_name("Service_name"), service);

            //mette l'appl servente in stato di attesa
            orb.run();

        } catch (Exception e){
            e.printStackTrace();
        }
    }
}
```

Il codice della classe `ServiceProvider` è riportato di seguito:

```
public class ServiceProvider implements Runnable {

    private ontoWS.ws.OntologyServerInterface server = null;
    private String id;
    private int delay = 6000;
    private Thread thread = null;
    private String file = null;
    private String ontologyUrl = null;
    private String xmlDescription = "";

    public ServiceProvider() {
        thread = new Thread(this);
        thread.start();
    }

    public static void main(String [] args) throws Exception {
        new ServiceProvider();
    }

    public void run() {
        // Legge i parametri di configurazione
        readConfig();

        // Make a service
        ontoWS.ws.OntologyServerInterfaceService service =
        new ontoWS.ws.OntologyServerInterfaceServiceLocator();

        ((ontoWS.ws.OntologyServerInterfaceServiceLocator)
        service).setontologyserverAddress(ontologyUrl);

        // Now use the service to get a stub to the service
        try {
            server = service.getontologyserver();
        } catch (Exception e) {}

        // Registra il servizio presso l'Ontology Server
        if (buildDescription())
            subscribeService();
    }
}
```

```
private boolean buildDescription() {
    boolean res = false;
    try {
        BufferedReader in = new BufferedReader
            (new FileReader(file));
        xmlDescription = "";
        String buf = null;
        while ( (buf = in.readLine()) != null)
            xmlDescription = xmlDescription + buf + "\n";
        res = true;
    } catch (Exception e) {
        System.out.println ("Impossibile trovare il file " +
file);
    }
    return res;
}

private void subscribeService() {
    boolean go = true;
    while (go) {
        try {
            System.out.print("Tentativo di registrazione del
servizio... ");
            if (server.entityConnection(xmlDescription))
                System.out.println("\nServizio registrato con
successo...");
            else
                System.out.println("\nRegistrazione fallita: verificare
la descrizione del servizio...");
            go = false;
        } catch (Exception e) {
            System.out.println("non riuscito.\nProblemi di
connessione con l'Ontology Server");
        }
        try {
            thread.sleep(delay);
        } catch (Exception e) {}
    }
}

public void readConfig() {
    Hashtable config = new Hashtable();
}
```

```
String[] xml = {"OntologyServerUrl", "description",
"delay"};
String xmlFile = "ServiceConfig.xml";

DOMParser parser = new DOMParser();
try {
    parser.parse(xmlFile);
} catch (Exception se) {
    System.err.println ("File Config.xml not found or
malformed");
    return;
}
Document document = parser.getDocument();
Element element = document.getDocumentElement();
for (int i=0; i<xml.length; i++) {
    NodeList nodelist=element.getElementsByTagName(xml[i]);
    if (nodelist.getLength()==1)
        config.put(xml[i], (String)
nodelist.item(0).getFirstChild().getNodeValue());
    else {
        System.err.println("Parsing error: " + xml[i] + "
not found");
    }
}

ontologyUrl = (String) config.get("OntologyServerUrl");
file = (String) config.get("description");
try {
    delay = Integer.parseInt((String) config.get("delay"));
} catch (Exception e) {
    delay = 60000;
}
}
```

Il file config.xml utilizzato in questa classe ha struttura:

```
<?xml version="1.0"?>
<config>
```

```
<OntologyServerUrl>
    http://localhost/axis/services/ontologyserver
</OntologyServerUrl>
<description>
    Service.daml
</description>
<delay>
    80000
</delay>
</config>
```

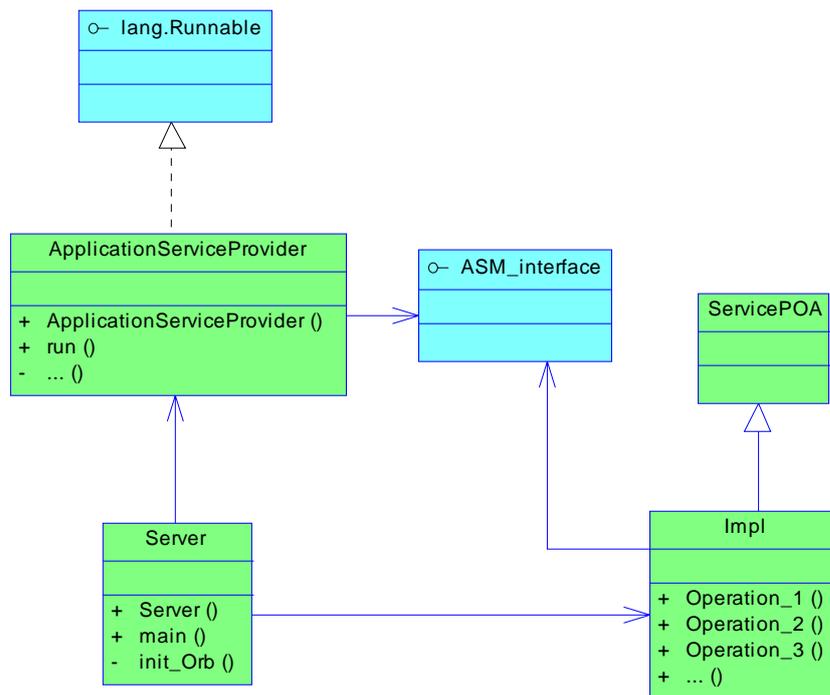
Nel file sono indicati: l'URL presso il quale contattare l'*Ontology Service* (<http://www.ubinet.it/axis/services/ontologyserver>); il nome del file dove è contenuta la descrizione del servizio (*Service.daml*); l'intervallo di tempo, espresso in millisecondi, ogni quanto riavviare un nuovo tentativo di registrazione del servizio.

Il file *Service.daml* è scritto in linguaggio DAML+OIL e contiene le informazioni riguardanti il servizio, oltre alle precondizioni sia hardware che software che il dispositivo utente deve rispettare per usufruirne.

Grazie alla presenza del manager dei servizi applicativi la fase di registrazione deve essere modificata. Infatti, la richiesta di registrazione dovrà essere inoltrata all'ASM che provvederà a memorizzare il riferimento al servizio e a richiedere all'*Ontology Service* la validazione della descrizione del servizio.

Risulta, quindi, evidente che la registrazione deve avvenire dopo l'attivazione del servant, ciò significa che il costruttore della classe che si occupa della registrazione dovrà essere invocato in un secondo momento. La classe *Server.java* non estende la classe *ServiceProvider*, ma utilizza la classe *ApplicationServiceProvider*, che ha un ruolo analogo.

Il diagramma delle classi dopo la modifica risulta:



**Fig. V.2 Diagramma delle classi dopo la modifica**

Il codice del server diventa:

```

public class Server {

    public Server(args) {
        initOrb(args);
    }

    public static void main(String[] args) {
        new Server(args);
    }

    private void initOrb (String args[]){
        Service service;
        try{
            // initializza l'ORB.
            ORB orb = ORB.init( args, null );

            // ottiene il riferimento al Naming Server
            NamingContextExt nc = NamingContextExtHelper.narrow(
                orb.resolve_initial_references("NameService"));

            //Ottiene rif POA

```

```

        POA rootPOA = POAHelper.narrow
            (orb.resolve_initial_references("RootPOA"));

        //Attiva POA
        rootPOA.the_POAManager().activate();

        //Crea l'obj servant
        Impl servant=new Impl();

        //registra obj nel POA
        service = servant._this(orb);

        //legge il riferimento dell'ASM dal naming service
        ASM_Reference=ASMHelper.narrow
            (nc.resolve(nc.to_name("ASM")));

        Reference = orb.object_to_string(pdf);
        String[] ref = new String[1];
        ref[0] = Reference;

        new ApplicationServiceProvider(ASM_Reference, ref,
"ServiceConfig.xml");

        //mette l'appl servente in stato di attesa
        orb.run();

    } catch (Exception e){
        e.printStackTrace();
    }
}
}

```

Di seguito si riporta il codice della classe ApplicationServiceProvider:

```

public class ApplicationServiceProvider implements Runnable {

    //private ontoWS.ws.OntologyServerInterface server = null;
    private String id = new String();
    private int delay = 6000;
    private Thread thread = null;
    private String file = new String();
    private String xmlDescription = new String();
    private ASM MyASM_Reference;
}

```

```
private String[] references;
private String service_name = new String();
private String xmlFile = new String();
private int rights = 0;

public ApplicationServiceProvider() {
    thread = new Thread(this);
    thread.start();
}

    public ApplicationServiceProvider(ASM ASM_Reference,
String[] references, String xmlFile) {
    thread = new Thread(this);
    thread.start();
    MyASM_Reference = ASM_Reference;
    this.references = new String[references.length];
    this.references = references;
    this.xmlFile = xmlFile;
}
public static void main(String [] args) throws Exception {
    new ApplicationServiceProvider();
}

public void run() {
    // Legge i parametri di configurazione
    readConfig();

    // Registra il servizio presso l'SSM
    if (buildDescription())
        subscribeService();
}

private boolean buildDescription() {
    boolean res = false;
    try {
        BufferedReader in = new BufferedReader(new
FileReader(file));
        xmlDescription = "";
        String buf = null;
        while ( (buf = in.readLine()) != null)
            xmlDescription = xmlDescription + buf + "\n";
        res = true;
    }
```

```
        } catch (Exception e) {
            System.out.println ("Impossibile trovare il file "
+ file);
        }
        return res;
    }

    private void subscribeService() {
        boolean go = true;
        while (go) {
            try {
                System.out.print("Tentativo di registrazione del
servizio... ");
                MyASM_Reference.register_application_service(xmlDescription,
service_name, rights, true, references) ;
                System.out.print("Registrazione del servizio
terminata!");
                go = false;
            } catch (UbiSys.ApplicationServiceManager.
                ASMPackage.ServiceNotRegistered e){
                System.out.println("Service not registered");
            }
            catch (Exception e) {
                System.out.println("non riuscito.\nProblemi
di connessione con l'Ontology Server");
            }
            try {
                thread.sleep(delay);
            } catch (Exception e) {}
        }
    }

    public void readConfig() {
        Hashtable config = new Hashtable();
        String[] xml = {"name", "description", "delay",
"rights"};
        System.err.println (xmlFile);
        DOMParser parser = new DOMParser();
        try {
            parser.parse(xmlFile);
        } catch (Exception se) {
            System.err.println ("File Config.xml not found or
```

```

        malformed");
        return;
    }
    Document document = parser.getDocument();
    Element element = document.getDocumentElement();
    for (int i=0; i<xml.length; i++) {
        NodeList nodelist =
            element.getElementsByTagName(xml[i]);
        if (nodelist.getLength()==1)
            config.put(xml[i], (String)
                nodelist.item(0).getFirstChild().getNodeValue());
        else {
            System.err.println("Parsing error: " + xml[i] + "
not found");
        }
    }
    service_name = (String) config.get("name");
    file = (String) config.get("description");
    rights=Integer.parseInt((String)config.get("rights"));
    try {
        delay = Integer.parseInt((String)config.get("delay"));
    } catch (Exception e) {
        delay = 60000;
    }
}
}

```

Anche il file di configurazione ha subito delle modifiche. Esso non riporta più l'URL dell'*Ontology Service*, ma riporta il nome simbolico del servizio con cui esso sarà registrato nell'ambiente e il livello di accesso necessario per utilizzarlo.

Il file `config.xml` utilizzato in questa classe ha struttura:

```

<?xml version="1.0"?>
<config>
  <name>
    Service_name
  </name>
  <description>
    Service.daml
  </description>
  <delay>
    80000

```

```

</delay>
<rights>
    0
</rights>
</config>

```

La parte di codice modificata è evidenziata in neretto.

Per chiarezza si riporta, in forma schematica la descrizione del servizio applicativo Mp3 JukeBox:

DESCRIZIONE DEL SERVIZIO MP3 JUKEBOX				
<b>ResourceName</b>		Mp3Player		
<b>ResourceTextDescr</b>		Questa risorsa consente di eseguire e controllare un file mp3 su una macchina remota.		
<b>ResourceType</b>		User		
<b>ResourcePageUrl</b>		http://192.168.0.8/servizi/MP3Player.jsp		
<b>ResourceUrl</b>		http://192.168.0.133/axis/services/Mp3Server		
<b>ResourceInteractive</b>		true		
<b>ResourceSingleInstance</b>		true		
<b>ResourceCommand</b>		Start		
<b>offeredBy</b>		Mach_08		
<b>Resource Profile</b>	<b>Hw Precondition</b>	<b>Audio-output</b>		true
		<b>Audio-input</b>		False
		<b>Video-input</b>		false
		<b>Connection Description</b>	<b>bandwidth</b>	100
			<b>bandwidthunit</b>	Mbit/s
		<b>Resolution Description</b>	<b>resolutionwidth</b>	240
			<b>resolutionheight</b>	320
			<b>resolutionunit</b>	pixel
			<b>resolutionbpp</b>	32
		<b>MemoryType Description</b>	<b>resolutiongraphics</b>	true
			<b>memoryamount</b>	6
			<b>memoryunit</b>	Gb
	<b>memoryusagetype</b>	Storage		

		<b>Cpu</b>	<b>cpuspeed</b>	1.33
		<b>Description</b>	<b>cpuunit</b>	MHz
	<b>Sw Preconditio n</b>	<b>Os</b>	<b>Name</b>	Windows
			<b>Vendor</b>	Microsoft
			<b>Version</b>	4.0
		<b>application</b>	<b>Name</b>	Windows Media Player
			<b>Vendor</b>	Microsoft
			<b>Version</b>	8.0

### V.3.2 WRAPPING DI UN SERVIZIO

Per l'esposizione di un servizio Corba come Web Service si segue lo stesso procedimento impiegato per wrappare l'*Ubiquitous Gateway*.

La classe *W\_Impl.java* sarà client del servizio Corba, quindi all'interno del suo costruttore deve eseguire le operazioni:

- *Inizializzazione dell'ORB.* Così come nel server, anche nel client ogni operazione CORBA deve essere preceduta dall'inizializzazione dell'oggetto ORB.
- *Dichiarazione variabile per l'oggetto.* Le invocazioni all'oggetto remoto avvengono in forma di invocazioni di metodo su un'istanza di una classe Java avente lo stesso nome dell'oggetto CORBA da invocare. Prima di tutto occorre quindi dichiarare una variabile che in seguito conterrà il riferimento remoto all'oggetto.
- *Reperimento dell'Object Reference.* L'OR dell'oggetto server desiderato viene reperito da dove era stato preventivamente memorizzato dal processo server (ad es. Naming Service o file system).
- *Creazione del riferimento remoto all'oggetto.* La variabile precedentemente dichiarata viene inizializzata con l'Object Reference reperito. Viene instaurata la connessione remota all'oggetto e si possono effettuare le richieste.

Prima delle modifiche apportate, l'IOR veniva richiesto al *Naming Service*.

Di seguito è riportato il codice Java del costruttore della classe *W\_Impl.java*:

```
public W_Impl() {
```

```

Service Reference = null;
try {
    //Inizializzazione ORB
    org.omg.CORBA.ORB orb=org.omg.CORBA.ORB.init(args,null);

    // ottiene il riferimento al Naming Server
    NamingContextExt nc = NamingContextExtHelper.narrow(
    orb.resolve_initial_references("NameService"));

    //legge il riferimento dell'implementazione del servant
    //dal naming service
    Reference = ServiceHelper.narrow
        (nc.resolve(nc.to_name("Service_name")));

    //ora è possibile invocare direttamente i metodi
}
catch (Exception e){
    e.printStackTrace();
}
}

```

Dopo le modifiche nella procedura di registrazione di un servizio applicativo, anche il reperimento dell'IOR deve avvenire in modo diverso.

L'ASM espone un metodo per richiedere l'IOR di un servizio applicativo.

Il codice del costruttore modificato risulta:

```

public W_Impl() {
    Service Reference = null;
    try {
        //Inizializzazione ORB
        org.omg.CORBA.ORB orb=org.omg.CORBA.ORB.init(args,null);

        // ottiene il riferimento al Naming Server
        NamingContextExt nc = NamingContextExtHelper.narrow(
        orb.resolve_initial_references("NameService"));

        ASM_Reference = ASMHelper.narrow
            (nc.resolve(nc.to_name("ASM")));
        String server_ref = ASM_Reference.
            require_service_reference("Service_name",ip);
        org.omg.CORBA.Object ob = orb.string_to_object(server_ref);
        Reference = ReferenceHelper.narrow(ob);
    }
}

```

```
        //ora è possibile invocare direttamente i metodi
    }
    catch (Exception e){
        e.printStackTrace();
    }
}
```

Le modifiche apportate sono evidenziate in grassetto.

In questo modo i servizi esistenti possono essere completamente integrati con la nuova versione di *UbiSystem*.

## VI. ESEMPIO D'USO

Dopo aver analizzato i vari componenti di *UbiSystem* e la loro interazione, risulta interessante analizzare il comportamento del sistema dal punto di vista dell'utente.

### VI.1 LO SCENARIO IN CUI OPERANO I SERVIZI

Lo scenario in cui i servizi realmente operano è costituito da due stanze: Stanza della Segreteria e Stanza di Motion Capture.

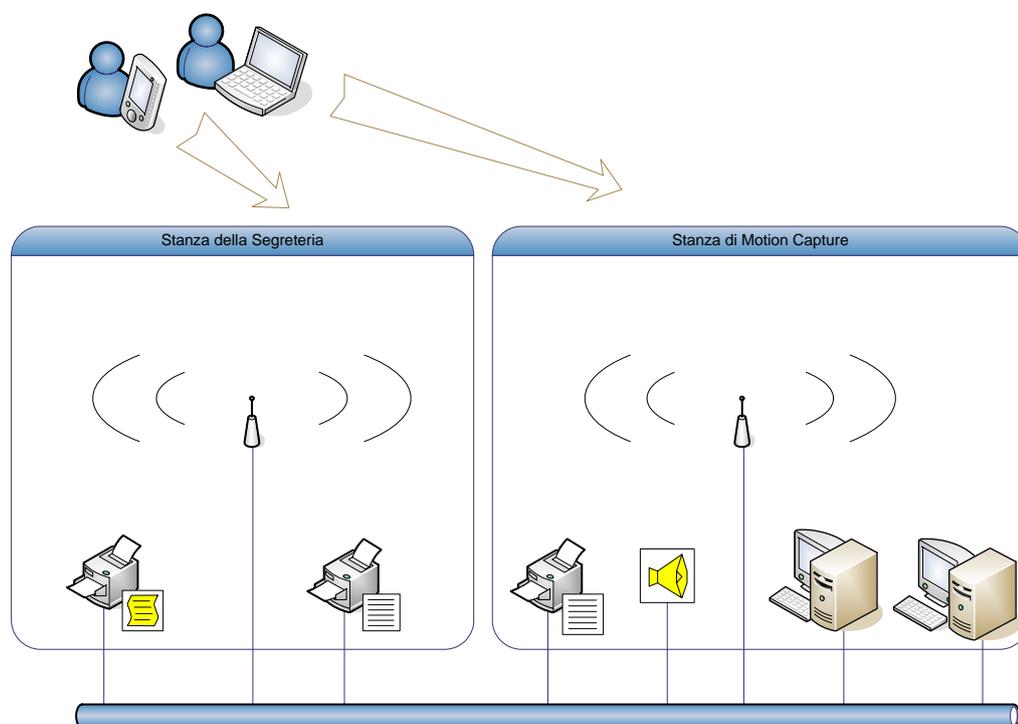


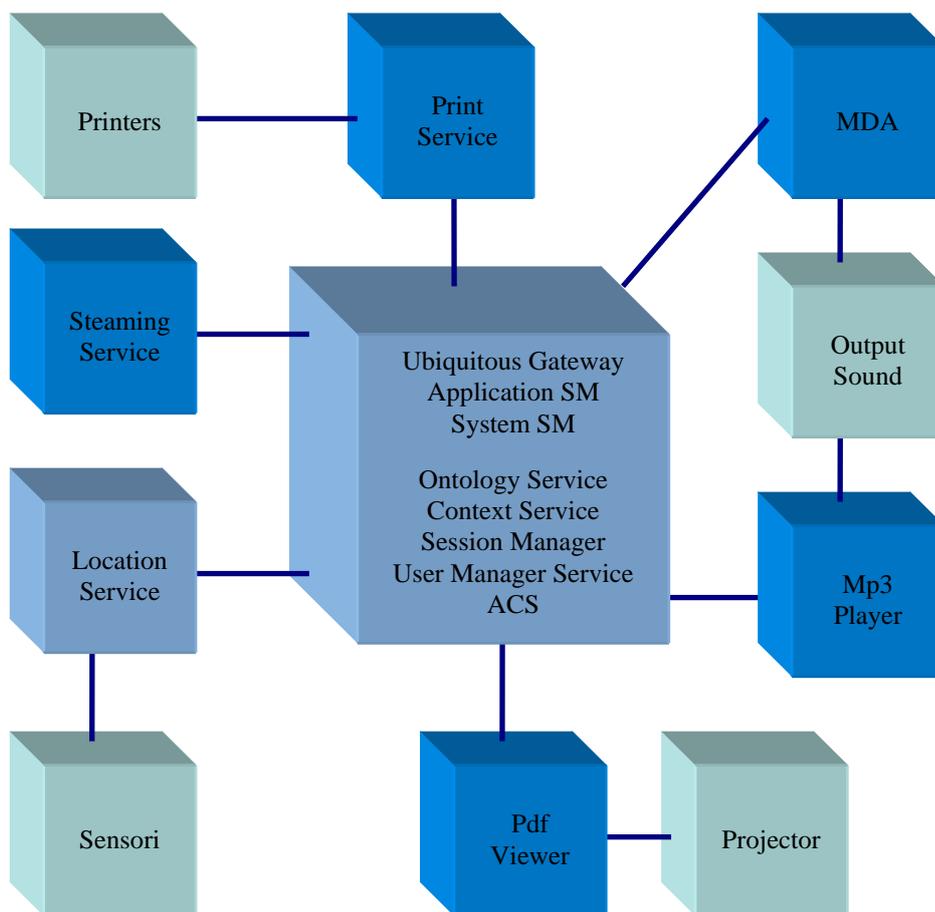
Fig. VI.1 Scenario relativo all'ambiente pervasivo implementato.

Ciascuna stanza comprende un sensore, un access-point D-link, per il rilevamento dei dispositivi mobili che entrano a far parte dell'ambiente.

Nella Stanza della Segreteria sono installate due stampanti, una per la stampa a colori, l'altra per quella in bianco e nero. La stanza di Motion Capture comprende invece una stampante per la stampa in bianco e nero ed un impianto sonoro per la riproduzione, ad esempio, di brani musicali. L'intero ambiente è coperto da una rete LAN con access-point wireless.

Oltre a questi ambienti c'è una stanza per le conferenze, dotata di schermo gigante, vicino la segreteria, e quindi nel raggio di copertura dello stesso access-point.

I componenti software di UbiSystem sono poi dislocati sui nodi della rete nel modo illustrato nella figura riportata di seguito.



**Fig. VI.2 Dislocazione dei Programmi sui nodi della rete**

I componenti potrebbero essere dislocati anche in maniera diversa.

### VI.1.1 LO START-UP DI UBISYSTEM

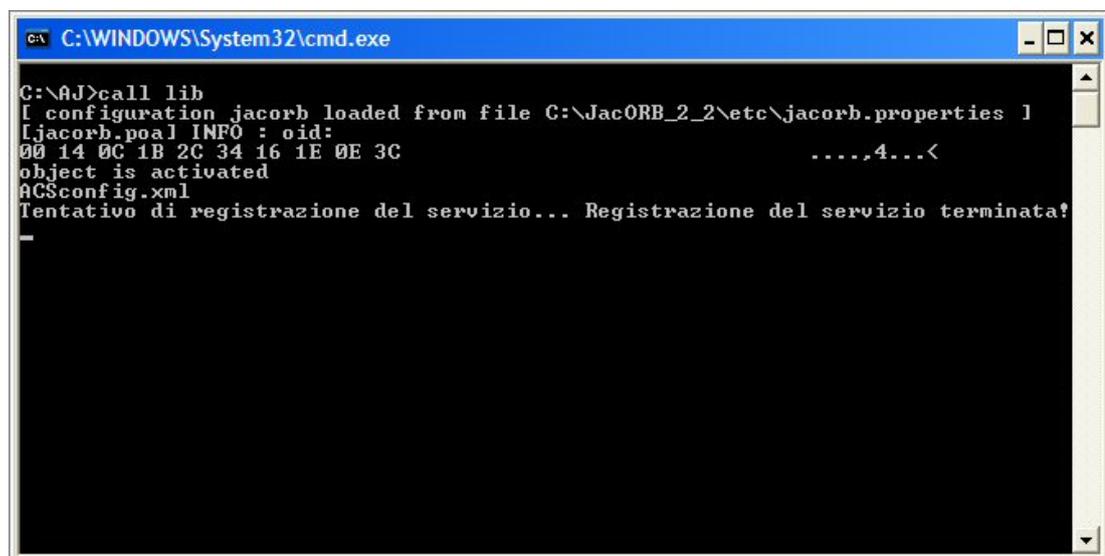
Per avviare il sistema è necessario seguire un certo ordine.

Innanzitutto è necessario avviare i manager, in modo da consentire la successiva registrazione dei servizi.

Il primo servizio a dover essere attivato deve essere l'*Ontology Service*, in modo che possa verificare le descrizioni dei servizi per validarne la correttezza sintattica e semantica secondo quanto asserito nelle ontologie. Per avviare questo servizio è necessario che sia già attivo il Fact-client, e successivamente bisognerà caricare l'ontologia di base.

Dopo aver eseguito queste operazioni potranno essere attivati gli altri servizi. Bisogna, però, fare attenzione che il *Location Service* sia attivato prima del *Context Service* e che all'attivazione di un qualunque servizio d'ambiente il *Session Manager* sia su.

Ad esempio, allo start-up di ACS, al termine della corretta registrazione del servizio viene generato il seguente output:



```
C:\WINDOWS\System32\cmd.exe
C:\>call lib
[ configuration jacobd loaded from file C:\JacORB_2_2\etc\jacobd.properties ]
[jacobd.pool INFO : oid:
00 14 0C 1B 2C 34 16 1E 0E 3C .....4...<
object is activated
ACSconfig.xml
Tentativo di registrazione del servizio... Registrazione del servizio terminata!
```

Fig. VI.3 Attivazione e registrazione dell'ACS

Prima che l'utente possa accedere al sistema, ovviamente, è necessario che *Ubiquitous Gateway* sia stato attivato.

### VI.1.2 L'INTERFACCIA UTENTE

Una volta attivato il sistema occorre analizzare il problema dell'interazione con l'utente che, per ipotesi di progetto, dovrà avvenire tramite browser.

Attualmente è lo stesso sviluppatore che mette a disposizione una serie di pagine Web attraverso le quali interagire per sfruttare le funzionalità offerte dal proprio servizio.

In alternativa, sarebbero possibili altri due tipi di scenari:

- nel momento in cui il servizio effettua la registrazione fornisce, oltre alla propria descrizione, anche l'interfaccia Web mediante la quale consumare il servizio; l'interfaccia verrebbe ospitata all'interno di un Web Application Server del sistema;
- lo sviluppo di un modulo capace di generare dinamicamente l'interfaccia Web per un servizio a partire dalla descrizione offerta.

Le due soluzioni proposte non sono esclusive ma complementari: si potrebbe adottare la seconda quando non è disponibile anche l'interfaccia utente.

Per ciascuno dei servizi pubblici è stata sviluppata un'applicazione web che offre una semplice interfaccia grafica per richiamare le funzionalità del relativo Web Service o semplicemente per mostrare la sua Home Page. Per tale scopo è stata adoperata la tecnologia delle Java Server Page e delle Servlet Java, che costituiscono una soluzione multiplatforma per la realizzazione di pagine HTML dinamiche. JSP e Servlet permettono, infatti, di svincolarsi sia dal tipo di sistema operativo che dal tipo di browser, in quanto il linguaggio Java viene interpretato solamente dal lato server sollevando il browser da questo compito, limitando la sua funzione a semplice interprete di pagine HTML.

Affinché l'utente possa interagire, attraverso le pagine *jsp*, con l'ambiente è necessario che *Tomcat* (*servlet container* utilizzato) sia attivato.

## VI.2 UBISYTEM IN AZIONE

L'entry point per l'ambiente è l'indirizzo <http://www.ubinet.it/>. La relativa pagina è mostrata in figura:



**Fig. VI.4 Home Page dell'ambiente pervasivo**

Scegliendo la voce *Servizi* del Menù a sinistra, viene visualizzata la pagina in cui vengono elencati tutti i servizi correntemente attivi nell'ambiente insieme con una breve descrizione. A seconda delle caratteristiche dell'utente, del suo dispositivo e della sua locazione i servizi sono resi disponibili o meno. Per i primi viene fornito il relativo link, per i secondi l'indicazione del motivo per cui non sono accessibili.

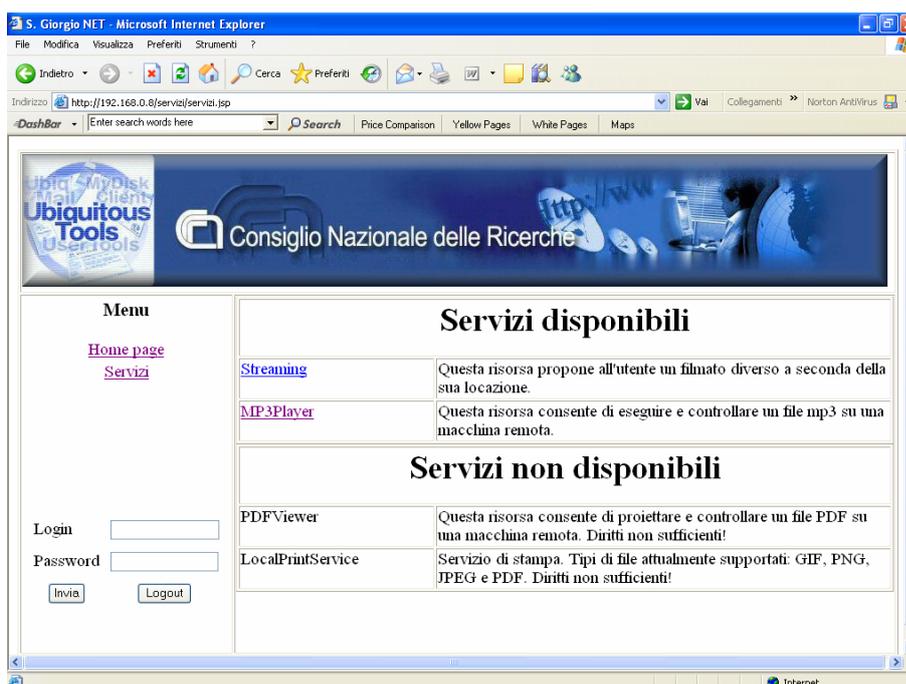


Fig. VI.5 Elenco dei servizi nell'ambiente

Nella lista non compare il servizio di musica d'ambiente in quanto non offre funzionalità fruibili direttamente dall'utente.

## VI.2.1 AUTENTICAZIONE DI UN UTENTE

Se un utente è munito di Account ha la possibilità di poter accedere ad un maggior numero di servizi. L'utente con login "Pluto" potrà usufruire di tutti i servizi.

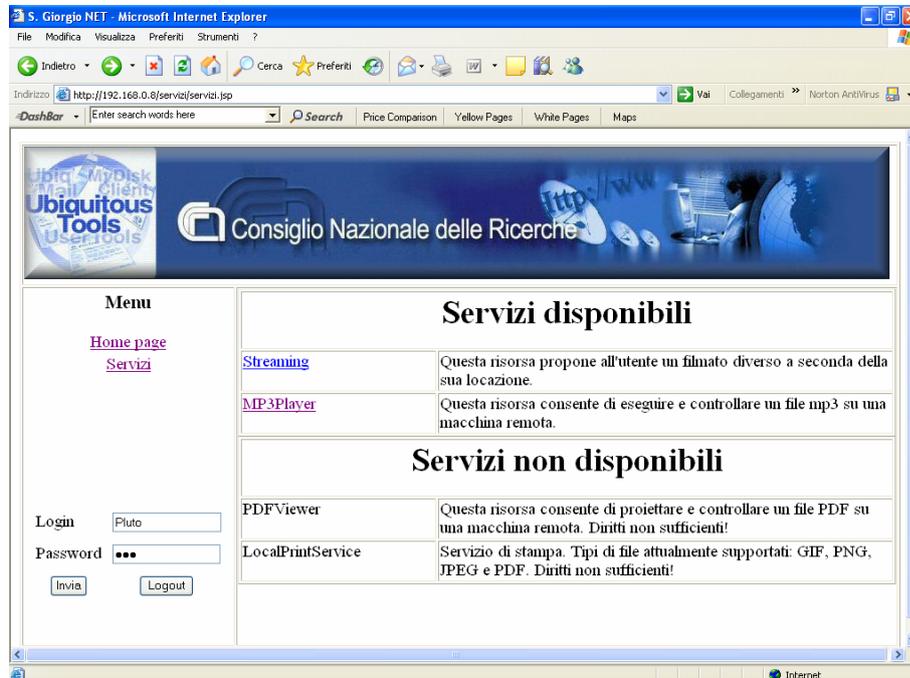


Fig. VI.6 Autenticazione di Pluto

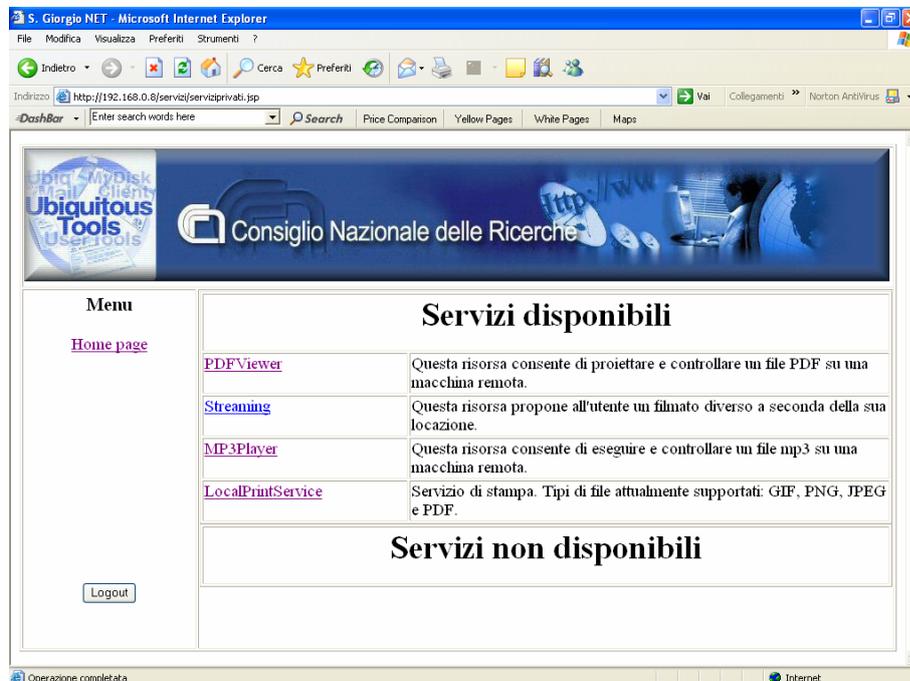
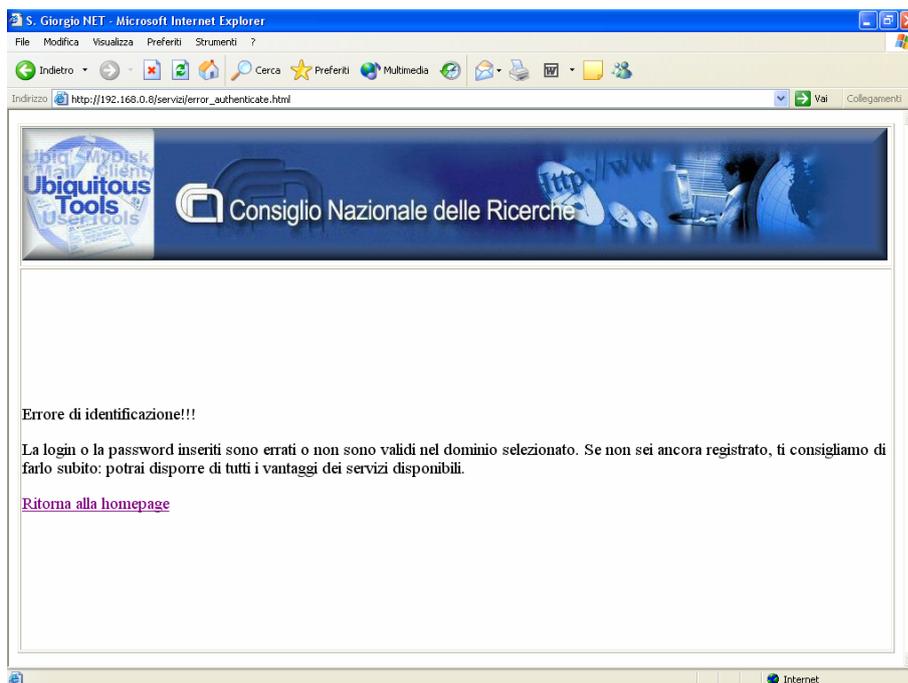


Fig. VI.7 Lista personalizzata per Pluto

Nel caso in cui i dati inseriti non fossero validi, sarà visualizzata una pagina di errore apposta:



**Fig. VI.8** Errore durante l'autenticazione

## VI.2.2 IL SERVIZIO PDF VIEWER

Dalla pagina web contenente l'elenco dei servizi disponibili è possibile accedere alla home page del servizio mediante il link relativo.

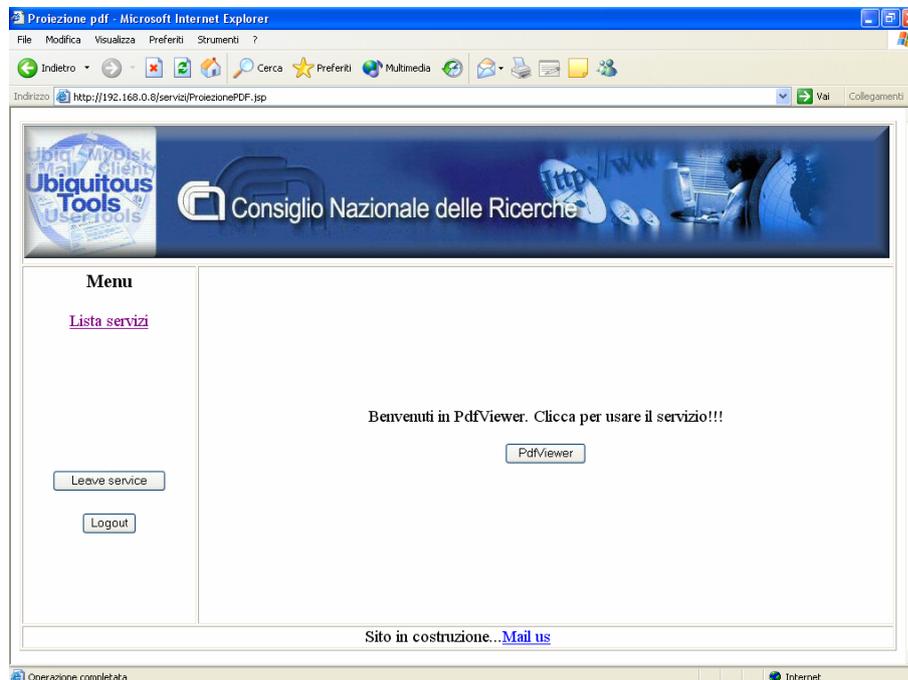
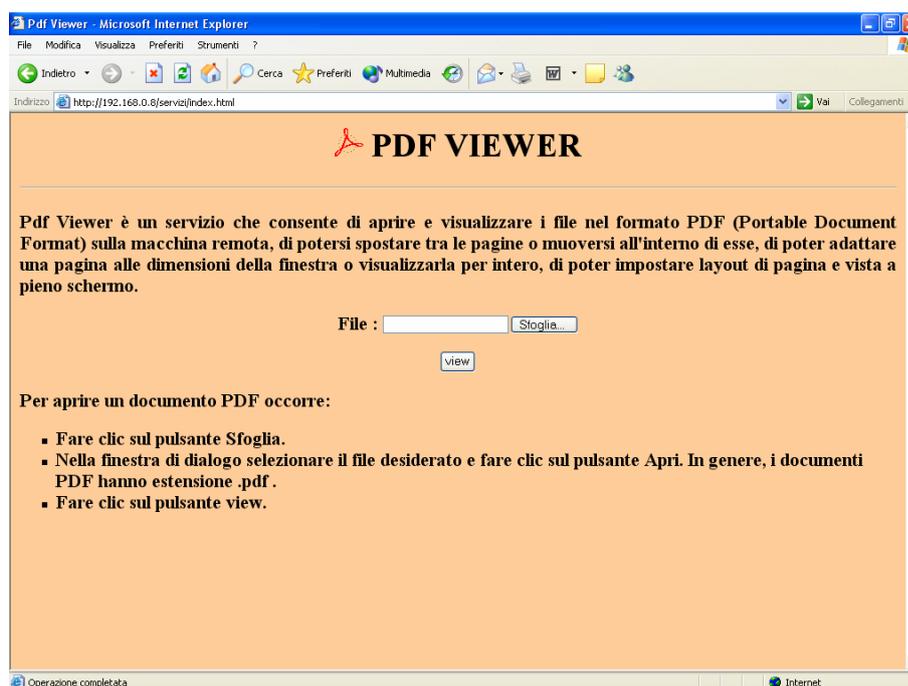


Fig. VI.9 Home page del servizio Pdf Viewer

Da qui è possibile passare all'utilizzo del servizio (pulsante "PdfViewer"), abbandonare il servizio (pulsante "Leave service") oppure lasciare l'ambiente (pulsante "Logout").

Cliccando sul bottone relativo al servizio si accede alla pagina:



**Fig. VI.10** Pagina del servizio Pdf Viewer

L'utente mediante il pulsante *Sfoglia* potrà selezionare un file pdf nel proprio File System e quindi avviare la presentazione mediante il pulsante *view*.

L'operazione potrà essere completata soltanto se viene selezionato un file la cui estensione è *.pdf*; in caso contrario verrà visualizzato un messaggio d'errore per l'utente.

Nel browser verrà quindi visualizzata la pagina contenente i pulsanti di controllo per la presentazione.

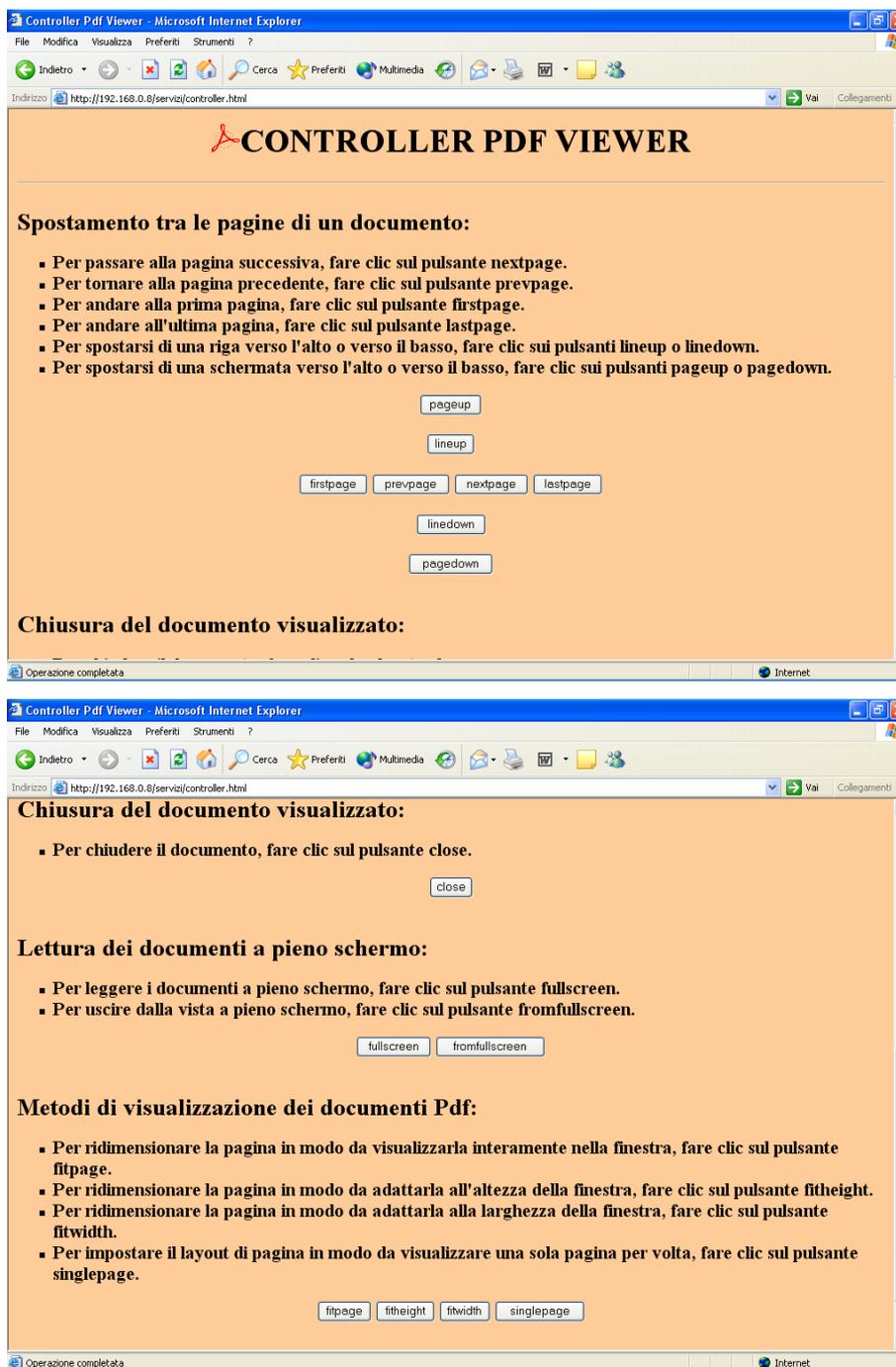


Fig. VI.11 Pagina di controllo della presentazione PDF.

## VI.2.3 IL SERVIZIO MP3 PLAYER

Dalla home page del servizio si passa alla pagina:



Fig. VI.12 Pagina del servizio Mp3 Player

La prima operazione che l'utente deve effettuare è creare una nuova playlist.

Cliccando sull'immagine "Crea la tua playlist" viene visualizzata la pagina mostrata di seguito. Le canzoni elencate sono quelle disponibili sul server.

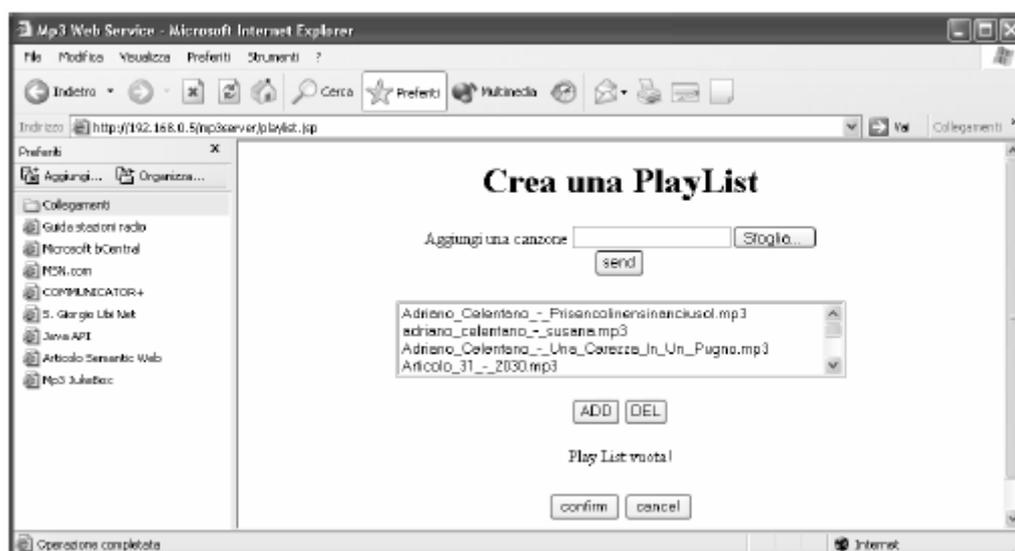


Fig. VI.13 Creazione di una playlist di brani musicali.

Attraverso l'opzione "Aggiungi canzone" è possibile effettuare l'upload sul server di una nuova canzone appartenente all'utente. Completata l'operazione, la canzone entrerà a par parte dell'elenco precedente. I pulsanti ADD e DEL servono rispettivamente per aggiungere o eliminare dalla playlist i brani selezionati.

Quando una nuova playlist è generata (la conferma attraverso il pulsante *confirm*), viene interrotta la sequenza in esecuzione e l'utente potrà ascoltare la nuova playlist tramite il pulsante play.

## VI.2.4 IL SERVIZIO DI STAMPA LOCALE

Dalla home page del servizio si passa alla pagina:

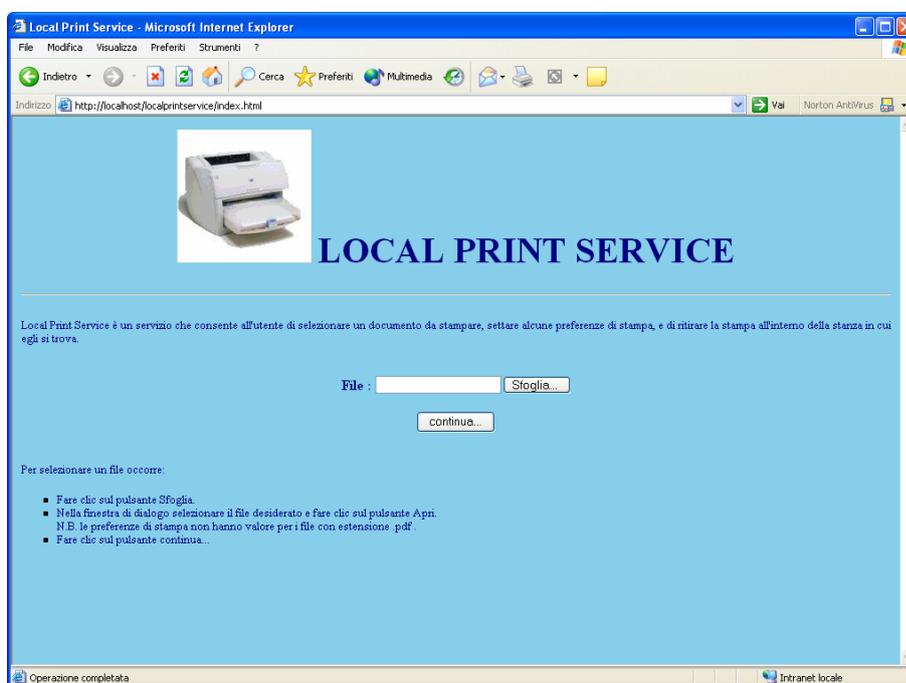
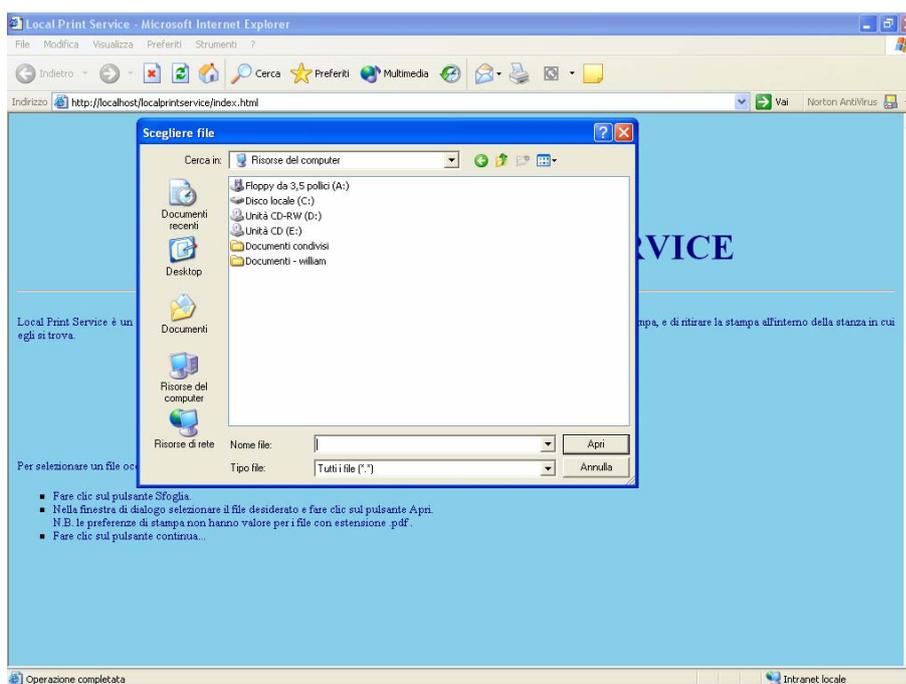


Fig. VI.14 Pagina del servizio di stampa locale

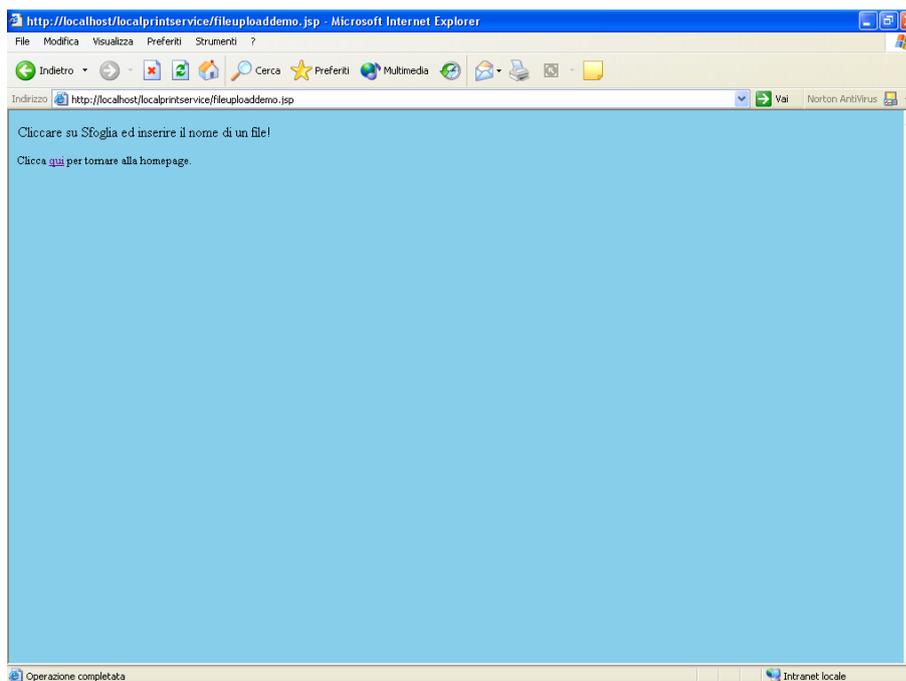
La pagina contiene una breve descrizione del servizio ed una breve guida per eseguire l'upload del documento che si intende stampare. L'utente può selezionare il documento che intende stampare attraverso il pulsante *Sfoglia*.

In seguito alla pressione del pulsante *Sfoglia* viene visualizzata la finestra che guida l'utente nella selezione del documento da stampare.



**Fig. VI.15** Finestra che guida l'utente nella scelta del file da stampare.

Selezionato il documento da stampare, l'utente può cliccare sul pulsante *continua*, presente nella home page del servizio, affinché il servizio possa poi guidarlo nella selezione delle preferenze di stampa attraverso la pagina illustrata in figura VI.17. Qualora l'utente avesse utilizzato il pulsante *continua* prima di selezionare alcun documento da stampare, il servizio di stampa locale l'avrebbe avvertito con il messaggio illustrato nella figura seguente.



**Fig. VI.16** Messaggio di errore visualizzato dal servizio quando l'utente intende settare le preferenze di stampa senza aver selezionato alcun documento da stampare.



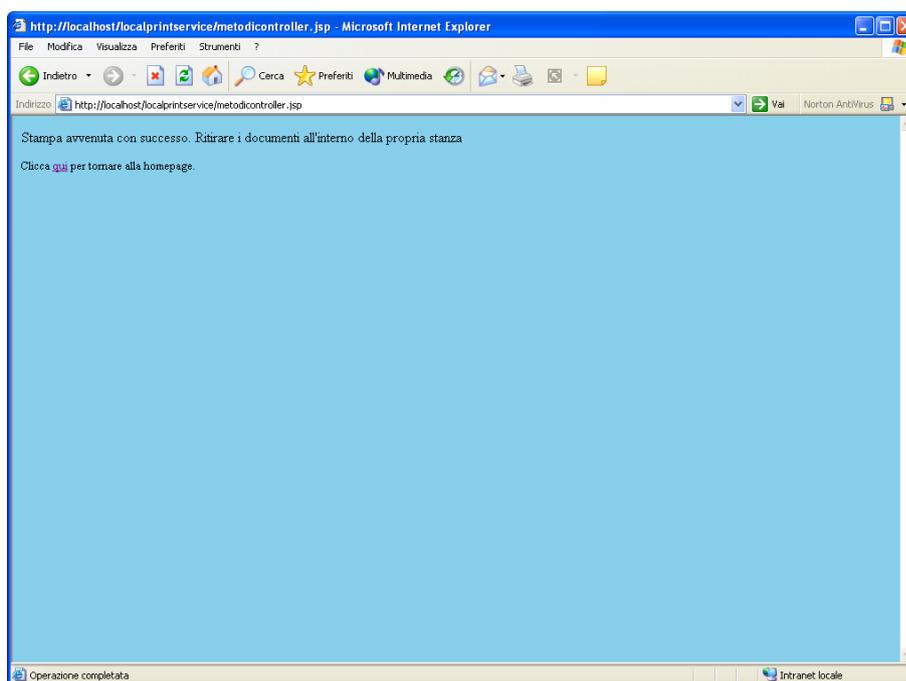
**Fig. VI.17** Pagina Web per la selezione delle preferenze di stampa.

A questo punto l'utente deve settare le proprie preferenze di stampa. Egli può selezionare la stampa a colori o in bianco e nero, mediante gli appositi bottoni radio, può ritornare alla home page per selezionare un documento da stampare diverso da

quello corrente, mediante il pulsante *ritorna*, oppure può avviare la stampa del documento, con le preferenze settate, attraverso il pulsante *stampa*.

A questo punto il servizio effettua il discovery delle stampanti presenti nella stanza dell'utente che possono soddisfare le richieste di stampa. Il discovery è possibile perché a ciascuna stampante installata nell'ambiente corrisponde un sua descrizione all'interno della ontologia di base.

Se presenti, il servizio avvia la stampa e ne comunica il buon esito attraverso il messaggio illustrato di seguito.



**Fig. VI.18** Messaggio di stampa avvenuta con successo.

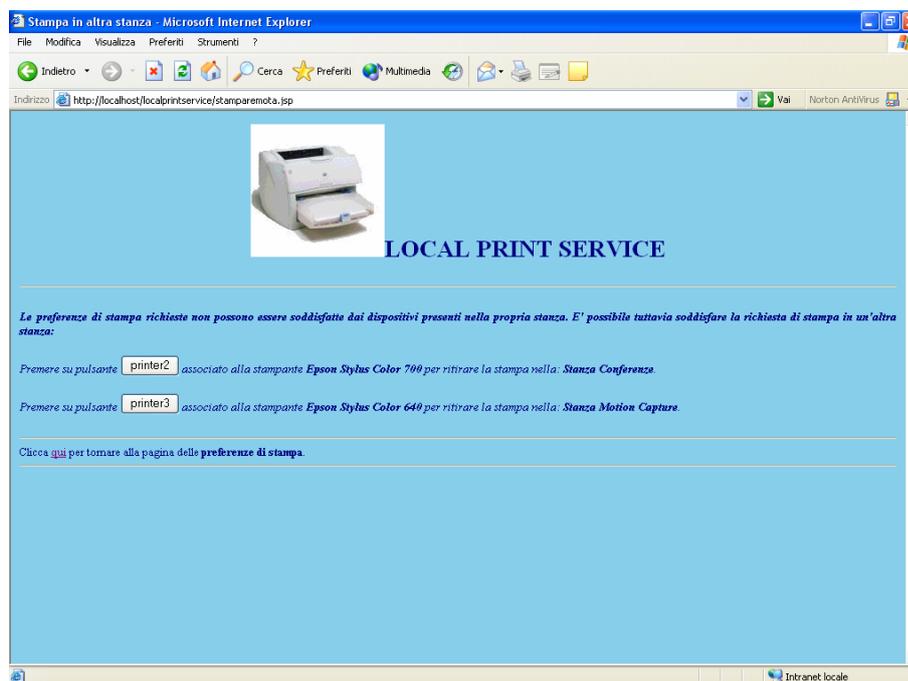
Qualora il servizio non trovi alcuna stampante che soddisfi le preferenze di stampa, all'interno della stanza in cui si trova l'utente, comunicherà a quest'ultimo la possibilità di ritirare la stampa all'interno di un'altra stanza mediante la pagina web illustrata in figura VI.19.

Questa pagina web contiene un elenco con tutte le stampanti in grado di soddisfare le richieste di stampa presenti nell'edificio e indica, per ciascuna stampante, la stanza in cui si trova.

Basandosi su quest'ultima informazione, l'utente sceglie in quale stanza prelevare la stampa.

L'utente comunica al servizio la sua scelta mediante l'apposito pulsante. Il servizio avvia prima la stampa sulla stampante nella stanza scelta dall'utente e poi

mostra a quest'ultimo una pagina web che, tramite messaggio di testo, gli ricorda la stanza dove ha scelto di ritirare i documenti stampati.



**Fig. VI.19** Pagina attraverso la quale l'utente può avviare la stampa in una stanza a sua scelta.

Quando non esiste alcuna stampante in grado di soddisfare le richieste di stampa il servizio invia all'utente l'apposito messaggio.

Le informazioni relative al nome ed alla posizione delle stampanti, elencate nella pagina web sopra illustrata, sono contenute nella ontologia di base.

## **VI.2.5 AUTENTICAZIONE DELL'AMMINISTRATORE**

Nel caso in cui ad autenticarsi è l'amministratore, verrà visualizzata la pagina in cui vengono elencati i servizi di sistema e quelli applicativi correntemente disponibili nell'ambiente, con una breve descrizione ed i relativi link.

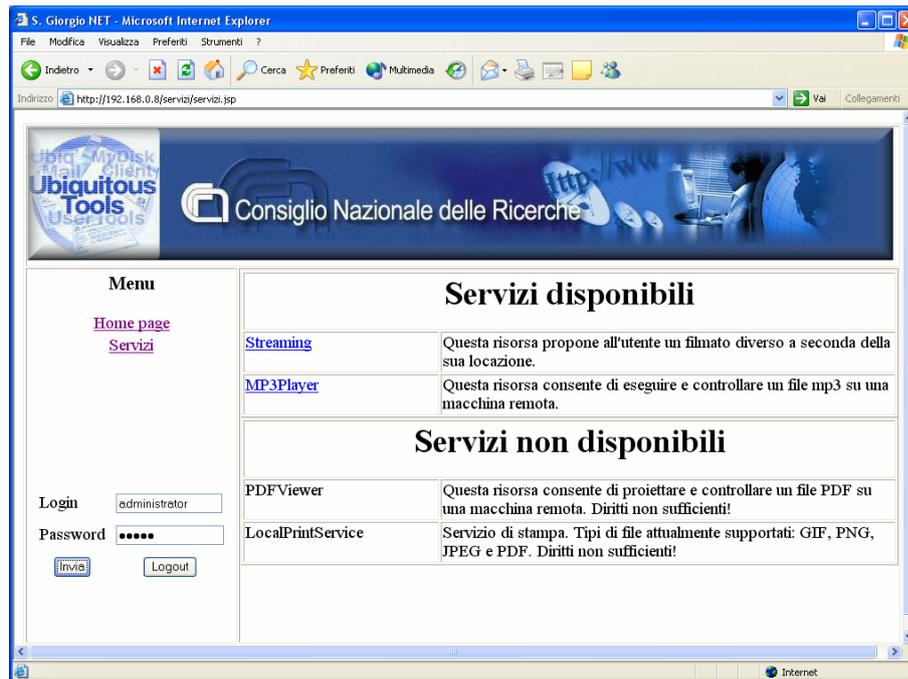


Fig. VI.20 Autenticazione dell'amministratore

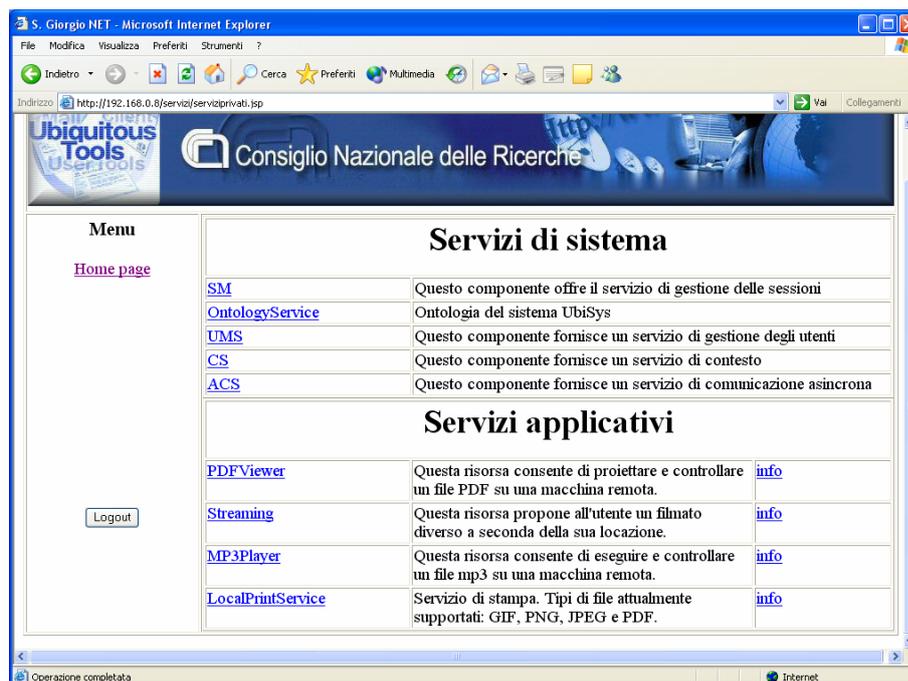


Fig. VI.21 Lista dei servizi accessibili all'amministratore

## VI.2.6 UTILIZZO DEI SERVIZI DI SISTEMA

### VI.2.6.1 SESSION MANAGER

Il link “SM” di figura VI.21 porterà alla home page del servizio *Session Manager* raffigurata in figura VI.22. Cliccando sul link “*Environment Session*” l'amministratore avrà accesso alla lista dei servizi d'ambiente attivi nel sistema.

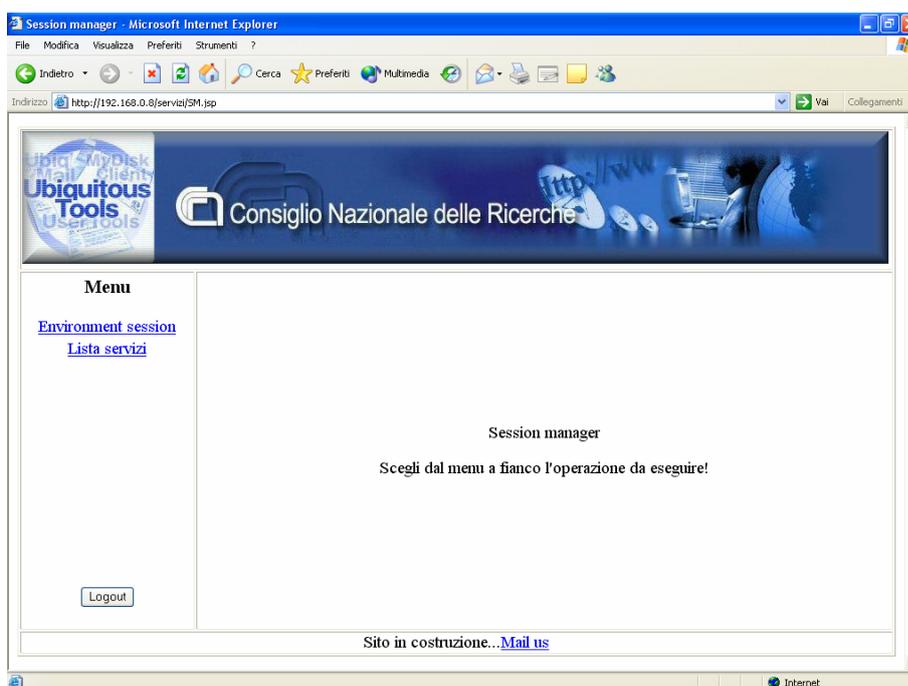
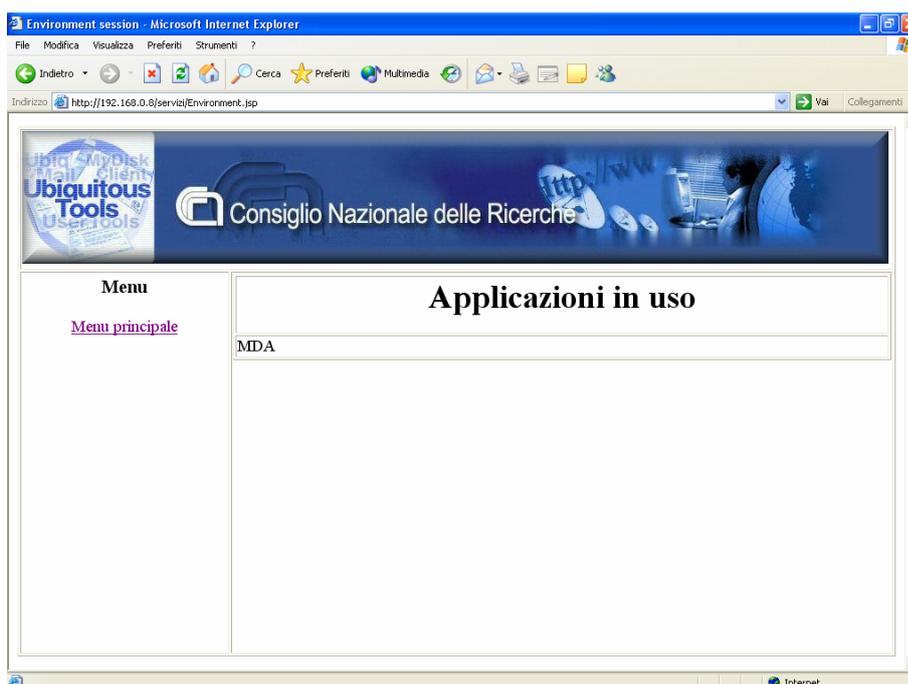


Fig. VI.22 Home page del servizio di sistema Session Manager

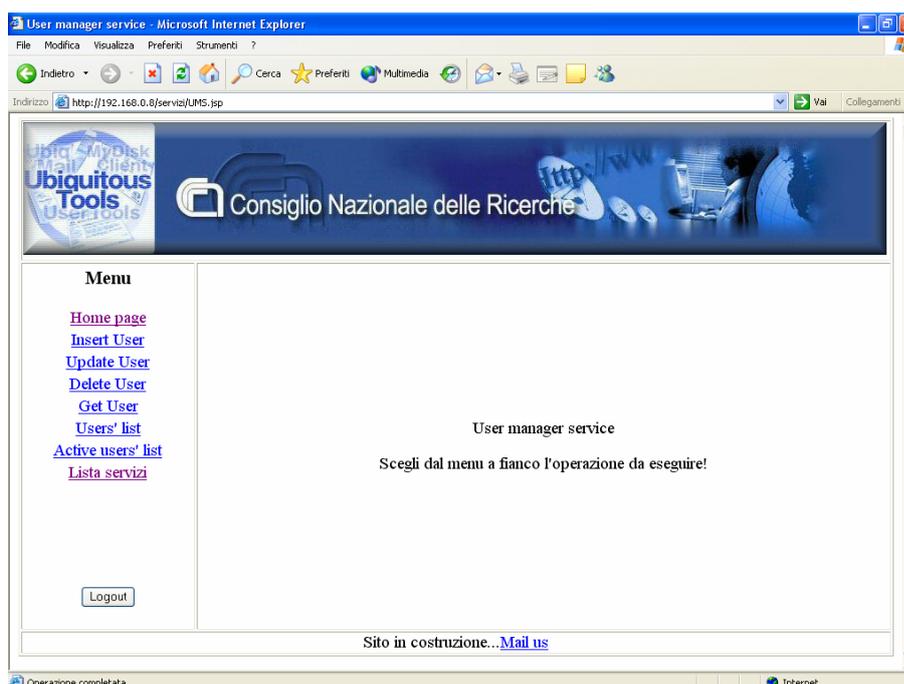


**Fig. VI.23** Lista dei servizi d'ambiente

#### VI.2.6.2 USER MANAGER SERVICE

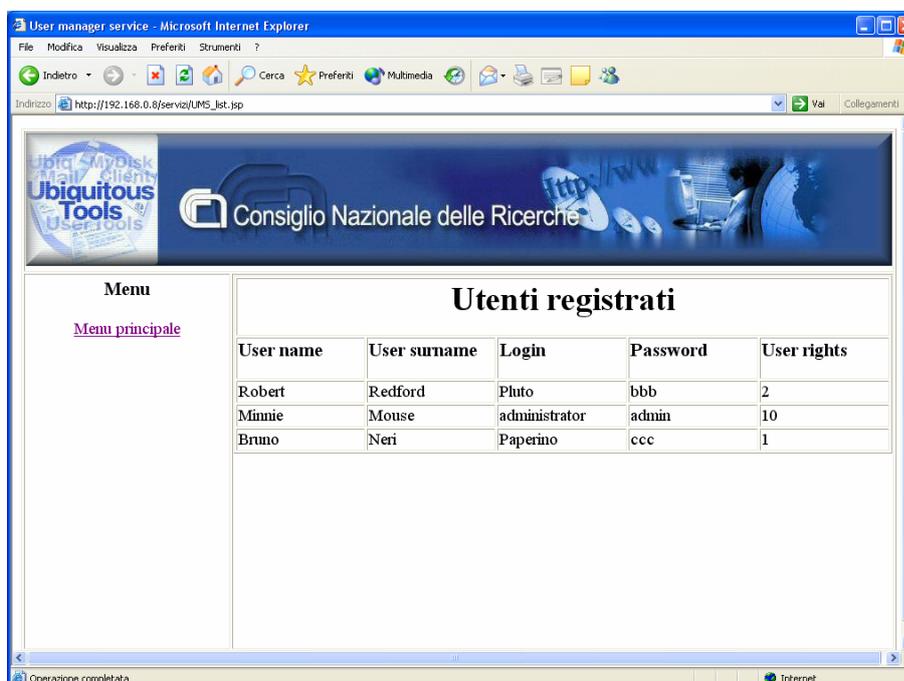
Il link “*UMS*” di figura VI.21 porterà alla home page del servizio *User Manager Service* raffigurata in figura VI.24.

Dal Menù posto a sinistra sarà possibile scegliere una delle operazioni disponibili per la gestione degli utenti.



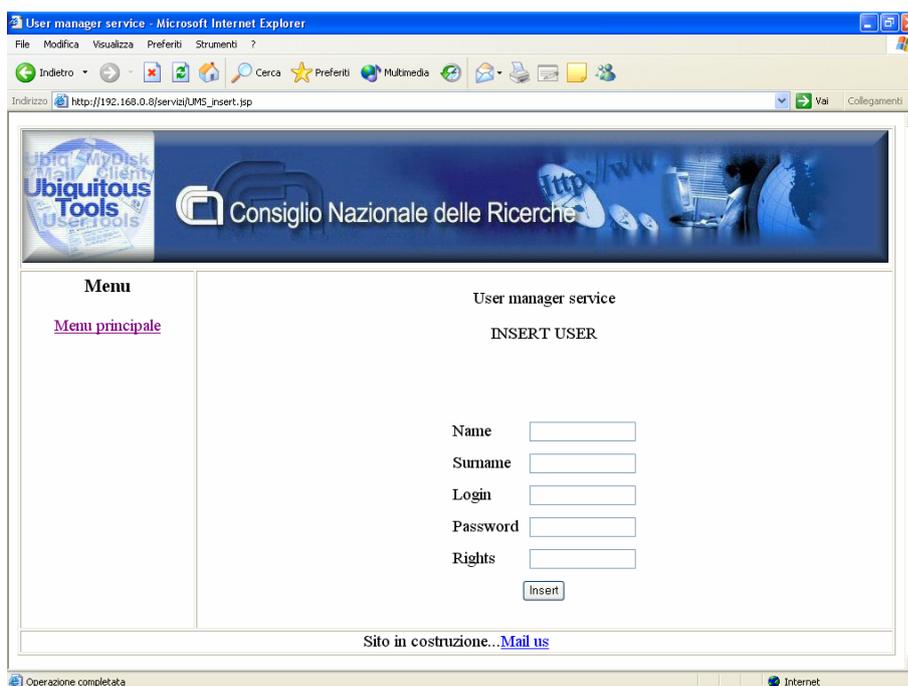
**Fig. VI.24 Home page dell'User Manager Service**

Cliccando sul link “Users’ list” l’amministratore avrà accesso alla lista degli utenti registrati presso il sistema, cioè dotati di account. Come mostra la figura VI.25, saranno visualizzate le informazioni relative a tali utenti, compresi i diritti di accesso.



**Fig. VI.25 Elenco degli utenti dotati di account**

Scegliendo la voce “*Insert User*”, sarà possibile, all’amministratore, creare un nuovo account. Come mostra la figura VI.26, verrà visualizzato un form in cui inserire i dati necessari. Una volta completato tale form e avviata la procedura d’inserimento mediante il pulsante “*Insert*”, apparirà un messaggio che indica se l’operazione sia andata a buon fine oppure no, come riportato nelle figure VI.28 e VI.29.



The image shows a screenshot of a Microsoft Internet Explorer browser window. The title bar reads "User manager service - Microsoft Internet Explorer". The address bar shows the URL "http://192.168.0.8/servizi/LMS\_insert.jsp". The page content includes a header with "Ubiquitous Tools" and "Consiglio Nazionale delle Ricerche" logos. Below the header, there is a "Menu" section with a link to "Menu principale". The main content area is titled "User manager service" and "INSERT USER". It contains a form with five input fields: "Name", "Surname", "Login", "Password", and "Rights". Below these fields is an "Insert" button. At the bottom of the page, there is a footer that says "Sito in costruzione... Mail us". The browser's status bar at the bottom indicates "Operazione completata" and "Internet".

**Fig. VI.26** Form per la creazione di un nuovo account

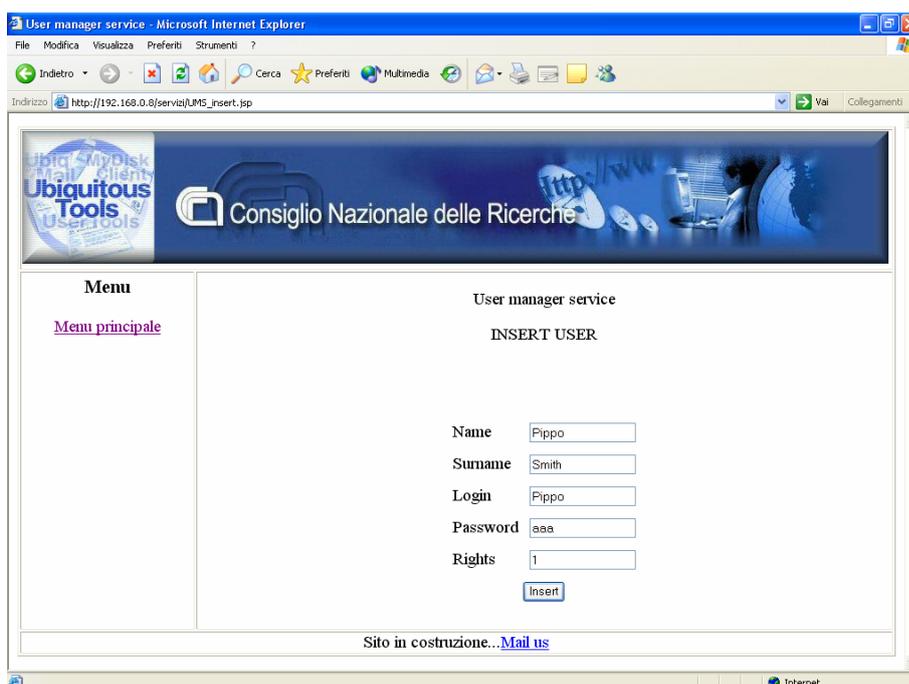


Fig. VI.27 Inserimento dei dati per la creazione di un nuovo account

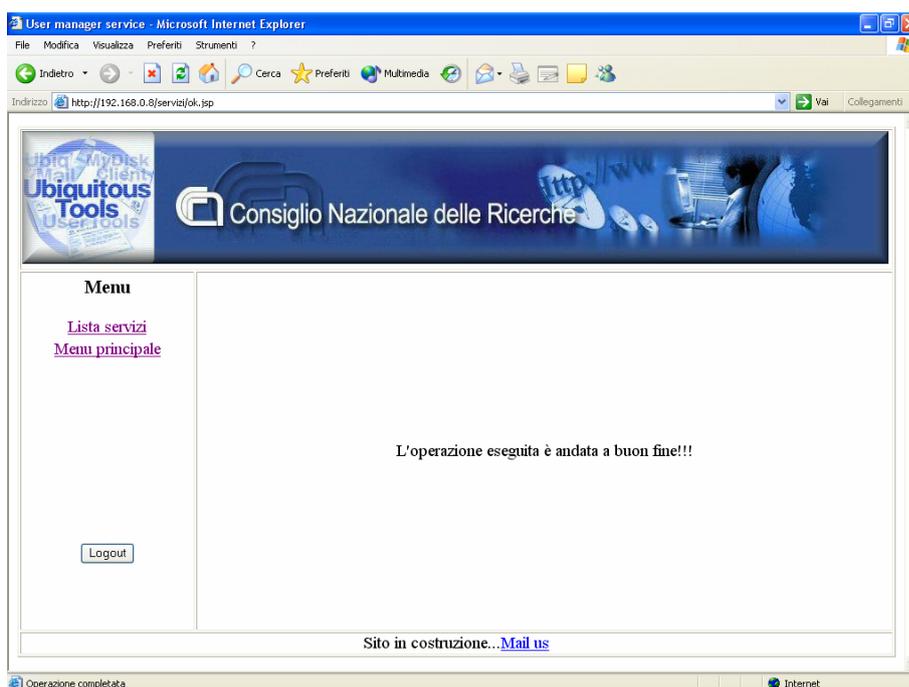


Fig. VI.28 Messaggio di operazione avvenuta con successo

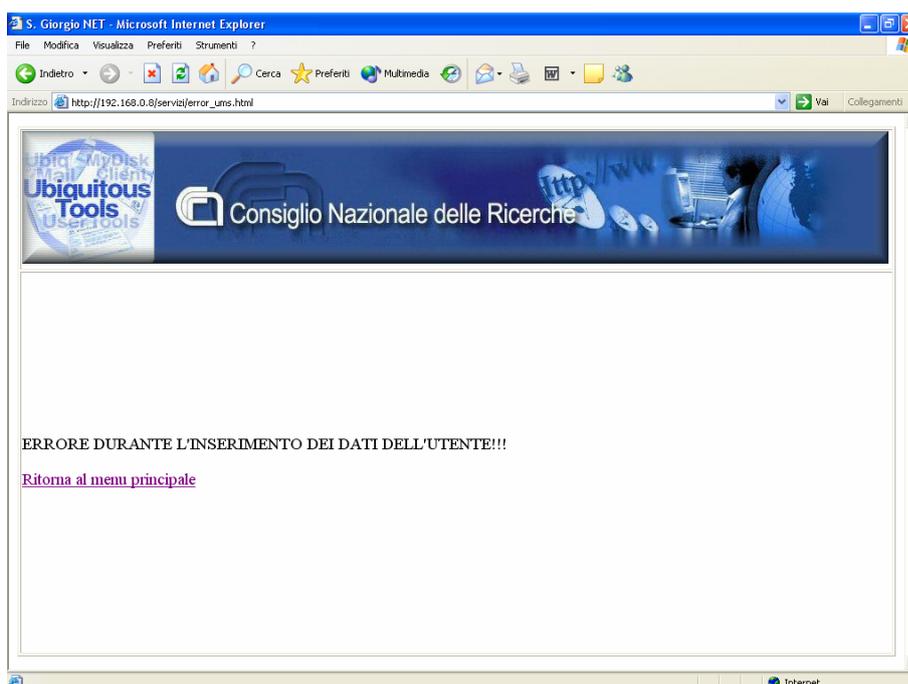


Fig. VI.29 Messaggio d'errore durante l'inserimento



Fig. VI.30 Elenco degli utenti dotati di account dopo il nuovo inserimento

Analogamente, scegliendo le voci “*Update User*”, “*Delete User*” e “*Get User*” sarà possibile, rispettivamente, data la login, modificare le informazioni relative ad un account, cancellare un account oppure ottenere le informazioni a riguardo.

Le figure che seguono, riportano tali situazioni.

Ubiquitous Tools  
Consiglio Nazionale delle Ricerche

Menu  
[Menu principale](#)

User manager service  
UPDATE USER

Name

Surname

Login

Password

Rights

[Mail us](#)

Operazione completata

Fig. VI.31 Form per la modifica di un account

Ubiquitous Tools  
Consiglio Nazionale delle Ricerche

Menu  
[Menu principale](#)

User manager service  
UPDATE USER

Name

Surname

Login

Password

Rights

[Mail us](#)

Operazione completata

Fig. VI.32 Inserimento dei dati per l'aggiornamento di un account

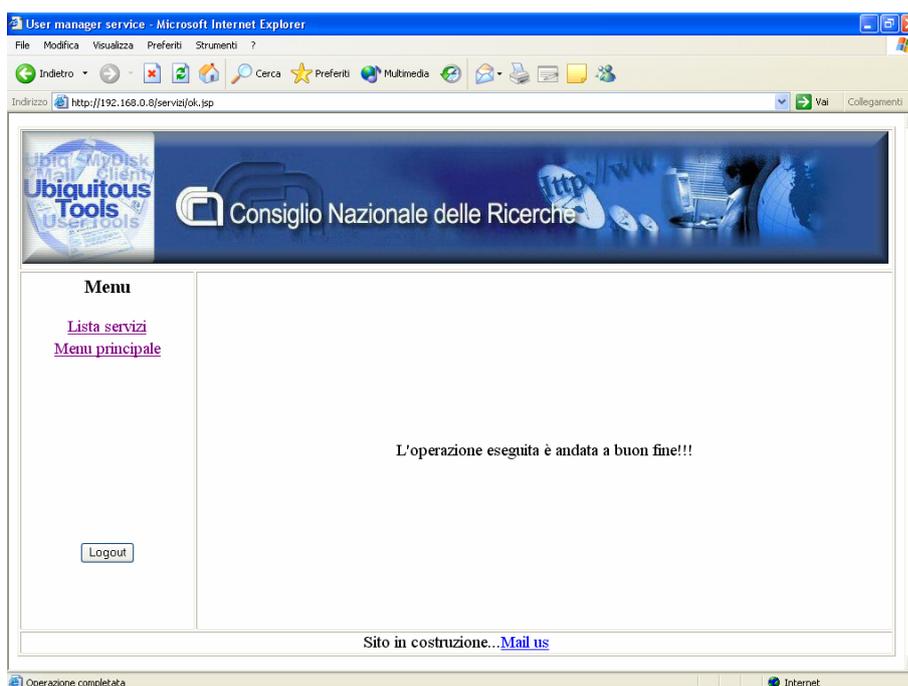


Fig. VI.33 Messaggio di operazione avvenuta con successo



Fig. VI.34 Lista degli account in cui si nota la modifica

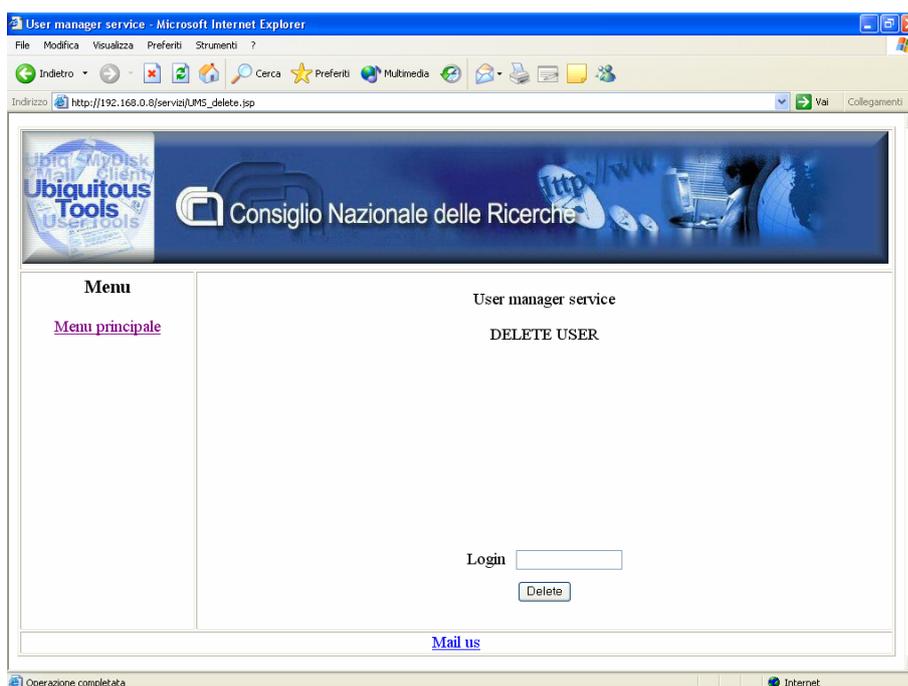


Fig. VI.35 Form per la cancellazione di un account

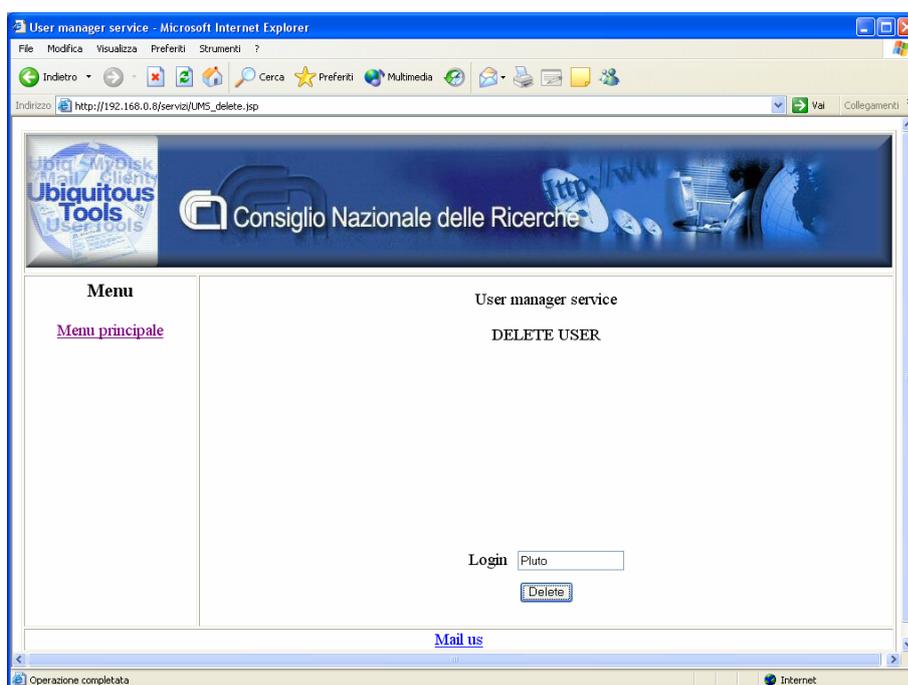


Fig. VI.36 Inserimento della login relativa all'account che si desidera cancellare

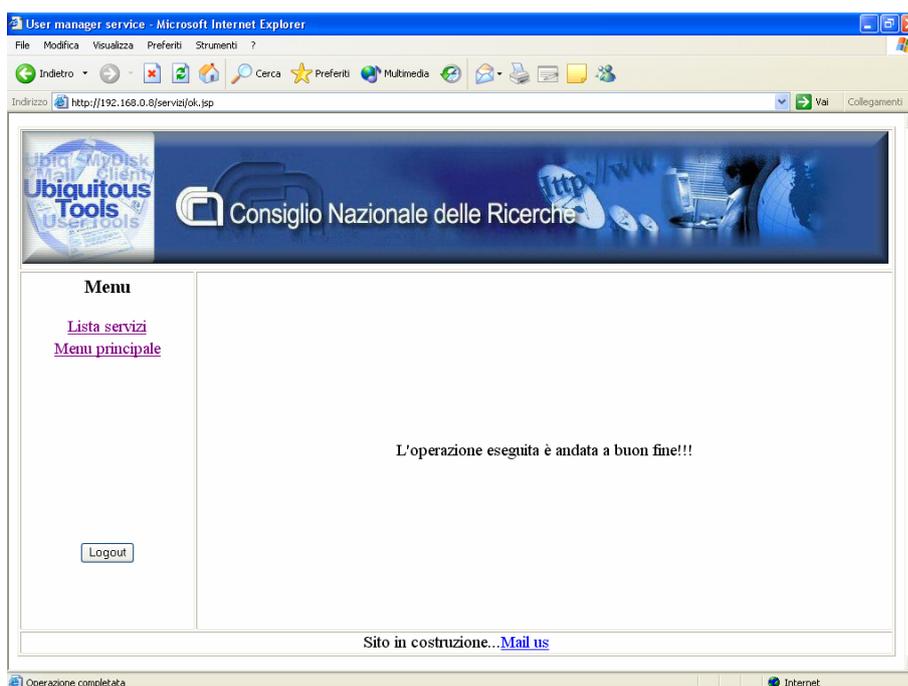


Fig. VI.37 Messaggio di operazione avvenuta con successo



Fig. VI.38 Lista degli utenti registrati in cui si nota la cancellazione dell'account relativo alla login Pluto

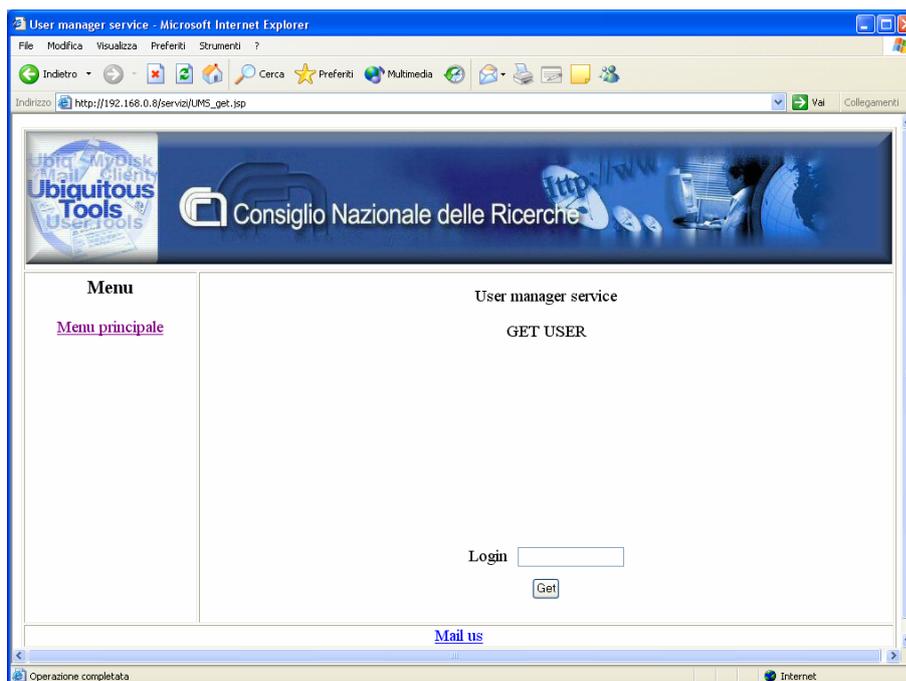


Fig. VI.39 Form per la visualizzazione dei dati di un utente

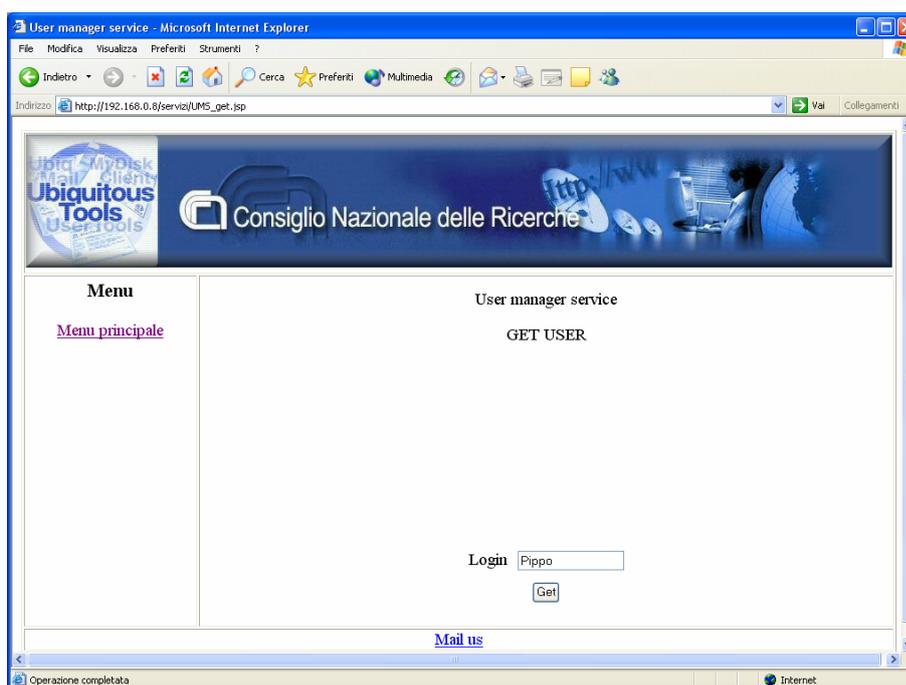
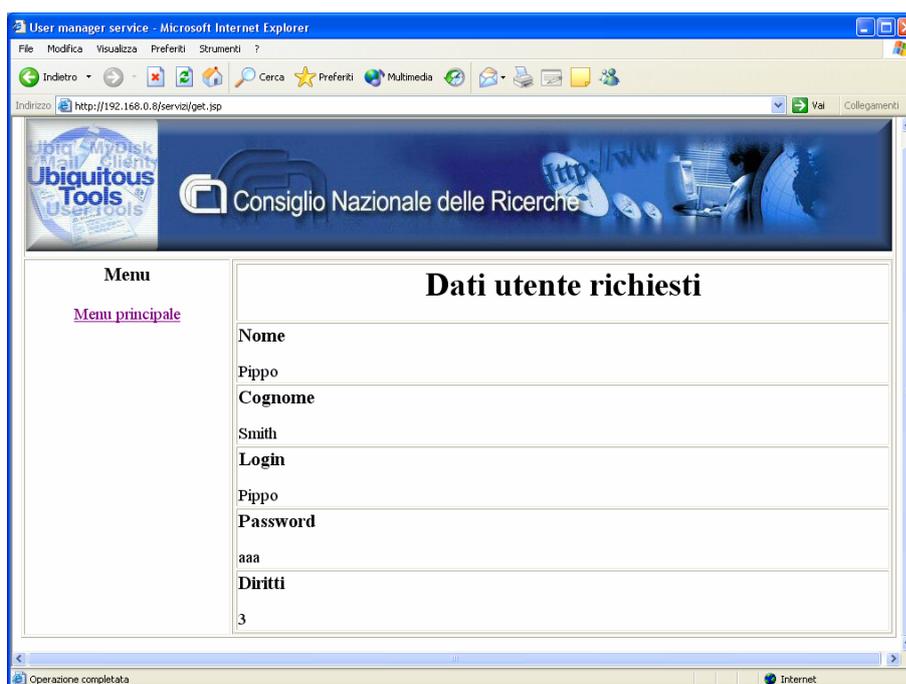


Fig. VI.40 Inserimento dei dati per la visualizzazione dei dati dell'utente con login Pippo



**Fig. VI.41** Informazioni contenute nell'account relativo alla login inserita

Scegliendo la voce "Active Users' list" l'amministratore potrà aver visione della lista degli utenti presenti attualmente all'interno dell'ambiente *UbiSystem*.

Come mostra la figura VI.42, saranno visualizzate le informazioni relative a tali utenti, riportando l'ip, la login, il tipo di device con il quale l'utente è entrato nel sistema e la location in cui esso si trova.

Inoltre, come riportato nelle figure VI.43 e VI.44, cliccando sul link relativo all'*User IP*, si potrà sapere quali servizi sta utilizzando quel particolare utente.



Fig. VI.43 Servizi attualmente in uso dall'utente Pluto

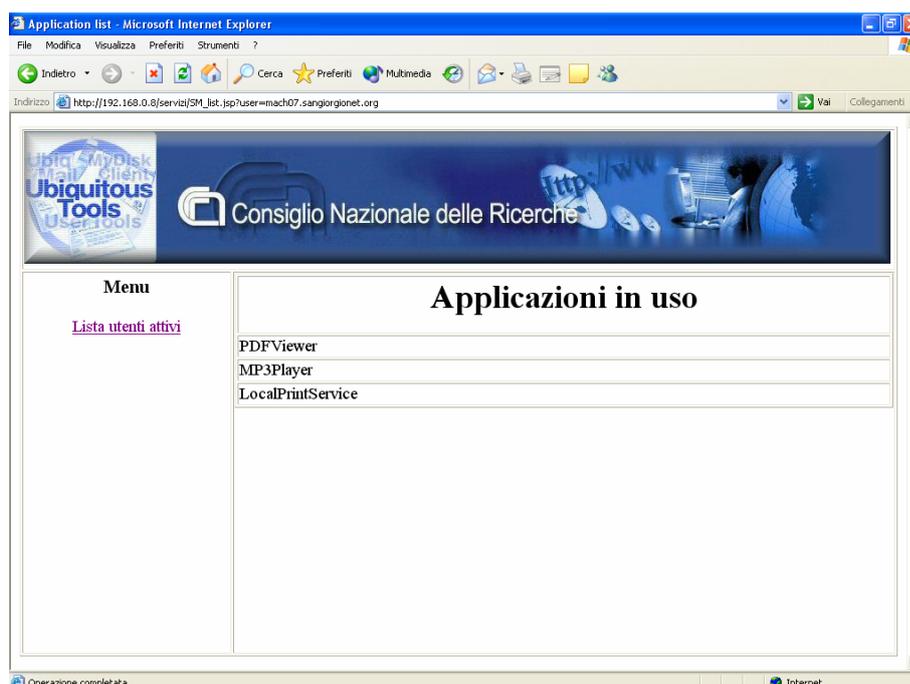


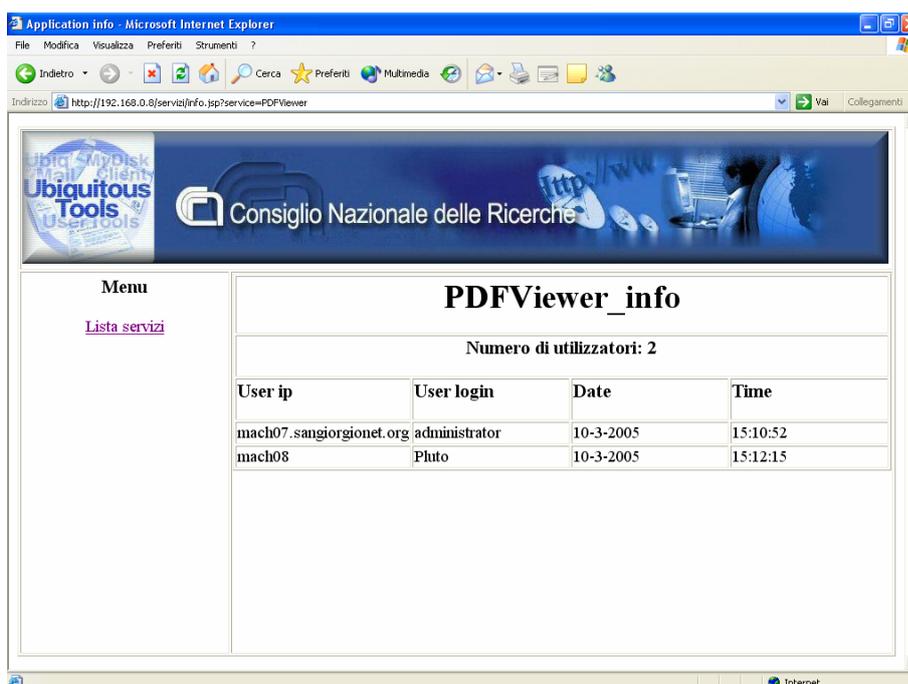
Fig. VI.44 Servizi attualmente in uso dall'utente administrator

## VI.2.7 INFORMAZIONI RELATIVE AD UN SERVIZIO APPLICATIVO

Nel paragrafo precedente si è visto come sia possibile conoscere i servizi in uso da un utente attivo nel sistema. Oltre questa funzionalità, l'amministratore può anche conoscere, viceversa, gli utenti che stanno utilizzando un particolare servizio.

Riferendoci alla figura VI.21, possiamo notare che, a fianco di ciascun servizio applicativo, è presente il link *info*. Cliccando su tale link verranno visualizzati il numero e la lista degli utenti che usano il particolare servizio.

Per ciascun utente saranno riportati l'IP e la login, oltre che la data e l'ora della richiesta del servizio.



Application info - Microsoft Internet Explorer

File Modifica Visualizza Preferiti Strumenti ?

Indietro Cerca Preferiti Multimedia

Indirizzo <http://192.168.0.8/servizi/info.jsp?service=PDFViewer> Vai Collegamenti

Ubiquitous Tools Consiglio Nazionale delle Ricerche

**Menu**

[Lista servizi](#)

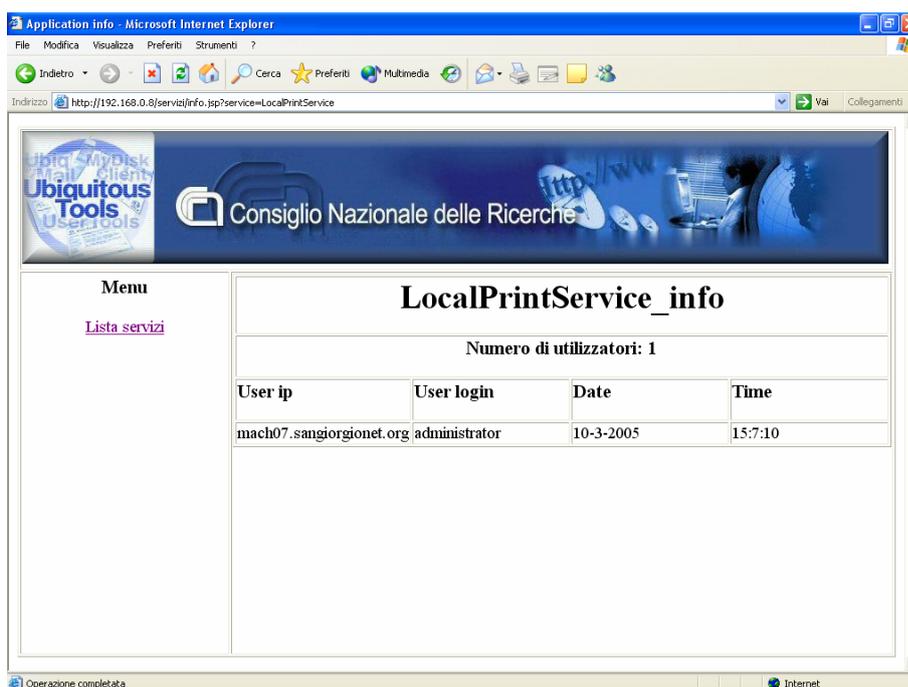
**PDFViewer\_info**

Numero di utilizzatori: 2

User ip	User login	Date	Time
mach07.sangiorgionet.org	administrator	10-3-2005	15:10:52
mach08	Pluto	10-3-2005	15:12:15

Internet

Fig. VI.45 Pagina di informazione del servizio PDFViewer



Application info - Microsoft Internet Explorer

File Modifica Visualizza Preferiti Strumenti ?

Indietro Cerca Preferiti Multimedia

Indirizzo <http://192.168.0.8/servizi/info.jsp?service=LocalPrintService> Vai Collegamenti

Ubiquitous Tools Consiglio Nazionale delle Ricerche

**Menu**

[Lista servizi](#)

**LocalPrintService\_info**

Numero di utilizzatori: 1

User ip	User login	Date	Time
mach07.sangiorgionet.org	administrator	10-3-2005	15:7:10

Operazione completata

Internet

Fig. VI.46 Pagina di informazione del servizio LocalPrintService



Fig. VI.47 Pagina di informazione del servizio MP3Player

## VI.2.8 MESSAGGI D'ERRORE

Se scade la sessione o in caso di altri problemi verrà visualizzata una generica pagina d'errore.

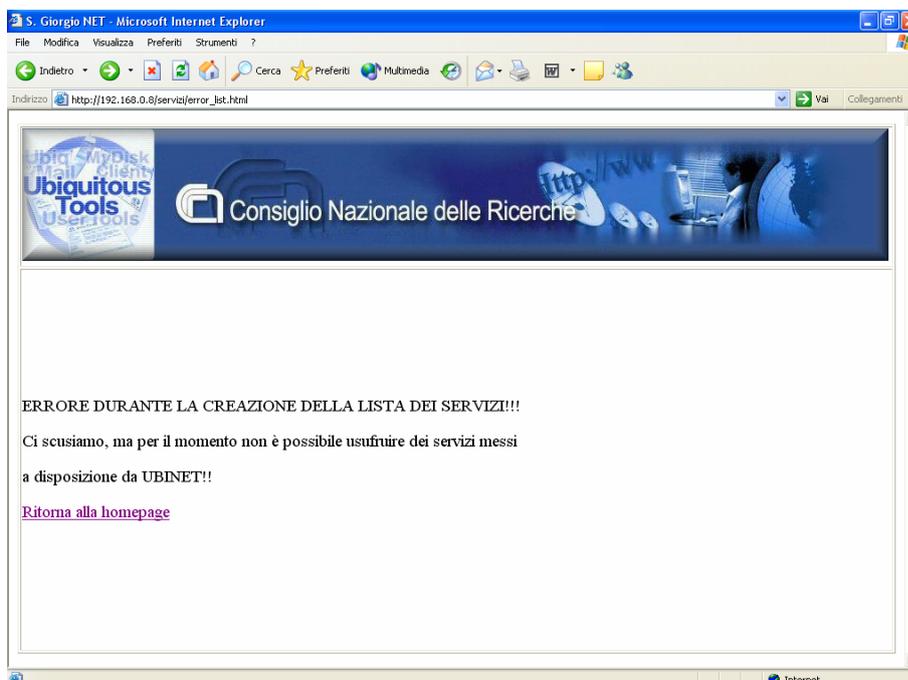


Fig. VI.48 Pagina d'errore

## CONCLUSIONI E SVILUPPI FUTURI

Il prototipo implementato adotta tecnologie web-based: l'interazione con l'ambiente e la ricerca delle risorse disponibili al suo interno avviene per mezzo di un semplice browser.

Per la realizzazione dei servizi sono state utilizzate le emergenti tecnologie dei Web Services, il cui punto di forza è di utilizzare un set base di protocolli disponibili ovunque, garantendo l'interoperabilità tra piattaforme diverse e mantenendo, comunque, la possibilità di utilizzare protocolli più avanzati e specializzati per effettuare compiti specifici.

Inoltre, l'approccio seguito ha visto l'impiego di elementi del Web Semantico. In particolar modo, si è fatto ricorso all'uso di ontologie per garantire l'interoperabilità sintattica e semantica fra le diverse entità presenti nell'ambiente e per fornire dei meccanismi avanzati di condivisione delle informazioni e di discovery dei servizi.

Il nuovo prototipo di UbiSystem include, ora, dei componenti che si occupano di applicare delle politiche e meccanismi di gestione dei servizi per facilitare l'interazione con l'ambiente pervasivo. Un utente, munito di dispositivo mobile, può muoversi in maniera facile ed intuitiva all'interno dell'ambiente per scoprire ed utilizzare le risorse disponibili in relazione alla propria posizione, alle caratteristiche del suo device e dei propri diritti d'accesso.

Per fare in modo di testare le nuove funzionalità introdotte è stato necessario apportare delle modifiche ai servizi applicativi esistenti.

Ci sono problemi a cui il sistema implementato ancora non fornisce una soluzione.

Ad esempio non è stato ancora risolto il problema di come un dispositivo possa fornire la sua descrizione in maniera automatica e dettagliata, quando entra a far parte dell'ambiente. Tale descrizione è infatti indispensabile per determinare il set di servizi compatibili con il dispositivo che l'ambiente può offrire oppure per determinare i servizi che il dispositivo stesso può offrire nel momento in cui entra a far parte dell'ambiente, considerazione, quest'ultima, di rilievo se si pensa anche al caso in cui è l'utente che entra nell'ambiente e vuole esporre un proprio servizio.

L'attuale soluzione prevede il prelievo delle informazioni contenute nell'header HTTP nel momento in cui viene effettuata una richiesta e la generazione di una descrizione di massima. Su questa strada si sta muovendo il World Wide Web Consortium che sta già affrontando questo genere di problematiche.

In questo lavoro di tesi non sono state toccate questioni riguardanti la sicurezza e la privacy delle informazioni scambiate: aspetti, che nei contesti di pervasive computing, diventano di fondamentale importanza.

Concludendo, l'architettura proposta rappresenta soltanto un ulteriore passo avanti verso la realizzazione di un ambiente pervasivo completo; molti sono gli sforzi che ancora si devono compiere e numerosi restano gli ambiti in cui la ricerca dovrà muoversi affinché l'*Ubiquitous Computing* diventi una realtà.

## BIBLIOGRAFIA

- [1] Anind K.D., “*Providing Architectural Support for Building Context-Aware Application*”, PhD thesis, Georgia Institute of Technology, 2000.
- [2] Apache Axis - Web Services, <http://ws.apache.org/axis/>.
- [3] Apache Jakarta Project, <http://jakarta.apache.org/>.
- [4] Banavar G., “*Challenges in Design and Software Infrastructure for Ubiquitous Computing Application*”, IBM TJ Watson Research Center, University of Zurich.
- [5] Barbeau M., “*Mobile, Distributed and Pervasive Computing*”, Handbook of Wireless Networks and Mobile Computing, John Wiley and Sons, Inc., Febbraio 2002.
- [6] Barbieri L., “*Web Services: Protocolli e Strumenti, Interoperabilità ed evoluzioni future*”, 2003.
- [7] Berners-Lee T., “*Sematic Web road map*”, <http://www.w3.org/DesignIssues/Semantic.html>.
- [8] Finin T., Chen H., “*An Ontology for Context Aware Pervasive Computing Environments*”, University of Mariland, Baltimore County, USA, 2003.

- 
- [9] Chen H., Finin T., “*Semantic Web in the Context Broker Architecture*”, University of Mariland, Baltimore County, USA, 2004.
- [10] Cooltown Homepage, <http://www.cooltown.com>.
- [11] DAML-S, <http://www.daml.org/services>
- [12] DARPA Agent Markup Language Homepage, <http://www.daml.org/>
- [13] Gellersen H.W., Davies N., “*Beyond Prototypes: Challenges in Deploying Ubiquitous Systems*”, IEEE Pervasive Computing, Gennaio-Marzo 2002.
- [14] GAIA Homepage, <http://choices.cs.uiuc.edu/gaia/>.
- [15] Grimm R. et al., “*System Directions for Pervasive Computing*”, University of Washington, 2000.
- [16] Harms D., “*JSP Servlet e MySql*”, Mc Graw-Hill Editore, Milano, 2002.
- [17] Rakotonirainy A., Henricksen K., Indulska J., “*Infrastructure for Pervasive Computing: Challenges*”, School of Computer Science and Electrical Engineering, The University of Queensland, 2000.
- [18] IEEE Pervasive Computing Homepage, <http://www.computer.org/pervasive/>.
- [19] Joshi A., Kagal L., Korolev V., Avancha S., Finin T., Yesha Y., “*Centaurus: An Infrastructure for Service Management in Ubiquitous Computing Environments*”, Dept. of Computer Science and Electrical Engineering, University of Maryland Baltimore Conty, USA, 2002.
- [20] Barton J., Kindberg T., “*A web-based nomadic computing system*”, White paper, HP Labs, cooltown.hp.com, 2001.
- [21] Fox A., Kindberg T., “*System Software for Ubiquitous Computing*”, IEEE Pervasive Computing, Gennaio-Marzo 2002.

- 
- [22] Maffioletti F., “*Requirements for an Ubiquitous Computing Infrastructure*”, *Cultura Narodov Prichernomoria Journal* vol.3. Simferopol, Ukraine, Settembre 2001.
- [23] Magnanini P., “*Sistemi di discovery: La soluzione Jini*”, Università degli Studi di Bologna, 2001.
- [24] Capra L., Mascolo C., Emmerich W., “*Mobile Computing Middleware*”, Dept. of Computer Science, University College London, 2002.
- [25] Sturm P., Mattern F., “*From Distributed Systems to Ubiquitous Computing*”, Department of Computer Science, Switzerland - Germany, 2002.
- [26] McGrath R.E., Ranganathan A., Campbell R.H., Mickus M.D., “*Use of Ontologies in Pervasive Computing Environments*”, Dept. of Computer Science, University of Illinois, Urbana-Champaign, USA, Aprile 2003.
- [27] Project Aura Homepage, <http://www-2.cs.cmu.edu/~aura/>.
- [28] Ranganathan A., Campbell R.H., “*A Middleware for Context-Aware Agents in Ubiquitous Computing Environments*”, Department of Computer Science, University of Illinois at Urbana-Champaign, USA, 2002.
- [29] Hess C., Roman M., Cerqueira R., Ranganathan A., Campbell R.H., Nahrsted K., “*A Middleware Infrastructure for Active Spaces*”, *IEEE Pervasive Computing*, Ottobre-Dicembre 2002.
- [30] Mukherjee A., Saha D., “*Pervasive Computing: A Paradigm for the 21st Century*”, *IEEE Computer Society*, Marzo 2003.
- [31] Saracco R., “*Ubiquitous Computing*”, *Mondo digitale*, n.3, Settembre 2003.
- [32] Satyanarayanan M., “*Pervasive Computing: Vision and Challenges*”, *IEEE Personal Communication*, Agosto 2001.
- [33] UDDI.org, *Universal Description, Discovery and Integration of Web Services*, <http://uddi.org/>.
-

- 
- [34] Baldoni R., Virgillito A., “*Una introduzione alla architettura CORBA con elementi avanzati di interazione client/server in Java*”, Dip. di Informatica e Sistemistica, Università di Roma “La Sapienza”.
- [35] Wallbank N., “*A Requirements Analysis of Infrastructure for Ubiquitous Computing Environments*”, Computing Department, Lancaster University, 2002.
- [36] Weiser M., “*The Computer for the 21<sup>st</sup> Century*”, Scientific Am., Settembre 1991; riedito dall’IEEE Pervasive Computing, Gennaio-Marzo 2002.
- [37] World Wide Web Consortium, *Extensible Markup Language (XML) 1.0 (Third Edition)*, <http://www.w3.org/TR/2004/REC-xml-20040204/>, Febbraio 2004.
- [38] World Wide Web Consortium, *OWL Web Ontology Language Semantics and Abstract Syntax*, <http://www.w3.org/TR/2004/REC-owl-semantics-20040210/>, Febbraio 2004.
- [39] World Wide Web Consortium, *Resource Description Framework*, <http://www.w3.org/RDF/>.
- [40] World Wide Web Consortium, *SOAP Version 1.2 Part 1: Messaging Framework*, <http://www.w3.org/TR/soap12-part1/>, Giugno 2004.
- [41] World Wide Web Consortium, *Web Services Architecture*, <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>, Febbraio 2004.
- [42] World Wide Web Consortium, *Web Services Description Language (WSDL) 1.1*, <http://www.w3.org/TR/wsdl>, Marzo 2001.
- [43] Zhu F., Mutka M., Ni L., “*Classification of Service Discovery in Pervasive Computing Environments*”, Michigan State University, East Lansing, 2002.
- [44] Stefano Russo, Carlo Savy, Domenico Cotroneo, Antonio Sergio, “*Introduzione a CORBA*”, McGraw-Hill, 2002.

- [45] Pradeep Gore, Ron Cytron Douglas Schmidt, Carlos O’Ryan, “*Designing and Optimizing a Scalable CORBA Notification Service*”.
- [46] Horrocks, Ian and Sattler, “CORBA-FaCT”,  
<http://www.cs.man.ac.uk/~horrocks/FaCT/CORBAFsCT.html>
- [47] David Garlan, Bradley Schmerl, “*Component-Based Software Engineering in Pervasive Computing Environments*”, Carnegie Mellon University.
- [48] JacORB, <http://www.jacorb.org>
- [49] João Pedro Sousa, David Garlan, “*Aura: An Architectural Framework for User Mobility in Ubiquitous Computing Environments*”, School of Computer Science Carnegie Mellon University, August 2002.