Università degli Studi di Padova

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Corso di Laurea Magistrale in Ingegneria Informatica

Piattaforme JAVA a supporto di servizi convergenti in reti di nuova generazione (IMS)

Laureando:
Federico Ballabio

Relatore:
Prof. Massimo Maresca

Correlatore:
Ing. Michele Stecca

Anno Accademico 2010–2011

Alla mia famiglia

Ringraziamenti

La tesi che viene presentata in queste pagine è solamente l'ultimo atto di un percorso, durato cinque anni, che oltre alle conoscenze nell'ambito dell'Ingegneria Informatica mi ha dato tante altre importanti lezioni di vita: in questa avventura ho avuto la fortuna di essere circondato da persone che hanno saputo volermi bene e con le quali ho condiviso tante esperienze che sono state sicuramente utili alla mia formazione.

Mi preme innanzitutto ringraziare i miei compagni di corso, senza i quali l'avventura universitaria non sarebbe stata così piacevole e ricca di soddisfazioni: insieme a Massimiliano, Luca, Davide, Lucio e Michele ho affrontato questo percorso senza scoraggiarmi d'innanzi alle difficoltà (che pure non sono mancate), ed è soprattutto grazie a loro se il ricordo che mi resta in mente di questi anni di studi è così straordinario.

Altrettanto importanti sono stati i numerosi amici di Bassano e dintorni, con i quali nel fine settimana ho potuto 'ricaricarmi' a suon di risate e serate in compagnia: senza di loro gli ultimi anni non sarebbero stati così divertenti.

Un sentito grazie anche agli amici e amiche della Facoltà di Lettere e Filosofia, con i quali ho condiviso pranzi indimenticabili e giornate diverse dall'ordinario.

Ringrazio poi il mio relatore Prof. Massimo Maresca, per la disponibilità dimostrata durante la stesura della tesi e per i preziosi consigli dispensati. Allo stesso modo va un grosso grazie all'Ing. Michele Stecca, che con grande pazienza mi ha aiutato nella risoluzione dei problemi che si sono presentati durante lo svolgimento di questo lavoro.

Un grazie ai miei genitori per avermi dato la possibilità di studiare, supportando le mie scelte, e mettendomi in guardia da possibili errori. A mio fratello Stefano, da più tempo di me nell'ambiente universitario, va un ringraziamento per i consigli forniti soprattutto all'inizio degli studi.

Infine, non me ne vogliano coloro che sicuramente ho dimenticato di citare: anche a loro va un grazie di cuore!

Sommario

Questa tesi ha lo scopo di analizzare le problematiche relative alla composizione di servizi nell'ambito delle reti IP Multimedia Subsystem (IMS), e sulla base di tali considerazioni proporre un modello di architettura in grado di operare in tale contesto. Nella formulazione delle soluzioni verranno tenuti presenti i punti di forza delle piattaforme commerciali esistenti, in particolare Ericsson Composition Engine (ECE), e sarà sviluppato un componente software in grado di operare all'interno di una piattaforma proprietaria per l'orchestrazione di servizi. Questo componente dovrà consentire il riutilizzo del codice ed avere dunque quelle caratteristiche di modularità tipiche dell'architettura *Event-Driven SOA*, ovvero orientata ai servizi e basata sull'utilizzo di eventi.

L'importanza di un simile componente è suggerito dalla constatazione di alcuni trend presenti sia nell'ambito *telco* che nel settore IT: per gli operatori telefonici le reti di nuova generazione (e quindi IMS) rappresentano un obiettivo da realizzare nei prossimi anni, poiché potenzialmente in grado di garantire maggiori guadagni per sottoscrittore; per gli operatori del settore IT, invece, la composizione di servizi (perlopiù Web, realizzata attraverso i cosiddetti *mashup*) rappresenta un' interessante metodologia di sviluppo rivolta alla creazione di funzionalità sempre più sofisticate in maniera modulare. Per questo ha senso considerare una piattaforma per l'orchestrazione di servizi in grado di operare anche al di sopra della rete IMS: nella tesi verranno dapprima studiate le caratteristiche di tale rete, e sulla base di esse verranno proposte varie soluzioni al problema iniziale.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

La prima generazione di servizi Internet era destinata principalmente al trasporto di dati privi di specifiche di tempo reale o senza particolari esigenze in termini di QoS; tuttavia, negli ultimi anni tali caratteristiche sono diventate importanti, in seguito al diffondersi di tecnologie più raffinate e dotate di vincoli temporali stringenti (es. VoIP, streaming video, ecc). Inoltre, i servizi multimediali stanno diventando sempre più rilevanti anche per gli operatori del settore telefonico, dal momento che garantiscono una buona percentuale di guadagni, peraltro in grande crescita.

Non c'è quindi da stupirsi nell'osservare che le reti di telecomunicazione stiano evolvendo soprattutto a favore delle architetture basate su IP e le cosiddette Next Generation Networks (NGN), in grado di fornire una convergenza tra servizi e reti, nonché abilitare la fornitura di servizi sempre più ricchi, perlopiù multimediali.

Il concetto di NGN è nato negli anni '90 per tener conto della nuova situazione e dei cambiamenti in corso nel settore delle telecomunicazioni, quali l'apertura dei mercati, la crescente domanda da parte degli utenti di servizi innovativi che incontrassero le loro esigenze, l'esplosione del traffico digitale sostenuto dalla diffusione di Internet. L'impatto di questa nuova tipologia di reti sarebbe stato sia economico che tecnico: i nuovi servizi voce e dati basati sulle preferenze degli utenti (VoIP, presence, streaming, push-to-talk, ecc.) avrebbero infatti creato un aumento della produttività da parte degli operatori, ed allo stesso tempo una riduzione dei costi di manutenzione dell'infrastruttura, proprio perché la rete di trasporto sarebbe stata unica per tutte le tecnologie d'accesso. Per questo, l'obiettivo delle reti NGN è sempre stato quello di porre una separazione tra le funzioni necessarie al trasporto dei dati (*switching functions*) e la logica utilizzata dai servizi: sarebbe stato dunque necessario abbandonare l'architettura classica verticale, nella quale i meccanismi per l'accesso, il controllo e l'erogazione di servizi sono strettamente legati, in favore di un'architettura orizzontale,

dove ogni strato fornisce delle funzionalità comuni riutilizzabili dagli altri [1].

Ma con NGN si intende anche l'evoluzione verso una tipologia di rete che consenta il trasporto di tutte le informazioni ed i servizi (voce, dati, comunicazioni multimediali) a mezzo di pacchetti: nella maggior parte dei casi le reti di tipo NGN sono infatti basate sul protocollo IP. Tramite le NGN si vuole perseguire l'obiettivo di **Fixed-Mobile Convergence**, ovvero ciò che l'International Telecommunication Union (ITU) definisce come 'la capacità di fornire servizi applicativi agli utenti indipendentemente dalla loro locazione fisica e dalla rete di accesso (sia essa fissa o mobile)'.[2] Lo stesso

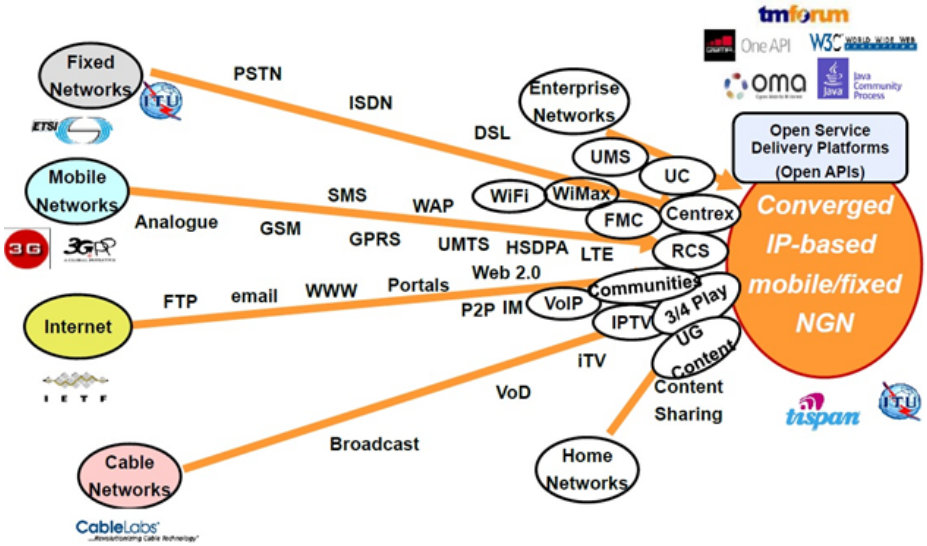


Figura 1.1: Convergenza fixed-mobile in una rete NGN IP-based

Vinton Cerf, uno dei cosiddetti 'padri di Internet', riferendosi alla migrazione della telefonia classica verso il VoIP, ha affermato che *'non ha senso mantenere due reti differenti se ci sono a disposizione sufficienti capacità e qualità per fonderle in un'unica rete'*[3]. Sulla base di questo, da un punto di vista architetturale le reti NGN implicano il consolidamento in un'unica struttura delle molteplici reti di trasporto nate storicamente per differenti tipologie di servizio; un esempio è dato dalla migrazione delle reti PSTN a commutazione di circuito, nate per la comunicazione voce, ad una tecnologia VoIP, basata sullo scambio di pacchetti IP. Caratteristiche fondamentali di una rete NGN, come prescritto da ITU, sono: [4]

- Trasferimento basato sui pacchetti;
- Separazione tra funzione di trasporto e fornitura di servizi;

- Supporto per una grande quantità di servizi ed applicazioni differenti;
- Supporto della banda larga e QoS;
- Mobilità generalizzata;
- Accesso libero degli utenti a provider diversi;
- Convergenza tra servizio fisso e mobile;
- Indipendenza delle funzioni legate ai servizi dalla rete sottostante;
- Compatibilità con la regolamentazione vigente, per quel che riguarda le chiamate d'emergenza, le politiche di sicurezza e privacy.

Il punto di svolta nell'ambito delle reti NGN è costituito dall'architettura **IP Multimedia Subsystem (IMS)**, standardizzata da 3GPP[1] e volta ad offrire agli operatori telefonici la possibilità di costruire un'infrastruttura aperta e basata sul protocollo IP che permetta la creazione di nuovi e più ricchi servizi di comunicazione multimediale, risultato dell'interazione fra i servizi classici (voce, SMS, ecc.) e quelli basati sulle connessioni dati. Lo standard facilita l'accesso alle applicazioni da qualsiasi terminale attraverso un strato di controllo comune, basato su protocollo SIP (Session Initiation Protocol, [5]), che isola la rete di accesso dallo strato dei servizi applicativi.

La formulazione originale di IMS (3GPP R5), rappresentava un approccio alla fornitura di servizi Internet su GPRS. Questa visione è stata poi modificata successivamente da altri enti, quali 3GPP2 e TISPAN [6], principalmente al fine di aggiungere il supporto anche per reti diverse da quella GPRS. Per consentire una facile integrazione con Internet, IMS utilizza il più possibile protocolli IETF [7], come il già citato SIP.

Entra a questo punto in gioco il concetto di **Service Delivery Platform (SDP)**: le SDP si svilupparono nell'ambito degli operatori telefonici come piattaforme per l'erogazione di servizi GSM/GPRS posizionate al di sopra della più complessa e variegata infrastruttura costituita dagli elementi proprietari della rete. Tale architettura può essere pensata come il punto di contatto tra le funzioni più di basso livello di una NGN (quindi controllo, abilitazione dei servizi, ecc.) e ciò che invece costituisce la struttura dei livelli superiori (elaborazione dei dati, storage, orchestrazione ecc.): in questo senso appare chiaro il forte legame con IMS, che costituisce la sezione più bassa della struttura.

Un tentativo di definire in maniera generale una SDP è stato compiuto dal TeleManagement Forum (TM Forum) [8]: 'con Service Delivery Platform

[1] *Third Generation Partnership Project*: accordo di collaborazione, formalizzato nel Dicembre 1998, fra enti che si occupano di standardizzare sistemi di telecomunicazione in diverse parti del mondo.

(SDP) ci si riferisce all'insieme delle componenti che forniscono un'architettura per l'erogazione di servizi (come la creazione del servizio, il controllo della sessione e dei protocolli) per un determinato tipo di servizio. Si tratta di un framework per la gestione di servizi di nuova generazione indipendente dalle tecnologie software e di rete utilizzate per implementare tali servizi.'

Dalla prospettiva SDP, IMS fornisce semplicemente un insieme di *service enablers*[2] che possono essere utilizzati per creare una vasta gamma di applicazioni multimediali di nuova generazione come pure servizi voce. Una SDP è dunque complementare per IMS nel senso che fornisce una serie di sistemi per l'erogazione, gestione ed orchestrazione di servizi che non appartengono propriamente alle specifiche IMS, ma che sono fondamentali all'interno di una NGN.

La tesi si focalizzerà proprio sullo studio di SDP che permettano di erogare servizi convergenti nel contesto delle reti IMS. Bisogna specificare che per **servizio convergente** si intende una funzionalità complessa costituita da più unità (ovvero servizi) autonomi, sviluppati con tecnologie diverse: a titolo d'esempio, si può considerare un servizio di trasferimento di chiamata che, consultando gli impegni di un utente attraverso il suo calendario online, effettua l'inoltro ad un numero differente a seconda di quanto vi è specificato, o ancora restituisce un particolare messaggio di avvertimento. In questo esempio il servizio convergente sarà dunque costituito da una *feature* telefonica, verosimilmente sviluppata attraverso la tecnologia SIP, interagente con un calendario online, quest'ultimo realizzato come Web Service. Questo primo esempio, per quanto semplice, mette in luce come sia possibile creare un numero di servizi convergenti potenzialmente infinito, a seconda di come vengono definite le interazioni tra i servizi di base.

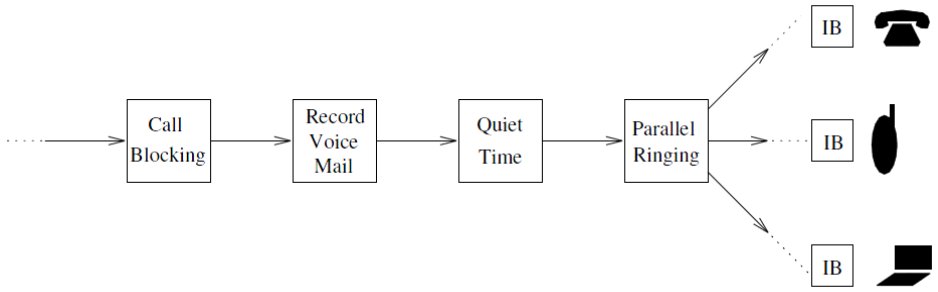


Figura 1.2: Un esempio di *Distributed Feature Composition*

La possibilità di far interagire più servizi tra di loro al fine di erogare una funzionalità più ricca non è un concetto nuovo: nel settore telefonico, la composizione di *feature* differenti, quali ad esempio trasferimento di

[2]Componente per la messa a disposizione a terze parti di particolari funzionalità.

chiamata, blocco selettivo e così via, è un meccanismo consolidato, che in letteratura viene generalmente indicato con i termini *feature interaction problem*. In questo contesto sono rilevanti gli studi condotti da M. Jackson e P. Zave, che in [9] analizzano il problema della crescita di funzionalità di *call-processing* e propongono una architettura per la composizione di *feature* telefoniche che prende il nome di *Distributed Feature Composition (DFC)* (figura 1.2). Ogni funzionalità costituisce un 'blocco', ed ognuno di questi può essere assemblato insieme ad altri in una catena: le comunicazioni tra vicini avvengono attraverso lo strato di collegamento sottostante, costituito dal canale fisico.

In ambito IT, invece, la composizione di servizi è identificata da un *mashup*: con questo termine si indica infatti una applicazione web ibrida, che include dinamicamente informazioni provenienti da più fonti, attraverso l'utilizzo di specifiche API. Il vantaggio è rappresentato dalla facilità con la quale è possibile progettare nuovi servizi, senza dover necessariamente disporre di sofisticate conoscenze tecniche: spesso, infatti, la creazione di un *mashup* avviene attraverso una interfaccia grafica nella quale i servizi sono semplici blocchi che lo sviluppatore può collegare a suo piacimento. Esistono già soluzioni di questo tipo, come Yahoo! Pipes [10], JackBe Presto [11] ed IBM Mashup Center [12].

L'idea di *mashup*, tipica del settore IT, unita alla tendenza degli operatori telco a migrare verso reti di nuova generazione, ha spinto a considerare la possibilità di creare servizi convergenti nel contesto della rete IMS. In particolare, questo obiettivo può essere perseguito attraverso due modelli differenti:

1. la piattaforma di composizione è *interna* alla rete IMS, ed in particolare sarà collocata nel livello dei servizi applicativi della rete, dove sarà rappresentata da un Application Server;

2. la rete IMS effettua *service exposure*, cioè fornisce una serie di servizi che l'utilizzatore finale può comporre a suo piacimento: in questo caso, dunque, la piattaforma per la composizione di servizi sarà un componente *esterno* alla rete IMS.

Nella tesi verrà considerata l'architettura di tipo (1) (si veda Fig. 1.3), pertanto la piattaforma di composizione costituirà un Application Server di IMS: tale scelta è ragionevole dal momento che lo standard IMS (che verrà descritto nel dettaglio nel Capitolo 2) prevede la presenza di un livello di servizi applicativi, che comunica con il sottostante livello di controllo attraverso delle interfacce e dei protocolli predefiniti. A questo proposito, anche Ericsson ha deciso di adottare questo modello architetturale nella sua piattaforma per la composizione di servizi, chiamata Ericsson Composition Engine [13], che costituisce lo stato dell'arte in questo contesto.

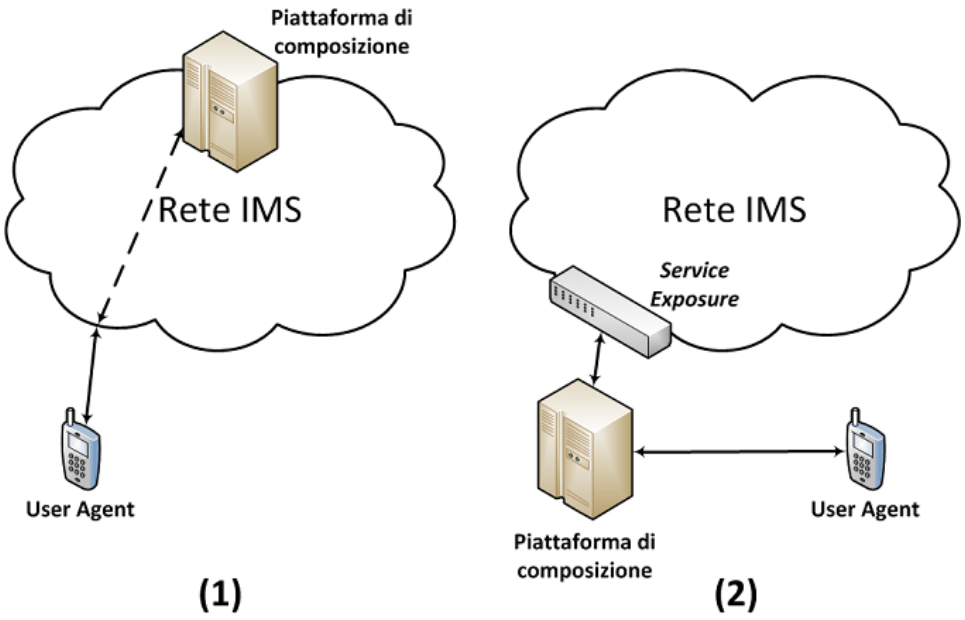


Figura 1.3: Piattaforma di composizione e IMS: interna (1) o esterna (2)

Nella trattazione seguente verranno dunque esaminate le caratteristiche di IMS nonché quelle delle soluzioni proposte fino ad oggi per il problema della composizione di servizi: esse costituiranno il punto di partenza per poter elaborare un nuovo modello architetturale destinato all'orchestrazione di servizi che operi in maniera dinamica tanto su *feature* telefoniche, quanto su servizi generici. Lo studio si concentrerà a livello di *Service Execution Environment*, quindi nell'ambito della piattaforma software che effettua l'esecuzione dei servizi nonché l'interazione con il livello di controllo della rete IMS: non verrà dunque considerato il sistema grafico per la creazione di servizi convergenti, che permette di assemblare le funzionalità di base in maniera semplice e limitata solo dalla creatività dell'utente.

Questo obiettivo richiederà lo sviluppo di un componente in grado di interfacciarsi sia con la rete IMS (quindi mediante il protocollo SIP) che con la piattaforma per l'orchestrazione, il tutto nell'ottica di interoperabilità tra il mondo *telco* e quello dei servizi IT. L'esigenza di un tale componente è dettata dalla volontà di superare i limiti imposti dall'utilizzo della tecnologia SIP Servlet, che non permette una realizzazione agevole di una piattaforma per la composizione di servizi dotata di caratteristiche dinamiche quali la selezione di servizi di base in fase di esecuzione, basata sul verificarsi di particolari eventi e dipendente dagli output generati. La sua creazione avverrà considerando i punti di forza e le debolezze delle tecnologie utilizzabili per lo sviluppo, quindi in particolare delle librerie JAIN SIP [14] e dello standard

Sip Servlet 1.1 [15], le più utilizzate in questo contesto.

Il lavoro è stato organizzato nei seguenti capitoli:

- **Capitolo 1**: vengono presentate le motivazioni che hanno spinto a considerare lo sviluppo del componente software descritto in precedenza;
- **Capitolo 2**: viene offerta una descrizione delle principali caratteristiche dell'architettura IP Multimedia Subsystem e del protocollo SIP, sul quale si basa lo strato di controllo della rete;
- **Capitolo 3**: vengono descritte le componenti di Open IMS Core, *testbed* utilizzato per simulare una rete IMS. Inoltre è fornita una descrizione dettagliata delle procedure necessarie alla sua installazione e configurazione;
- **Capitolo 4**: si fornisce un'analisi più dettagliata dello strato dei servizi applicativi della rete IMS, in quanto su di esso opererà la piattaforma per l'orchestrazione dei servizi. In particolare, viene dato risalto all'interazione tra il componente S-CSCF appartenente al *core* della rete e gli Application Server; degli ultimi, componenti fondamentali dello strato applicativo, viene data una categorizzazione a seconda della tecnologia utilizzata;
- **Capitolo 5**: viene presentata l'architettura *Event-Driven*, sulla quale si basa la piattaforma per l'orchestrazione di servizi utilizzata nel seguito. Viene descritto il funzionamento di *Ericsson Composition Engine*, promettente soluzione per l'orchestrazione di servizi con la quale la piattaforma proprietaria utilizzata per i test condivide molte caratteristiche;
- **Capitolo 6**: vengono presentate le problematiche relative ai conflitti tra servizi operanti in una rete IMS. Inoltre vengono presentati vari modelli architetturali utilizzabili per la composizione di servizi in questo contesto;
- **Capitolo 7**: si offre la descrizione del componente software sviluppato per poter interagire con IMS e la piattaforma proprietaria per la composizione di servizi. Tale descrizione è corredata dal codice JAVA delle sezioni più importanti;
- **Capitolo 8**: vengono formulate le conclusioni alla tesi e si analizzano i possibili sviluppi di questo lavoro.

Capitolo 2

IP Multimedia Subsystem (IMS)

Indice

2.1 Introduzione

Come già accennato, l'IP Multimedia Subsystem (IMS) è un'architettura funzionale di rete che costituisce una promettente soluzione per facilitare la creazione ed erogazione di servizi multimediali, così come il supporto all'interoperabilità tra di essi e la convergenza delle differenti tipologie di rete. IMS permette poi agli operatori di rete di giocare un ruolo centrale nella distribuzione del traffico: per questi motivi nei confronti di IMS si sono rivolti intensi sforzi sia nel campo della ricerca che della standardizzazione.

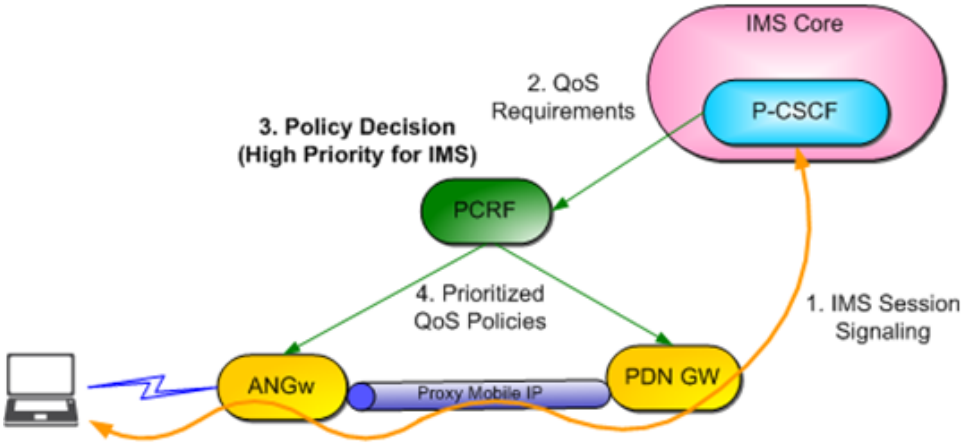
2.2 Principi di base

Uno degli obiettivi di IMS è rendere più facile la gestione della rete. Pertanto, esso separa le funzioni di controllo da quelle di trasporto: ciò significa che IMS realizza un servizio di trasporto come overlay di una infrastruttura a commutazione di pacchetto. Inoltre IMS deve consentire la migrazione di servizi basati sulla commutazione di circuito (Circuit Switched - CS) come la comunicazione voce verso il dominio della commutazione di pacchetto (Packet Switched - PS). Come risultato, IMS porta ad una semplificazione della gestione di rete, dal momento che una struttura basata solo su IP è più facilmente amministrabile.

IMS è un'architettura end-to-end che deve supportare vari tipi di apparecchiature; inoltre IMS deve essere 'access agnostic' ovvero il servizio erogato deve essere indipendente dalla tecnologia d'accesso sottostante adoperata, ed in questo contesto appare ovvia la scelta del protocollo IP, dotato delle più ampie caratteristiche di compatibilità. A questo proposito può essere necessaria l'adozione di tecnologie che permettano di accedere alle funzionalità di IMS mediante delle interfacce standard: un esempio, appartenente al settore delle telecomunicazioni mobile, è rappresentato dall'Evolved Packet Core (EPC). Si tratta di una serie di specifiche, definite da 3GPP, volte alla creazione di una piattaforma di controllo della connettività IP per le tecnologie d'accesso wireless, incluse quelle più moderne appartenenti agli standard Long Term Evolution (LTE) [16]. EPC costituisce dunque il nucleo del sistema di controllo che permette a tecnologie diverse, quali WiMax, CDMA2000, WiFi, 3G, di usufruire dei servizi messi a disposizione dalle reti sovrastanti, facendo uso di protocolli standard IETF. In questo senso EPC risulta essere uno strato intermedio tra le reti d'accesso di livello più basso e il livello più alto dei servizi applicativi, la cui gestione può essere delegata a strutture differenti quali IMS, Internet, ecc.

Un'implementazione software dei principi esposti da 3GPP per EPC è stata eseguita dall'istituto Fraunhofer FOKUS, con il prodotto open source OpenEPC [17]. Un esempio di interazione tra EPC e la rete IMS è illustrato nella figura seguente, dove un dispositivo dotato di connettività wireless che

ha effettuato la registrazione alla rete IMS ottiene l'allocazione delle risorse necessarie alla fruizione del servizio dalla struttura EPC. Tornando ad IMS,



il livello di **QoS** è critico all'interno di una rete di questo tipo, dal momento che determina il tipo di servizi che vi possono essere integrati; come conseguenza, le funzionalità di gestione della QoS devono essere inserite all'interno dell'architettura IMS. Bisogna infatti ricordare che una rete basata sulla commutazione di pacchetti è di tipo best-effort, e pertanto la presenza di un sistema per regolare la QoS è di importanza primaria per garantire dei livelli adeguati di performance.

L'architettura IMS è poi *orizzontale*: fornisce cioè una serie di funzioni comuni chiamate 'service enablers' che possono essere utilizzate da svariati servizi (ad esempio la gestione dei gruppi, la supervisione, il billing, ecc.); in questo modo l'implementazione dei servizi diventa più semplice e veloce. Inoltre, ciò consente una stretta interazione tra servizi differenti. Si tratta quindi di una caratteristica apprezzabile, soprattutto se confrontata con le altre architetture a disposizione, basate perlopiù su di un'implementazione dei servizi rigida e verticale (architetture 'stovepipe') (si veda la Figura 2.1): queste ultime, infatti, presentano dei vantaggi dal punto di vista della rapidità di sviluppo di applicazioni poiché focalizzate in un determinato ambito, ma si rivelano essere svantaggiose qualora si desiderino integrare due o più servizi (a causa della duplicazione necessaria di unità funzionali). Riassumendo, le motivazioni che hanno portato alla progettazione dell'architettura IMS da parte di 3GPP sono:

- Avvalersi del paradigma dell'Internet Mobile;

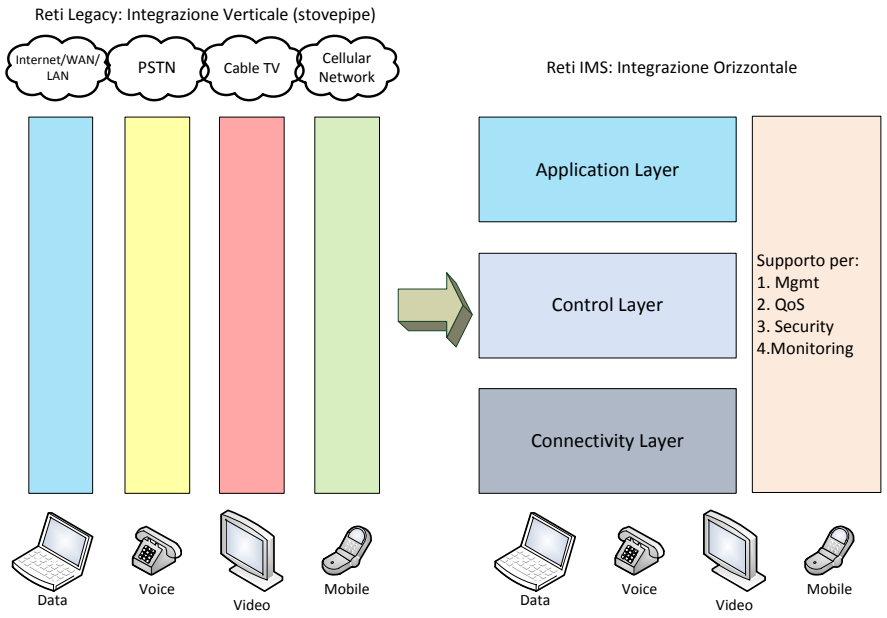


Figura 2.1: Differenze tra un'architettura legacy (orizzontale) e una IMS (verticale)

- Creare una piattaforma comune per sviluppare diversi servizi multimediali;
- Creare un meccanismo per dare un impulso ad un maggior utilizzo di cellulari compatibili con reti a commutazione di pacchetto.

Tali obiettivi sono quindi perseguibili mediante:

- Supporto alla creazione di sessioni multimediali su IP;
- Supporto per un meccanismo di negoziazione della Quality of Service (QoS);
- Supporto per l'interazione con Internet e le reti a commutazione di circuito;
- Supporto per il roaming;
- Supporto per il controllo imposto dall'operatore nei servizi forniti all'utente finale;
- Supporto per la rapida creazione di servizi senza necessità di standardizzazione.

2.3 Architettura

L'architettura di rete di IMS è divenuta standard grazie alla collaborazione dei già citati TISPAN e 3GPP, commissioni istituite dall'organo di standardizzazione European Telecommunications Standard Institute (ETSI). In Figura 2.2 è rappresentata una rete IMS, ed in particolare è evidenziata la presenza di 3 strati (layer):

- *Transport Layer* o **Livello di Accesso**;
- *IMS Layer* o **Livello di Controllo**;
- *Service/Application Layer* o **Livello dei Servizi Applicativi**.

Scopo di tale suddivisione è isolare il livello di accesso da quello applicativo: da un punto di vista logico-architetturale i servizi non hanno più bisogno di un sistema ad-hoc per controllare le chiamate, dal momento che è lo strato di controllo comune che fornisce queste funzioni, *indipendentemente dalla rete di accesso*. Ogni strato è costituito da diverse funzioni, che comunicano tra di loro attraverso delle interfacce standardizzate: questa logica ricalca la definizione di IMS data da 3GPP, in cui si fa appunto riferimento ad un insieme di funzioni ed interfacce standard piuttosto che un insieme di nodi.[19] Per questo, una funzione non è necessariamente un nodo fisico della rete: l'implementatore è libero di combinare più funzioni in un unico nodo, oppure dividere una singola funzione in 2 o più nodi. Ciascun nodo può anche essere presente più volte in una singola rete per motivi di dimensionamento, bilanciamento del carico o di organizzazione. Per facilitare l'integrazione con Internet, IMS utilizza il più possibile protocolli IETF (Internet Engineering Task Force): non a caso il **protocollo SIP (Session Initiation Protocol)** costituisce il fulcro di IMS.

2.3.1 Livello di Accesso

L'utente può connettersi ad una rete IMS in vari modi, molti dei quali utilizzano lo standard IP. I terminali IMS (così come cellulari, PDA e computer) possono registrarsi direttamente sulla rete IMS, anche se sono in roaming in un'altra rete di accesso o paese. Unico prerequisito è che essi siano in grado di utilizzare il protocollo IP ed attivare un agente SIP. Gli accessi da rete fissa (DSL, modem, Ethernet, ecc.) o da rete mobile (GSM, GPRS, ecc.) o wireless (WLAN, WiMAX, ecc.) sono tutti ugualmente supportati. Altri sistemi, come POTS (Plain Old Telephone Service - tecnologia di base per i servizi telefonici analogici) oppure quelli non compatibili con il VoIP di IMS sono supportati attraverso specifici gateway.

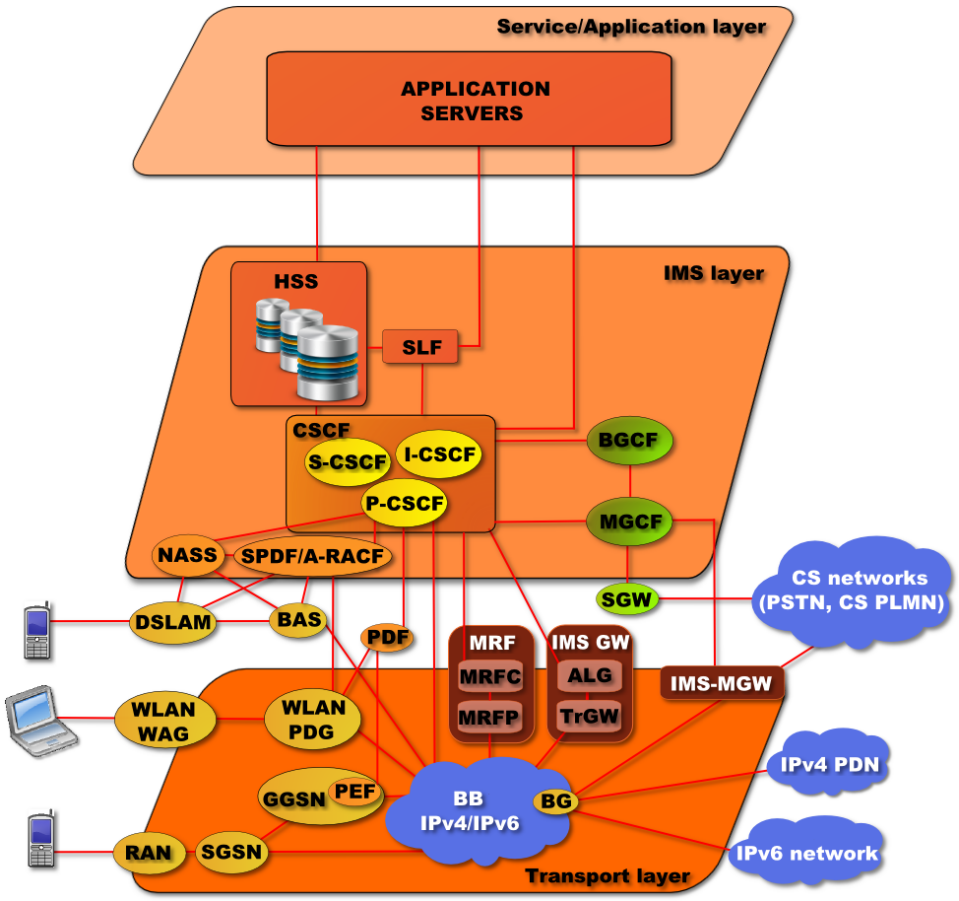


Figura 2.2: I tre strati che compongono l'architettura di IMS

2.3.2 Livello di Controllo

Il nucleo dell'architettura IMS è rappresentato da questo livello, che difatti viene spesso identificato con il termine **IMS Core**. Il livello di controllo deve svolgere essenzialmente due funzioni principali:

- Gestire le identità degli utenti della rete;
- Gestire le sessioni e le chiamate (incluso il controllo dei media).

Tali obiettivi vengono realizzati da due componenti distinte, dette rispettivamente Home Subscriver Server (HSS) e Call Session Control Function (CSCF), alle quali si affiancano il Media Resource Function (MRF), Breakout Gateway Control Function (BGCF) e Media Gateway Control Function (MGCF).

Home Subscriber Server (HSS)

L'HSS è un repository centralizzato contenente tutte le informazioni riguardanti gli utenti (User Profile), e tecnicamente può essere pensato come l'evoluzione dell'*Home Location Register (HLC)*, un database presente nell'architettura GSM che contiene informazioni sugli abbonati. Le informazioni presenti in un HSS, necessarie alla gestione delle sessioni multimediali, comprendono:

- Informazioni sulla regione in cui si trova l'abbonato;
- Informazioni sulla sicurezza, come dati di autenticazione e di autorizzazione;
- Informazioni sul profilo utente, inclusi i servizi ai quali l'utente è abbonato;
- Informazioni sul Serving-CSCF (vedi seguito) dell'utente.

Una rete può contenere più di un HSS, qualora il numero di utenti sia elevato oppure nell'ottica di un'architettura ridonante che possa far fronte ad eventuali guasti. Ad ogni modo, tutti i dati relativi ad uno specifico utente sono memorizzati in un unico HSS. Per poter gestire più HSS, c'è quindi bisogno di un **Subscriber Location Function (SLF)**, che ha il compito di mappare gli indirizzi degli utenti ai relativi HSS: cioè, un nodo che necessita delle informazioni relative ad un determinato utente invia il suo indirizzo all'SLF, il quale cercherà nel suo database interno l'indirizzo dell'HSS corretto e lo restituirà al nodo richiedente.

Call Session Control Funcion (CSCF)

Per quanto riguarda invece la gestione delle sessioni e delle chiamate, si fa riferimento al Call Session Control Function (CSCF), cioè un server SIP che elabora le segnalazioni basate su tale protocollo all'interno di IMS. Qualsiasi CSCF può appartenere ad una delle seguenti categorie:

- P-CSCF (Proxy-CSCF);
- I-CSCF (Interrogating-CSCF);
- S-CSCF (Serving-CSCF).

In Figura 2.3 sono rappresentate, in maniera schematica, le interazioni tra le componenti dei diversi livelli della rete IMS, con particolare attenzione ai CSCF. Un **Proxy-CSCF** è un proxy server SIP, e rappresenta il primo punto di contatto tra i terminali e la rete IMS. Il P-CSCF è assegnato ad un terminale in fase di registrazione, e tale assegnazione non cambia per tutta la durata della registrazione. Le principali funzioni di un P-CSCF sono:

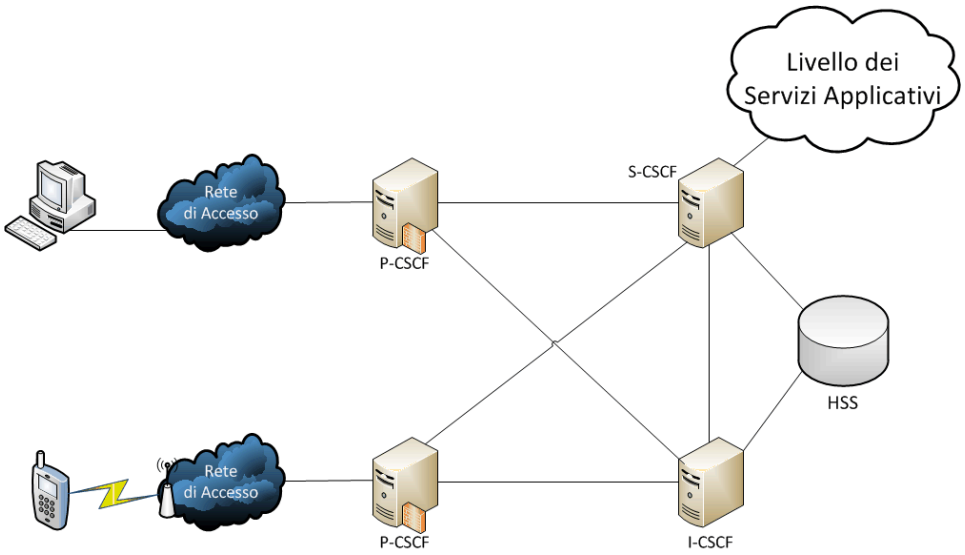


Figura 2.3: Interazioni tra i diversi CSCF e l'HSS

- Accettazione e inoltro delle richieste di sessione SIP;
- Autenticazione dell'utente tramite IPSec[1];
- Compressione/Decompressione di messaggi SIP;
- Generazione dei dati di tariffazione.

Un **I-CSCF** è invece un proxy server SIP collocato sul bordo del dominio di amministrazione ed ha il compito di interrogare il database HSS (o SLF qualora vi siano più HSS) al fine di conoscere ed inoltrare ad altri componenti della rete IMS le informazioni relative agli abbonati. Esso costituisce poi il punto d'ingresso per tutte le chiamate provenienti da altri domini amministrativi: difatti il suo indirizzo IP è pubblicato nel DNS (Domain Name Server) del dominio amministrativo, cosicché i server remoti possano trovarlo ed utilizzarlo come un punto di inoltro dei pacchetti SIP. Fino alla Release 6 di IMS poteva essere utilizzato per nascondere la rete interna al mondo esterno (ed era quindi chiamato Topology Hiding Internetwork Gateway (THIG)), ma dalla release 7 in poi questa funzione è stata rimossa dall'I-CSCF e aggiunta all'Interconnection Border Control Function (IBCF), che fornisce servizi di NAT e firewall.

Infine, il **S-CSCF** costituisce la parte più importante dell'IMS Core: esso è un SIP server che effettua il controllo delle sessioni attive, ed è sempre

[1]IP Security, standard definito negli RFC 2401-2412 il cui scopo è ottenere connessioni basate su reti IP sicure.

posizionato all'interno della Home Network[2]. Un S-CSCF dispone, in aggiunta alle funzionalità si server SIP, anche quelle di SIP-Registrar, ovvero mantiene il legame tra la posizione dell'utente (l'indirizzo IP del terminale) e il suo indirizzo SIP (la cosiddetta Public User Identity). Il S-CSCF dialoga con l'HSS al fine di:

- Scaricare ed utilizzare i vettori di autenticazione dell'utente;
- Scaricare lo User Profile: quest'ultimo include il Service Profile, che è un insieme di trigger che possono causare l'instradamento di un messaggio SIP verso uno o più Application Server;
- Informare l'HSS su quale S-CSCF è stato assegnato all'utente per la durata della registrazione.

Dopo la fase di registrazione, il S-CSCF interagisce con gli Application Server per l'eventuale erogazione di servizi richiesti dall'utente e, se i messaggi di segnalazione sono destinati ad un'altra rete, scopre l'indirizzo del relativo I-CSCF e vi inoltra i messaggi. Infine applica la tariffazione suggerita per il particolare utente tramite la generazione dei cosiddetti Call Data Record (CDR). Vi possono comunque essere più S-CSCF in una rete IMS, al fine di bilanciare il carico o per la garanzia di affidabilità. Va sottolineato che è l'HSS ad assegnare uno specifico S-CSCF all'utente nel momento in cui il primo viene interrogato dall'I-CSCF.

Media Resource Function (MRF)

Il **Media Resource Function (MRF)** fornisce un supporto alla gestione e manipolazione delle risorse all'interno di IMS. Esso è suddiviso in due unità:

- Media Resource Function Controller (MRFC), che controlla le risorse per i flussi multimediali nel MFRP (vedi punto successivo) e interpreta le informazioni provenienti dal S-CSCF o dall'Application Server per il trattamento dei media;
- Media Resource Function Processor (MRFP), che controlla il trasporto sull'interfaccia di collegamento con il S-CSCF, determina il mixing degli stream multimediali ricevuti e applica l'elaborazione richiesta (ad es. transcodifica o analisi dei media).

[2]Questo significa che, in caso di roaming, il S-CSCF che gestisce la sessione di un utente è sempre quello della sua Home Network, mentre le risorse adoperate sono quelle della rete fisica in cui esso si trova. Tale principio (tipico di IMS) è detto *'Home Control of Services'*.

La combinazione di tali operazioni di transcodifica dei flussi media con il supporto della rete alla QoS, permette ad IMS di regolare in maniera dinamica la quantità di banda ottimale da dedicare ad ogni servizio nell'ipotesi di una loro esecuzione concorrente. [3]

Breakout Gateway Control Function (BGCF)

Il **Breakout Gateway Control Function (BGCF)** svolge funzioni di controllo nell'interazione con reti a commutazione di circuito: esso viene utilizzato solo in sessioni inizializzate da terminali IMS e dirette ad un utente che appartiene ad una rete Public Switched Telephone Network (PSTN) oppure Public Land Mobile Network (PLMN) (vedi Figura 2.4). Principali funzionalità di BGCF sono dunque:

- selezione della rete a commutazione di circuito appropriata, qualora ve ne sia necessità;
- selezione del PSTN Gateway (appartenente alla rete IMS) appropriato: nella Figura 2.4 esso è logicamente equivalente alle tre componenti SGW, MGCF, MGW.

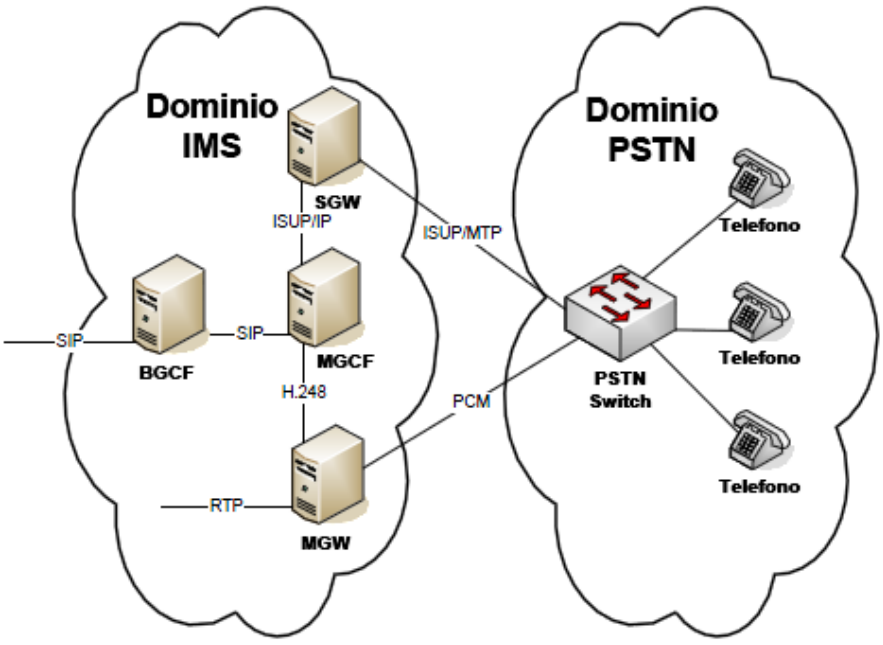


Figura 2.4: Interazione tra la rete IMS e una rete PSTN

Media Gateway Control Function (MGCF)

Il **Media Gateway Control Function (MGCF)** è il nodo centrale del PSTN Gateway: scopo di tale componente è la conversione da protocollo SIP (in uso nella rete IMS) a protocollo ISUP o BICC (protocolli in uso nelle reti a commutazione di circuito), nonché il controllo delle risorse del Media Gateway (MGW).

2.3.3 Livello dei servizi applicativi

Il livello dei servizi applicativi è costituito fondamentalmente dai vari Application Server (AS), ovvero le entità che si occupano di ospitare ed eseguire i servizi, interfacciati con il S-CSCF mediante il protocollo SIP. Ci possono essere 3 tipi di Application Server:

- **SIP AS**: è il nativo Application Server che ospita ed esegue servizi multimediali IP, basato su SIP
- **OSA-SCS (Open Server Access Service Capability Server)**: l'Application Server fornisce un'interfaccia per il framework OSA Application Server. OSA è una collezione di API che consentono a terze parti di sviluppare ed implementare applicazioni che accedono alle piene funzionalità della rete sottostante preservando l'integrità di quest'ultima. Da un lato il nodo agisce come Application Server interfacciandosi con il S-CSCF tramite il protocollo SIP; dall'altro si comporta come un'interfaccia tra l'OSA Application Server e l'OSA API. [20]
- **IM-SSF (IP Multimedia Service Switching Function)**: l'Application Server permette di utilizzare i servizi CAMEL (Customized Applications for Mobile network Enhanced Logic) ovvero le applicazioni sviluppate per reti GSM all'interno delle reti IMS. Esso alloca un GsmSCF (GSM Service Control Function) per controllare una sessione IMS. L'IM-SSF agisce come Application Server su un lato, quindi interfacciandosi con il S-CSCF tramite il protocollo SIP; sull'altro lato agisce invece come SSF (Service Switching Function), interfacciandosi con il GsmSCF mediante il protocollo basato su CAP (CAMEL Application Part [21]).

In Figura 2.5 sono rappresentate le tre tipologie di Application Server appena descritte, con l'indicazione dei nomi delle interfacce come da standard 3GPP (si veda 2.4 più sotto) e la specifica di utilizzo del protocollo SIP o DIAMETER. Esempi importanti di Application Server sono:

- **Presence Server**: permette agli utenti di impostare il proprio stato personale e di conoscere lo stato di attività degli Application Server nella rete IMS. In particolare, un utente invia il suo stato al Presence

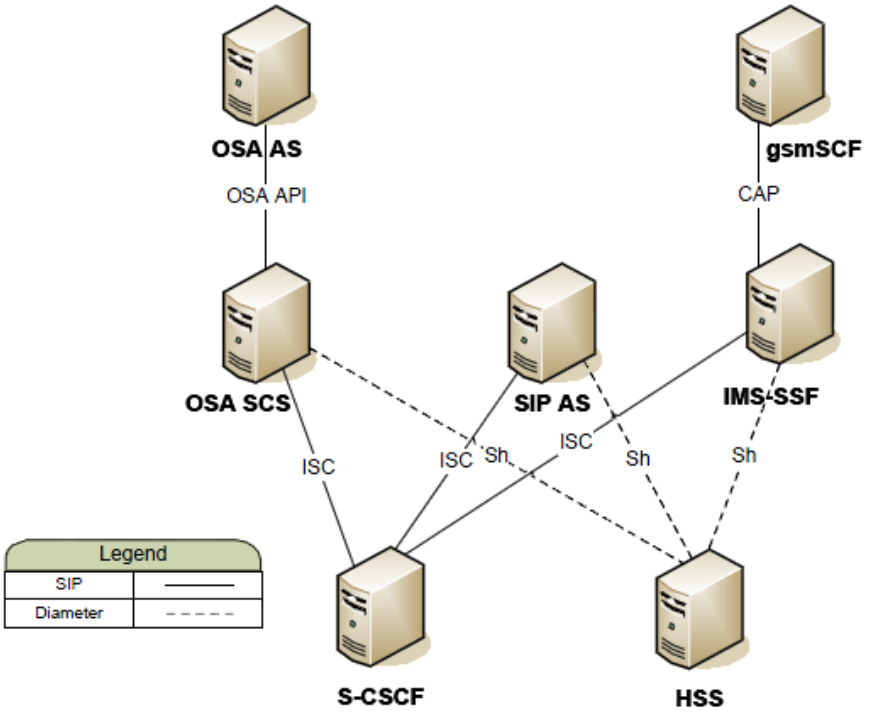


Figura 2.5: Le tre tipologie di Application Server e le interfacce collegate

Server, il quale si occupa di rendere nota tale informazione ai partecipanti del gruppo dell'utente. L'utilizzo di un tale server permette anche di inoltrare al rispettivo destinatario messaggi temporaneamente memorizzati, qualora fosse stato offline al momento dell'invio

- **B2BUA (Back-to-Back User Agent)**: un B2BUA opera tra i due end-point di una comunicazione, dividendola in due sezioni ed operando come gestore della segnalazione SIP sia sul chiamante che sul chiamato. A differenza di un semplice proxy server SIP, (che contiene solo lo stato della transazione in corso) il B2BUA contiene lo stato di tutte le chiamate gestite, e per questo motivo può fornire funzionalità di tariffazione, trasferimento di chiamata, ecc.

- **Instant Messaging**: per Instant Messaging si intende un sistema di comunicazione (generalmente di tipo client-server) che consente di scambiare messaggi di testo o altro contenuto tra due computer connessi alla rete. Generalmente esiste un server centrale al quale vengono inviate le richieste di connessione ad altri client: la sua funzio-

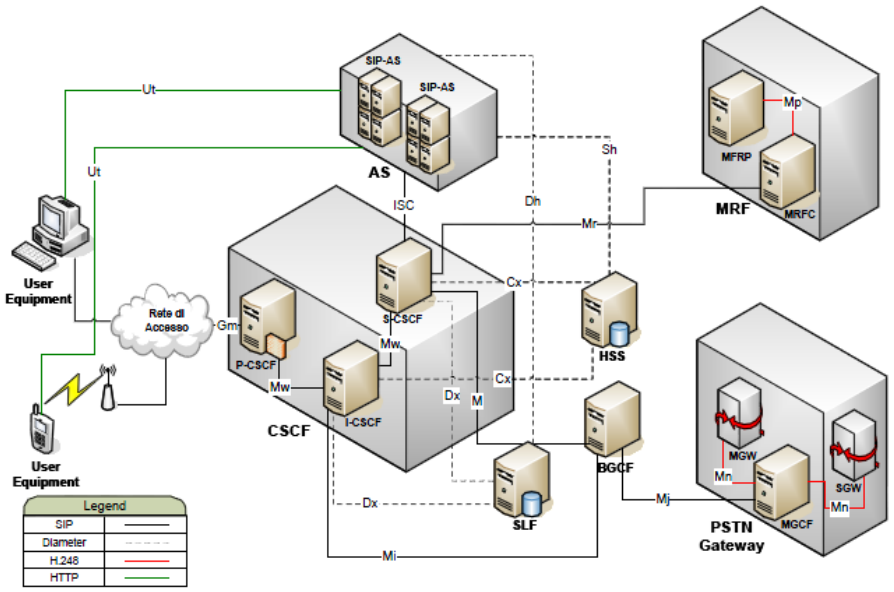


Figura 2.6: Principali componenti di una rete IMS con le indicazioni delle interfacce standard

ne è quindi tener traccia degli utenti connessi al servizio e gestire la comunicazione tra essi.

2.4 Protocolli

Come accennato, i protocolli utilizzati nell'architettura IMS sono il più possibile basati su standard IETF. Essi possono essere suddivisi in 3 grandi categorie:

- Protocolli per il Controllo della Sessione;
- Protocolli AAA;
- Altri Protocolli.

Grazie a questi protocolli, IMS può gestire tutte le fasi della comunicazione, dal riconoscimento e autenticazione dell'utente fino al trasferimento di flussi di dati. In Figura 2.6 è presente lo schema delle interfacce utilizzate in IMS, mentre nella Tabella 2.1 la descrizione dei loro utilizzi e dei protocolli usati.

Nome Interfaccia	Entità collegate	Descrizione	Protocollo
Gm	UE - P-CSCF	Scambio di messaggi tra UE e P-CSCF	SIP
Mw	P-CSCF - S-CSCF	Scambio di messaggi tra i CSCF	SIP
Mw	I-CSCF - S-CSCF	cambio di messaggi tra i CSCF	SIP
ISC	S-CSCF - AS	Scambio di messaggi tra S-CSCF e Application Server	SIP
Mg	MGCF - I-CSCF	Conversione tra messaggi SIP e ISUP	SIP
Mg	MGCF - S-CSCF	Conversione tra messaggi SIP e ISUP	SIP
Mi	BGCF - S-CSCF	Scambio di messaggi tra BGCF e S-CSCF	SIP
Mj	MGCF - BGCF	Scambio di messaggi SIP tra BGCF e MGCF appartenenti alla stessa rete IMS	SIP
Mk	BGCF - BGCF	Scambio di messaggi SIP tra BGCF appartenenti a due reti IMS diverse	SIP
Mr	S-CSCF - MRFC	Scambio di messaggi tra S-CSCF e MRFC	SIP
Mp	MRFC - MRFP	Permette all'MRFC di controllare le risorse fornite dall'MRFP	H.248
M	S-CSCF - BGCF	Scambio di messaggi tra S-CSCF e BGCF	SIP
Mn	MGCF - SGW	Gestione delle risorse	H.248
Mn	MGCF - MGW	Gestione delle risorse	H.248
Cx	S-CSCF - HSS	Scambio di messaggi tra l'HSS e l'S-CSCF	DIAMETER
Cx	I-CSCF - HSS	Scambio di messaggi tra l'HSS e l'I-CSCF	DIAMETER
Dx	S-CSCF - SLF	Scambio di messaggi tra S-CSCF e SLF per la scelta dell'HSS	DIAMETER
Dx	I-CSCF - SLF	Scambio di messaggi tra I-CSCF e SLF per la scelta dell'HSS	DIAMETER
Sh	HSS - AS	Scambio dei dati relativi agli utenti e attivazione di eventuali iFC	DIAMETER
Dh	SLF - AS	Scambio di messaggi tra AS e SLF per la scelta dell'HSS	DIAMETER
Gq	P-CSCF - PDF	Scambio di messaggi per il controllo della QoS	DIAMETER
Go	PDF - PEP	Scambio di messaggi per il controllo della QoS tra reti IMS e GPRS	DIAMETER
Ut	UE - AS	Scambio di messaggi per la gestione dei propri servizi tramite accesso HTTP	HTTP

Tabella 2.1: Interfacce di IMS

2.4.1 Protocolli per il controllo della sessione

In tutti i sistemi di telefonia, sono fondamentali i protocolli che gestiscono le chiamate attraverso opportuni messaggi di segnalazione. Per raggiungere questo scopo in IMS, 3GPP ha scelto **SIP (Session Initiation Protocol)**, un protocollo basato su IP e definito dalla RFC 3261, già impiegato largamente par applicazioni di telefonia su IP di tipo VoIP.

Panoramica del protocollo SIP

SIP gestisce in modo generale una sessione di comunicazione tra due o più entità, cioè fornisce meccanismi per instaurare, modificare e terminare una sessione; attraverso il protocollo SIP possono essere trasferiti dati di diverso tipo (audio, video, messaggistica, ecc.). SIP favorisce poi un'architettura modulare e scalabile: queste caratteristiche hanno fatto sì che SIP sia, oggi, il protocollo VoIP più diffuso nel mercato residenziale e business, sorpassando altri protocolli quali H.323 e MGCP. In particolare, rispetto ad H.323, SIP risulta meno complesso e necessita di una quantità inferiore di risorse. Gli sviluppatori di SIP hanno preso in prestito i princìpi di progettazione di Simple Mail Transfer Protocol (SMTP, [22]) e di HyperText Transfer Protocol (HTTP, [23]), in quanto protocolli tra i più usati su Internet; ad esempio, SIP utilizza un modello di sintassi text-based, proprio come avviene in HTTP. Le principali funzioni del protocollo SIP sono:

- Localizzazione degli utenti
  - acquisire le preferenze degli utenti;
- Invitare gli utenti a partecipare ad una sessione
  - negoziare le capability;
  - trasportare una descrizione della sessione;
- Instaurare le connessioni di sessione;
- Gestire eventuali modifiche dei parametri di sessione;
- Rilasciare le parti;
- Cancellare la sessione in qualunque momento si desideri.

Per instaurare una sessione SIP avviene un three-way handshaking, concettualmente simile a quello adoperato nell'ambito TCP. Inoltre, importanti caratteristiche del protocollo SIP sono:

- Utilizzo sia in contesti client-server che peer-to-peer;
- Facile da estendere e programmare;

- I server possono essere di tipo stateless oppure stateful[3];
- Indipendenza dal protocollo di trasporto.

Entità operanti nel protocollo SIP

Le entità essenziali per il protocollo SIP sono:

- **SIP User Agent**: è un end point della comunicazione. Può fungere sia da server che da client, dal momento che nel corso di una sessione i ruoli sono interscambiabili: nel primo caso è l'entità che ascolta le richieste e, se possibile, le soddisfa; nel secondo caso dà inizio alla transazione originando le richieste
- **Registrar Server**: è un server dedicato o collocato in un proxy. Quanto un utente è iscritto ad un dominio invia un messaggio di registrazione del suo attuale punto di ancoraggio nella rete al Registrar Server
- **Proxy Server**: può rispondere direttamente alle richieste oppure inoltrarle ad client, ad un server o eventualmente ad un altro proxy. Esso analizza i parametri di instradamento dei messaggi e nasconde la reale posizione dei destinatari all'interno della rete.

Nella Figura 2.7 è raffigurato il flusso dei messaggi per l'instaurazione di una sessione SIP tra due entità (User A ed User B):

1. L'utente A chiama l'utente B;
2. Il Proxy Server SIP contatta il Registrar Server per ottenere la locazione dell'utente B;
3. La chiamata viene inoltrata all'utente B;
4. L'utente B risponde;
5. La risposta dell'utente B viene inoltrata all'utente A;
6. Viene stabilito il canale di comunicazione tra A e B.

[3]Un'entità si dice *stateful* se mantiene traccia delle transazioni client-server durante l'elaborazione della richiesta; viceversa si dice *stateless* se non mantiene traccia della transazione ed effettua solo operazioni di inoltro e instradamento della segnalazione SIP.

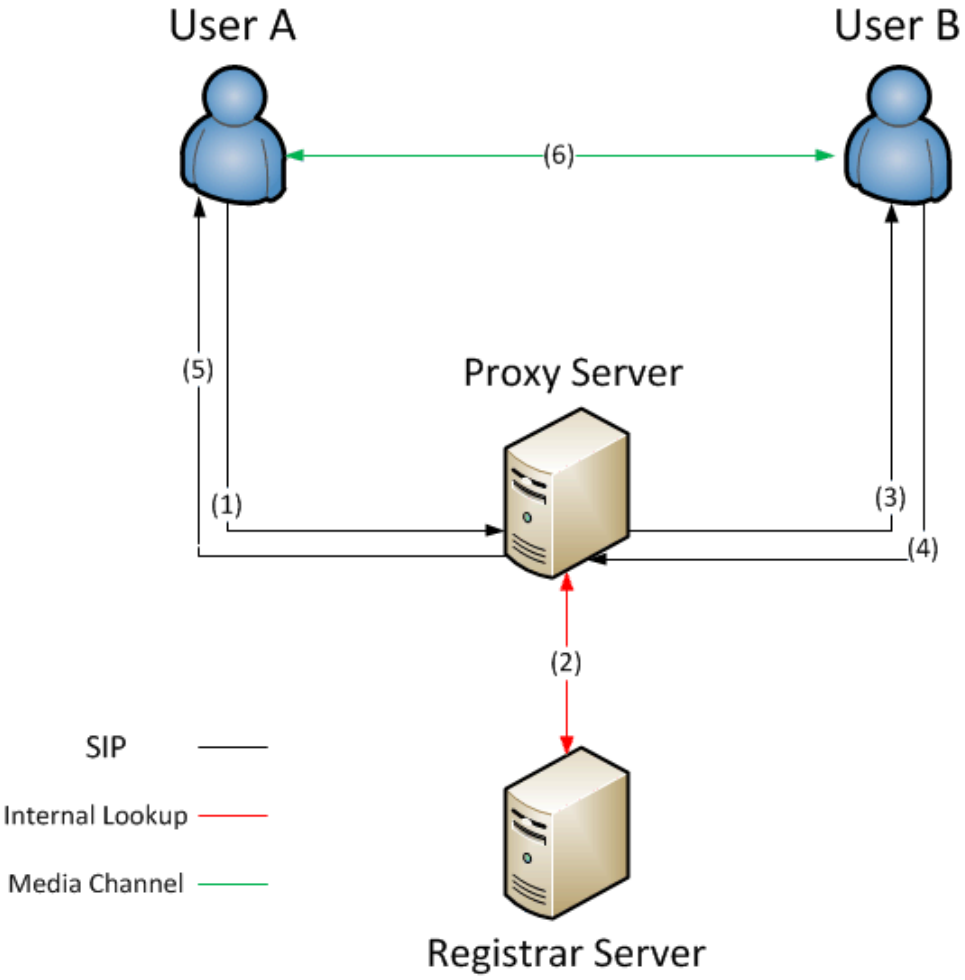


Figura 2.7: Esempio di sessione SIP

Tipi di messaggi SIP

Gli utenti SIP sono risorse identificabili mediante URL o URI che contengono informazioni sul dominio, sul nome utente, sull'host o il numero col quale il particolare utente partecipa alla sessione; la forma è del tipo user@host dove user corrisponde al nome utente o numero di telefono, host al nome del dominio o indirizzo di rete. Possibili esempi sono:

- `sip:federico@nomeDominio.it`
- `sip:federico@194.151.22.30:5068`
- `sip:+39-123-456@mioGateway.com`

Un messaggio SIP è costituito dalla chiamata di un metodo (SIP Request Type), seguito da una serie di campi detti headers, simili a quelli presenti nelle email. Inoltre, separati da una riga vuota, vi sono gli header del protocollo SDP (Session Description Protocol [24]) che segnalano all'host contattato che tipo di sessione multimediale verrà instaurata per la comunicazione. Alcuni messaggi utilizzati frequentemente sono:

- `REGISTER`: inviato da uno User Agent quando vuole registrare presso un Registrar Server il proprio punto di ancoraggio alla rete
- `BYE`: utilizzato per chiudere il dialogo SIP
- `CANCEL`: per terminare un dialogo se la sessione non ha ancora avuto inizio
- `INVITE`: per invitare un utente a partecipare ad una sessione
- `ACK`: messaggio di riscontro
- `OPTIONS`: richiesta delle funzionalità del server o di altri dispositivi.

Possibili messaggi di risposta sono:

- `(1XX)` Informational: il server sta contattando l'utente chiamato (non è disponibile una risposta definitiva, il client viene lasciato in attesa)
- `(2XX)` Success: la richiesta ha avuto successo
- `(3XX)` Redirection: la richiesta deve essere inoltrata ad un altro indirizzo
- `(4XX)` Request Failure: la richiesta non è andata a buon fine
- `(5XX)` Server Failure: il server non riesce ad elaborare la richiesta
- `(6XX)` Global Failure: nessun server può elaborare la richiesta.

Un esempio di messaggio di richiesta SIP è il seguente:

```
INVITE sip:userA@domain.com SIP/2.0
Via: SIP/2.0/UDP pc1.domain.com
Max-Forward: 70
To: "userB" <sip:userB@domain.com >
From: "userA" <sip:userA@domain.com >;tag=123
Call-ID: 1234567890@171.1.1.1
CSeq: 1 INVITE
Contact: <sip:userB@ pc1 domain.com >
Content-Type: application/sdp
Content-Length: 100

------Segue la sezione SDP------
```

E un ipotetico messaggio di risposta:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP pc1.domain.com
To: "userB" sip:userB@domain.com ; tag=999
From: "userA" <sip:userA@domain.com >;tag=123
Call-ID: 1234567890@171.1.1.1
CSeq: 1 INVITE
Contact: <sip:userB@ pc1 domain.com >
Content-Type: application/sdp
Content-Length: 100

------Segue la sezione SDP------
```

2.4.2 Protocolli AAA

Il protocollo utilizzato in IMS per l'AAA (Authentication, Authorization and Accounting) è **DIAMETER**, successore del protocollo RADIUS. DIAMETER consiste di un protocollo di base (le cui specifiche si trovano in [25]) e di un set di estensioni chiamate *applicazioni Diameter*, che permettono al protocollo di adattarsi ad una particolare applicazione in un dato ambiente. DIAMETER opera su layer di trasporto affidabile (a differenza di RADIUS), quindi TCP o SCTP, e i dati sono trasportati sotto forma di header seguito da *AVP (Attribute Value Pair)*, ovvero una serie di attributi. Le entità definite dal protocollo sono:

- **Client Diameter**, dispositivo che accede alla rete in maniera controllata;
- **Server Diameter**, l'entità che gestisce le richieste di autenticazione, autorizzazione e accounting per un dominio;
- **Relay Agent**, si occupa del routing dei messaggi DIAMETER basandosi sulle informazioni contenute in essi;
- **Proxy Agent**, anch'esso parte del meccanismo di routing, può implementare decisioni di policy, come ad esempio il controllo di utilizzo della sessione;
- **Redirect Agent**, restituisce al peer che ha effettuato la richiesta le informazioni necessarie ad inviare la successiva richiesta alla destinazione corretta;
- **Translation Agent**, effettua la conversione tra protocolli diversi, come ad esempio da RADIUS a DIAMETER.

Nella Figura 2.6 (più sopra) le linee tratteggiate corrispondono a flussi di messaggi DIAMETER. In IMS il protocollo DIAMETER entra in gioco per la prima volta durante la fase di autenticazione dell'utente: poiché le informazioni relative agli utenti sono memorizzate all'interno dell'HSS, sarà il S-CSCF a dialogare con questa componente mediante il protocollo DIAMETER. Anche l'autorizzazione, che si basa sulle informazioni presenti nel profilo dell'utente, vede instaurarsi la medesima interazione. Infine, per quanto riguarda l'accounting e la gestione della tariffazione, IMS può svolgere tali operazioni secondo una logica offline (nel caso di contratto ad addebito) od online (ad esempio per clienti con tessere prepagate). Nel primo caso è richiesta l'interazione con il modulo detto **Charging Collection Function (CCF)** (che contiene le informazioni relative alle chiamate effettuate); nel secondo caso, più complesso, si utilizza una applicazione Diameter detta Diameter Credit-Control Application [26], sviluppata appositamente per poter effettuare la tariffazione online. La Figura 2.8 rappresenta questo possibile scenario di utilizzo.

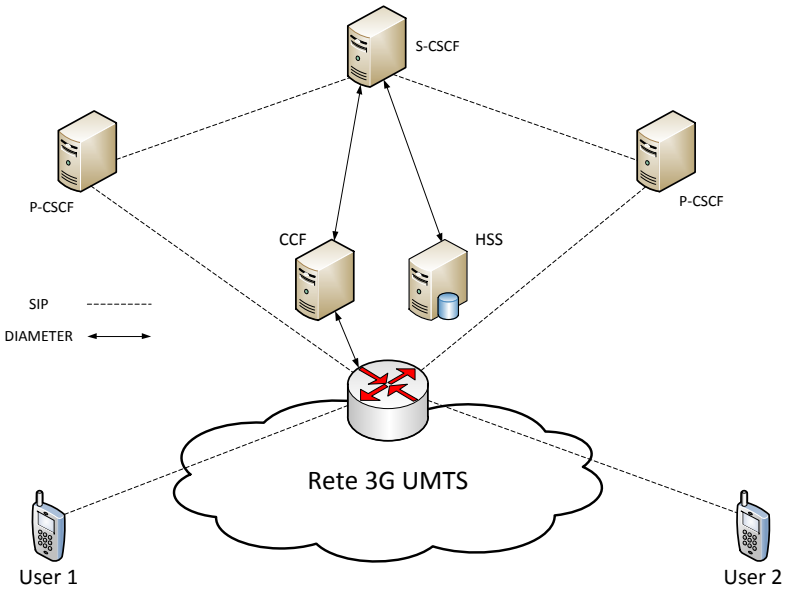


Figura 2.8: Uso del protocollo DIAMETER per le operazioni di AAA

2.4.3 Altri protocolli

Oltre ai già citati protocolli di controllo per la sessione e di AAA, all'interno di IMS ne trovano spazio anche altri, volti alla gestione di funzionalità

opzionali o dei flussi multimediali. Di seguito vi è una panoramica dei più importanti protocolli di questo tipo usati in IMS:

- **H.248**: sviluppato da ITU-T ed IETF, questo protocollo è utilizzato in IMS per gestire la segnalazione a livello di trasferimento dei flussi multimediali (si veda Figura 2.4) [27];
- **Real-Time Procotol (RTP)** e **Real-Time Control Protocol (RTCP)**: utilizzati per il trasporto di flussi dati in tempo reale, come ad esempio i flussi audio e video;
- **HyperText Transfer Protocol (HTTP)**: il principale protocollo di trasferimento dati sul web, in IMS viene adoperato per la gestione dei servizi presenti sull'Application Server da parte dell'utente (a mezzo dell'interfaccia Ut);
- **Session Description Protocol (SDP)**: utilizzato per la negoziazione delle funzionalità di una sessione multimediale attraverso lo scambio di messaggi *offer/answer*. Consente di descrivere le sessioni multimediali in formato testuale.

2.5 QoS Policy

In uno scenario reale, servizi differenti possono avere differenti caratteristiche, a seconda del profilo dell'utente, della posizione o del tipo di accesso alla rete. In particolare, vi sono due limitazioni alla tipologia di sessioni che un terminale può stabilire in una rete IMS:

- Limitazioni a specifici utenti;
- Policy generali della rete.

La piattaforma IMS è responsabile del controllo della sessione, ma i flussi di dati delle applicazioni sono al di fuori della sua competenza, dal momento che potrebbero non attraversare nemmeno i nodi del dominio IMS. Per poter instaurare i meccanismi di QoS nei flussi delle applicazioni controllate da IMS, è quindi necessario che quest'ultimo interagisca con gli elementi della rete che trasportano i dati; l'entità IMS adibita a questo scopo è la cosiddetta **Policy Decision Function (PDF)**. Essa è a conoscenza dei dettagli riguardanti il livello applicativo della sessione in corso di elaborazione, come ad esempio i codec utilizzati e la quantità di banda richiesta; il P-CSCF si occupa di fornirgli questi dettagli, che il PDF sfrutta poi per informare i nodi della rete dedicati al trasporto. In sostanza, il PDF è, per la QoS, come un intermediario tra il livello di QoS definito al livello applicativo e la sua realizzazione al livello di rete.

2.5.1 Scambio di informazioni di QoS Policy

Le informazioni di QoS raccolte dalla Policy Decision Function (PDF), devono essere trasmesse ai relativi nodi CSCF affinché la sessione possa rispettare le particolari policy adottate. Tuttavia, per ottenere tali informazioni, la PDF deve interagire con un altro modulo, detto **Policy Enforcement Point (PEP)**, che gestisce le risorse in termini di banda disponibile, dimensione del buffer dei router interni, ecc. Lo scambio di informazioni tra PDF e PEP avviene tramite messaggi di *Policy Request* e *Policy Decision*, formattati secondo le specifiche del protocollo DIAMETER. Nel caso

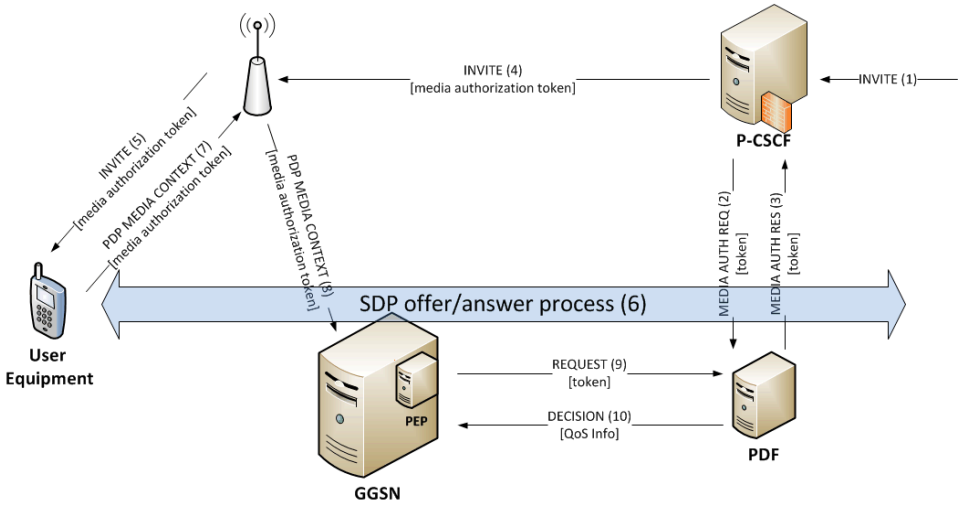


Figura 2.9: Scambio di messaggi QoS nel caso in cui il destinatario appartenga alla rete GPRS

di interazione con una rete GPRS, il nodo PEP è generalmente integrato all'interno del **Gateway GPRS Support Node (GGSN)**, cioè quella componente che funge da collegamento tra la rete GPRS e la rete IMS. In questo contesto, se l'utente destinatario appartiene alla rete GPRS, il compito del PDF è quello di generare un token che consente all'utente a ricevere un messaggio di `INVITE` e autorizzare l'allocazione di risorse presso il GGSN. Difatti, quando il P-CSCF riceve l'`INVITE` (1) (Figura 2.9), esegue una `MEDIA AUTH REQ` (2) al PDF e da quest'ultimo riceve il token all'interno del messaggio di risposta `MEDIA AUTH RES` (3). In questo modo il PDF registra l'ID della Sessione (utilizzato per l'allocazione delle risorse) e successivamente inoltra l'`INVITE` (4) contenente il token di media authorization verso l'utente. A questo punto, tramite il protocollo di offer/answer SDP (6), i due utenti negoziano le risorse e i codec che possono supportare: i P-CSCF e S-CSCF possono infatti decidere di bloccare determinate tipologie di flussi

multimediali andando a leggere i campi SDP del messaggio di `INVITE` ed, eventualmente, inviando il rifiuto al mittente.

Una volta portato a termine il processo di offer/answer, l'utente può richiedere di allocare le risorse concordate: nel caso di GPRS, l'utente genera un messaggio `PDP MEDIA CONTEXT` (7) in cui è aggiunto il token di autorizzazione ricevuto dal P-CSCF. Questo messaggio è inoltrato al GGSN (8): infine il modulo PEP interroga il PDF attraverso un `REQUEST` (9) contenente il token, e quest'ultimo elabora la richiesta confrontando l'ID della Sessione contenuto nel token inviando poi il messaggio `DECISION` (10) al GGSN, che allocherà le risorse richieste in caso di decisione favorevole.

2.6 Identificazione

Una delle più importanti funzionalità che la rete deve garantire è l'identificazione degli utenti. Questi devono essere riconosciuti in maniera tale che le chiamate vengano inoltrate alla persona corretta e i profili siano aggiornati secondo i servizi sottoscritti e le chiamate effettuate. IMS definisce un metodo standardizzato per identificare gli utenti, i servizi ed i nodi della rete.

2.6.1 Private e Public User Identity

In IMS ad un utente possono essere associati i seguenti identificativi:

- *IP Multimedia Private Identity (IMPI)* o più semplicemente **Private Identity**;
- *IP Multimedia Public Identity (IMPU)* o **Public Identity**.

Ogni abbonato dispone di una Private Identity, che viene utilizzata per le procedure di registrazione, autenticazione, autorizzazione, accounting e amministrazione. Tale identità è presente nell'HSS ed è scritta nel formato Network Access Identify (NAI), come definito nella RFC 2486, ovvero:

`nomeutente@dominio`

Caratteristiche principali della Private Identity sono:

- è autenticata solo nella fase di registrazione dell'utente;
- è un identificativo globale usato dall'operatore IMS;
- deve essere assegnato in maniera permanente ad una registrazione dell'utente ed è valido per la sola durata della registrazione;
- il S-CSCF deve averla a disposizione per poter effettuare le procedure di registrazione e de-registrazione.

Un utente può invece possedere più di una Public User Identity: tale identificativo è infatti utilizzato nelle comunicazioni tra gli utenti (per il routing dei messaggi), e permette inoltre ad una persona di essere riconosciuta con nomi diversi da gruppi diversi (ad esempio amici vs colleghi); può presentarsi sotto forma di SIP URI o nel formato TEL URI, quindi rispettivamente:

```
sip:nomeutente@dominio
```

oppure:

```
sip:+39-123-456-7890@dominio; user=phone
```

Caratteristiche della Public User Identity sono:

- se multipla può fungere da alias;
- non è autenticato durante la fase di registrazione;
- può essere usato per identificare le informazioni dell'utente all'interno dell'HSS.

Sia le Public User Identity che le Private User Identity sono salvate all'interno dell'HSS; inoltre un utente può disporre di più Private Identity, nel caso in cui voglia utilizzare contemporaneamente le funzionalità della rete da più punti d'accesso (ad esempio, per un utente GSM, corrisponde a possedere più di una SIM card). [28]

2.6.2 Relazioni tra Private e Public User Identity

Gli operatori assegnano una o più Public User Identity ed una Private User Identity ad ogni utente: nel caso dei GSM/UMTS la smart card contiene la Private User Identity ed almeno una Public User Identity. L'HSS, in quanto database contenente tutte le informazioni degli utenti, contiene sia la Private User Identity che la lista di tutte le Public User Identity collegate ad essa. Inoltre, sia l'HSS che il S-CSCF sono in grado di mettere in relazione i differenti tipi di identità. La Figura 2.10 illustra le relazioni che intercorrono tra le identità all'interno di una rete IMS: un utente è assegnato ad una particolare Private User Identity e ad un certo numero di Public User Identity; questo è quanto avveniva fino alle specifiche 3GPP di IMS Release 5.

Con la Release 6, 3GPP ha esteso queste relazioni: ad un utente della rete possono essere assegnate più Private User Identity; nel caso di UMTS, solo una di queste è salvata nella smart card, ma lo stesso cliente può avere più smart card da inserire in differenti terminali IMS. E' poi possibile che alcune Public User Identity siano utilizzate in combinazione con più Private User Identity;

la Figura 2.11 mostra come una stessa Public User Identity (la numero 2) possa essere utilizzata simultaneamente da due terminali IMS, ognuno assegnato ad una Private User Identity (cioè aventi smart card differenti).

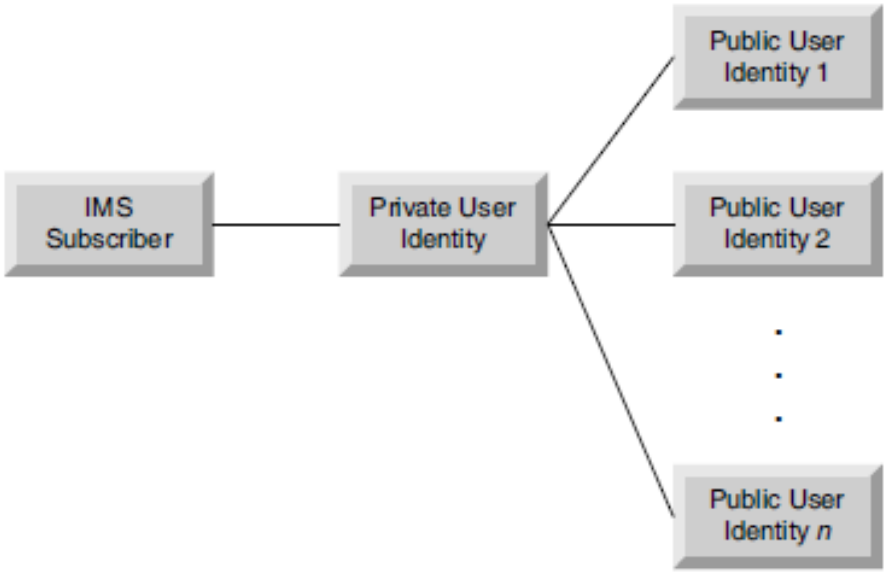


Figura 2.10: Relazioni tra utente, Private User Identity e Public User Identity (3GPP R5)

2.6.3 Service e Network Identity

Non solo gli utenti devono essere identificati all'interno della rete: anche tutti i servizi attivi e le altre entità presenti al suo interno devono poter essere tracciati; a questo scopo, IMS definisce i concetti di Public Service Identity (PSI) e Network Identity. Il primo fa riferimento all'identità posseduta dagli Application Server, ed ha la forma di un SIP URI: ad esempio, nei servizi di messaggistica, una tale identità serve per determinare il servizio di messaging list che riceve i messaggi dagli utenti e si occupa di smistarli ai destinatari corretti. Anche nel caso di conferenze audio/video si utilizza questo meccanismo: una specifica URI viene creata per la conferenza in corso. Per quanto riguarda la Network Identity, essa non è altro che l'identificativo di un nodo di rete che svolge una funzione di routing dei messaggi SIP, ed ha anch'essa la forma di una SIP URI. In genere tali URI vengono pubblicati e distribuiti da un server DNS. Un esempio può essere l'URI assegnato ad un particolare nodo S-CSCF:

```
sip:  region.scscf1@dominio
```

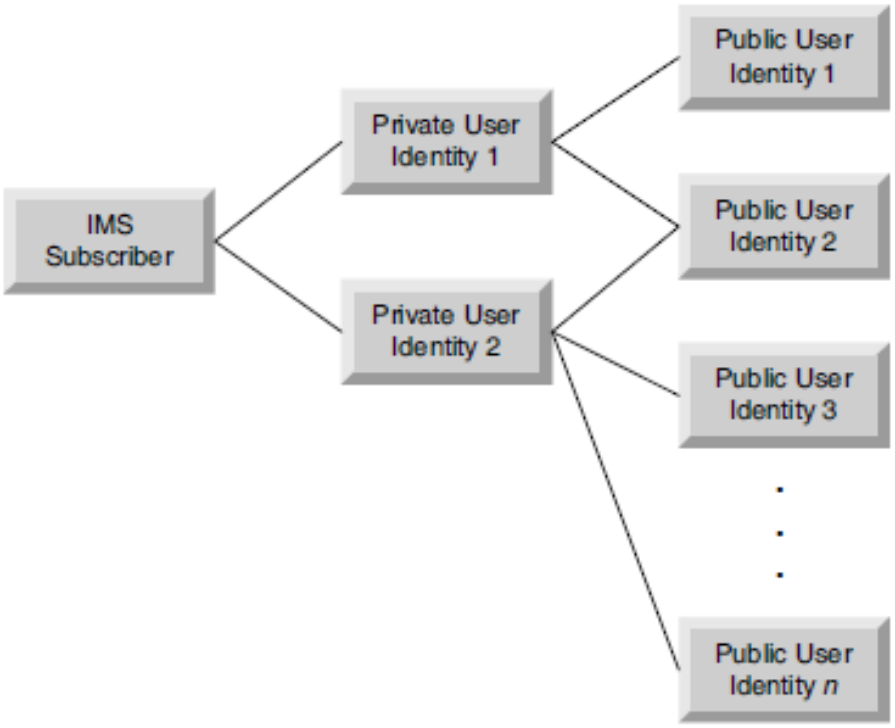


Figura 2.11: Relazioni tra utente, Private User Identity e Public User Identity (3GPP R6)

2.7 Esempi di segnalazione in IMS

Come già ampiamente discusso, la segnalazione all'interno dell'architettura IMS avviene principalmente mediante il protocollo SIP; in aggiunta alle funzionalità di base offerte da tale protocollo, in IMS è stato aggiunto il supporto al protocollo SDP, utile per la gestione dei messaggi offer/answer durante la fase di negoziazione delle capabilities tra terminali IMS. In particolare, le procedure fondamentali possono essere riassunte in:

- **Registrazione** di un utente: permette di identificare l'utente ed i servizi che esso può utilizzare nella rete;
- **Instaurazione** di una chiamata: allestimento di un canale di comunicazione tra due utenti (che possono essere o meno client IMS);
- **Chiusura** di una chiamata: chiusura del canale di comunicazione precedentemente instaurato.

E' bene specificare che le comunicazioni SIP tra due (o più) entità avvengono mediante scambi di messaggi che possono essere definiti in due modi:

- **Transazioni SIP**;
- **Dialoghi SIP**.

2.7.1 Transazioni SIP

Vi sono tre tipi di transazioni: le transazioni regolari, le transazioni INVITE-ACK, e le transazioni CANCEL; il tipo dipende dalla richiesta che la inizializza.

Le **transazioni regolari** sono inizializzate da qualsiasi richiesta ad esclusione di `INVITE`, `ACK` o `CANCEL`; un esempio è la transazione che avviene a seguito della richiesta di `BYE`, illustrata nella Figura 2.12. In una transazione di questo tipo l'User Agent Server (UAS) riceve una richiesta e genera una risposta che termina la transazione. Teoricamente, l'UAS potrebbe generare più risposte (dette *provisional response*) prima di quella finale, ma si tratta di una eventualità piuttosto rara nell'ambito delle transazioni regolari. Una **transazione INVITE-ACK** interessa due differenti

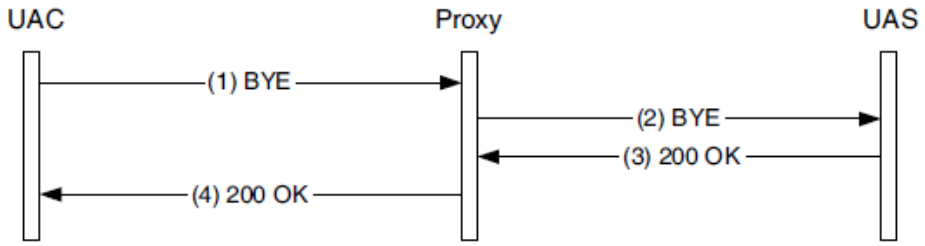


Figura 2.12: Una transazione regolare (`BYE`)

transazioni: una `INVITE` ed una `ACK`. L'UAS riceve la richiesta `INVITE`, dopodiché genera zero o più risposte intermedie e la risposta finale; una volta che l'UAC riceve quest'ultima, genera una richiesta `ACK`, alla quale non fa seguito alcuna risposta. Tale comportamento è illustrato nella Figura 2.13 Infine, le **transazioni CANCEL** sono inizializzate da una richiesta di `CANCEL`, e sono sempre collegate ad una precedente transazione (cioè quella che deve essere annullata). Queste transazioni sono simili a quelle regolari, con la differenza che la risposta finale è generata dal prossimo SIP hop (generalmente un proxy) e non dall'UAS. La Figura 2.14 mostra una richiesta di `CANCEL` volta ad annullare la precedente transazione `INVITE`: si noti che la transazione `INVITE`, una volta cancellata, termina come di consueto (quindi con una risposta finale più il relativo `ACK`).

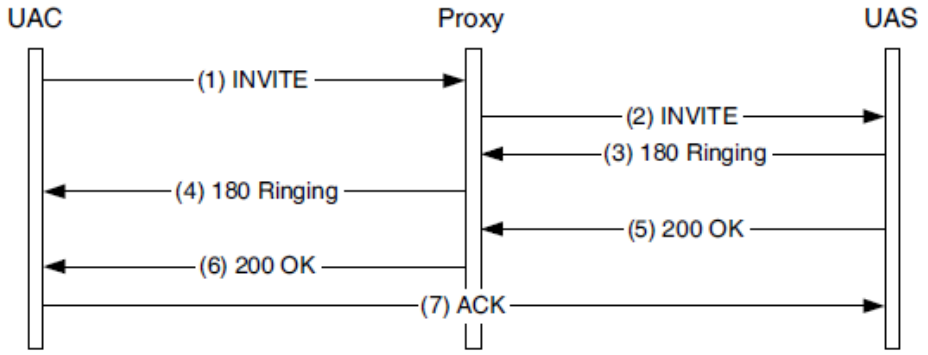


Figura 2.13: Una transazione `INVITE-ACK`

2.7.2 Dialoghi SIP

Per **dialogo SIP** si intende lo scambio di messaggi SIP tra due User Agent. Un esempio di dialogo è lo scambio di messaggi che avviene a partire dalla transazione di `INVITE` proveniente da un User Agent fino alla transazione di `BYE` che pone fine alla comunicazione. Oltre ai messaggi `SIP INVITE` ne esistono anche altri in grado di inizializzare un dialogo (ad esempio `SUBSCRIBE`).

Una volta stabilito un dialogo, tutte le successive richieste all'interno di esso seguiranno uno stesso percorso: ad esempio, i due User Agent potrebbero comunicare end-to-end durante tutto il dialogo. Tuttavia, alcuni proxy potrebbero decidere di rimanere all'interno della segnalazione SIP, mediante l'utilizzo dei campi dell'header `Record-Route`, `Route` e `Contact`.

2.7.3 I campi `Record Route`, `Route` e `Contact`

Un proxy può richiedere di restare all'interno del percorso relativo alle richieste successive di un dialogo aggiungendo il campo `Record-Route` nell'header del messaggio di `SIP INVITE`: in Figura 2.15 un proxy effettua questo tipo di richiesta inserendo nel messaggio (2) la sua URI nel campo `Record-Route` (il parametro 'lr' indica che si utilizza il meccanismo di *loose routing*, come specificato nella RFC 3261) destinato ad Alice; Bob, invece, riceve questa informazione all'interno del messaggio `200 OK` (4). Da questo momento in poi, Bob ed Alice inseriranno il campo `Route` nell'header delle richieste, indicando che il proxy presente all'indirizzo `sip:p1.domain.com` deve essere visitato. Il messaggio di `ACK` inviato da Bob ad Alice (5-6) è un esempio di richiesta che fa uso del campo `Route`; allo stesso modo il messaggio di `BYE` passerà attraverso il proxy (7).

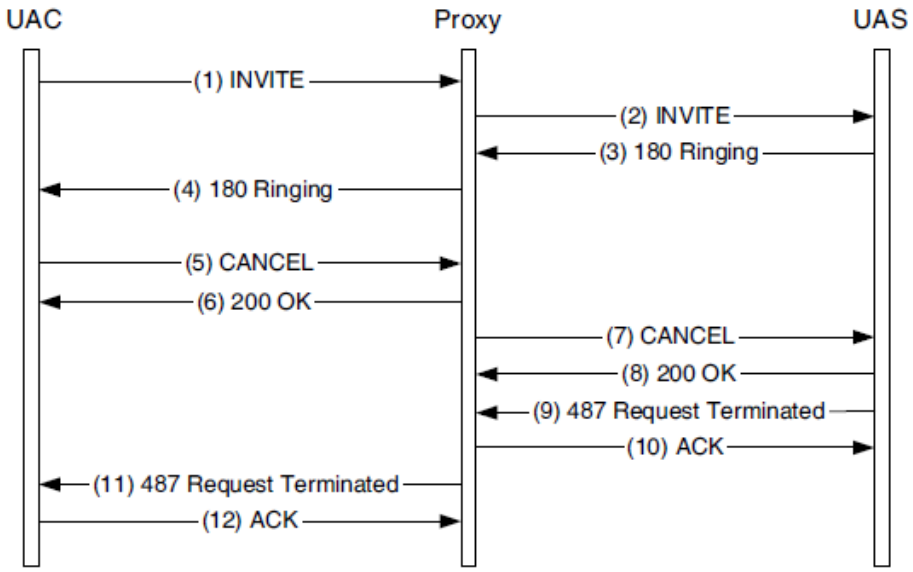


Figura 2.14: Una transazione `CANCEL`

2.8 Uso degli Initial Filter Criteria

Per poter offrire servizi agli utenti, IMS fa uso delle informazioni contenute nei profili salvati nell'HSS: questi vengono richiesti dal S-CSCF durante la fase di registrazione dell'utente, e sono conservati al suo interno. In accordo con le specifiche 3GPP TS 23.218 [29], un *Service Profile* è composto da una *Private User Identity* al quale è applicabile e uno o più servizi associati; per ogni Service Profile è poi definita una *Public User Identity* e zero o più **initial Filter Criteria (iFC)**. Gli iFC contengono tutte le informazioni necessarie ai CSCF per decidere se uno specifico Application Server deve essere o meno interessato nell'erogazione di un servizio per l'utente in questione. Gli iFC vengono valutati dal S-CSCF solo per le richieste SIP che inizializzano una sessione di dialogo (dette *initial request* - quindi ad esempio `INVITE`, `SUBSCRIBE`, `REGISTER`), secondo il seguente procedimento:

1. Creare una lista degli *initial Filter Criteria* per la richiesta in questione, ordinati secondo la loro priorità
2. Esaminare la richiesta ricevuta per determinare il Trigger Point
3. Controllare se vi è una corrispondenza con uno dei Trigger Point presenti nell'iFC di priorità massima e:
   (a) in assenza di match il S-CSCF procede con il passo 4
   (b) se vi è un match, il S-CSCF:

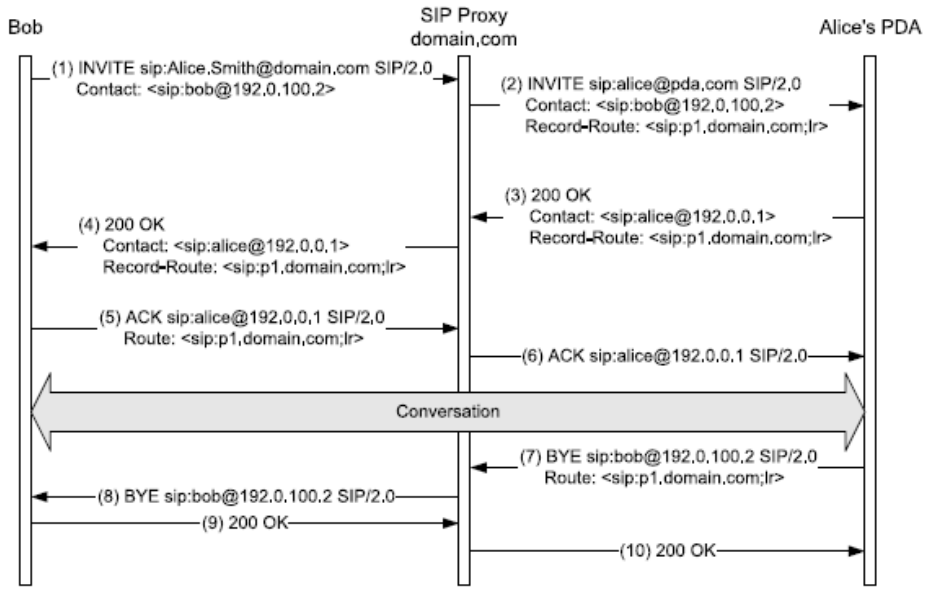


Figura 2.15: Funzionamento dei campi `Record-Route`, `Route` e `Contact`

   i. aggiunge un *Original Dialog Identifier (ODI)* alla richiesta, per consentire al S-CSCF di identificare la provenienza del messaggio anche qualora l'Application Server effettui delle modifiche all'ID del dialogo
   ii. invia la richiesta all'AS specificato nell'iFC corrente. L'AS effettua le operazioni richieste: può modificare la richiesta e inviarla nuovamente al S-CSCF lungo l'interfaccia ISC
   iii. procede con il passo 4 se viene ricevuta una richiesta con lo stesso ODI proveniente dall'interfaccia ISC (quindi da un AS)

4. Ripetere i passi 2. e 3. per ognuno degli iFC ordinati al punto 1, finché anche l'ultimo non viene esaminato

5. Instradare la richiesta secondo quanto prescritto normalmente dal protocollo SIP

Nota: qualora un AS decida di terminare l'elaborazione di una richiesta ed invii tale segnalazione lungo l'interfaccia ISC, il S-CSCF dovrà abbandonare il controllo degli iFC di priorità inferiore (se presenti). Inoltre, il S-CSCF utilizza il meccanismo di instradamento detto di loose routing (come definito nella RFC 3261) per assicurare che, dopo le elaborazioni effettuate dall'AS, il messaggio torni ad esso per le altre operazioni; pertanto, il S-CSCF inserisce nella prima posizione del campo Route del messaggio SIP l'URI dell'AS,

seguita dalla sua stessa URI, che include l'*Original Dialog Identifier (ODI)*. La Figura 2.16 illustra il procedimento appena descritto.

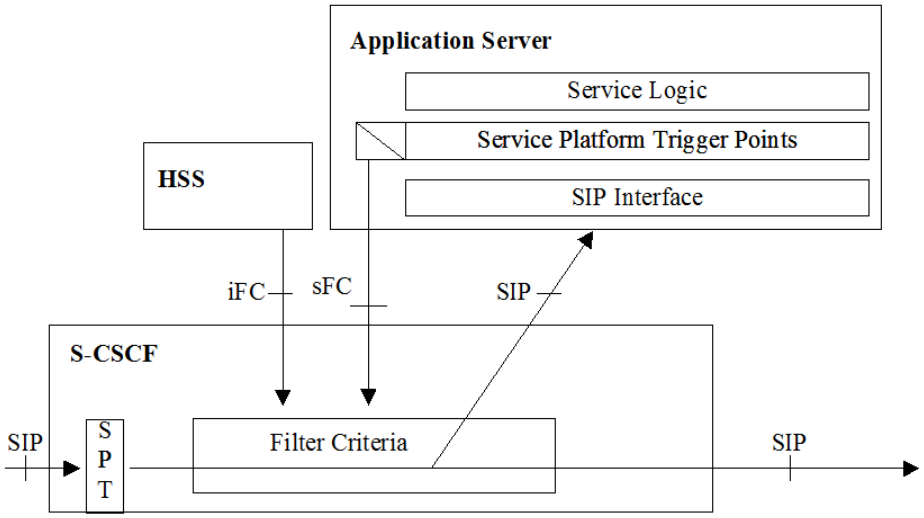


Figura 2.16: Funzionamento degli *initial Filter Criteria*

Per quanto riguarda la struttura interna di un iFC, vengono definiti i seguenti campi:

- **Priority**: determina l'ordine nel quale questi Filter Criteria devono essere valutati, in relazione ad eventuali altri criteri associati allo stesso Service Profile.
- **Trigger Points**: Contiene l'espressione che è valutata per determinare se una specifica richiesta SIP debba essere o meno inoltrata ad un AS. Gli elementi che possono essere presi in considerazione sono l'URI della richiesta, il metodo SIP utilizzato, la presenza/assenza di un particolare header SIP o la parziale/completa corrispondenza di un pattern al suo interno, il contenuto della *Session Description*. Nota: nel caso non vengano definiti dei Trigger Points all'interno dell'iFC, le richieste verranno inoltrate incondizionatamente verso l'AS.
- **Application Server**: Contiene la SIP URI dell'AS cui deve essere inoltrata la richiesta nel caso venga attivato un Trigger Point. Nell'eventualità che non si possa contattare l'AS, la richiesta verrà trattata in accordo con quanto definito nel campo *Default Handling*. Infine il campo Service Information contiene alcuni dati che l'AS potrebbe utilizzare per elaborare la richiesta.

Nel caso in cui vi sia più di un AS interessato (cioè più di un servizio deve essere erogato all'utente), il S-CSCF inoltrerà la richiesta ad ognuno di essi,

secondo la priorità indicata nell'opportuno campo degli iFC. Un esempio di

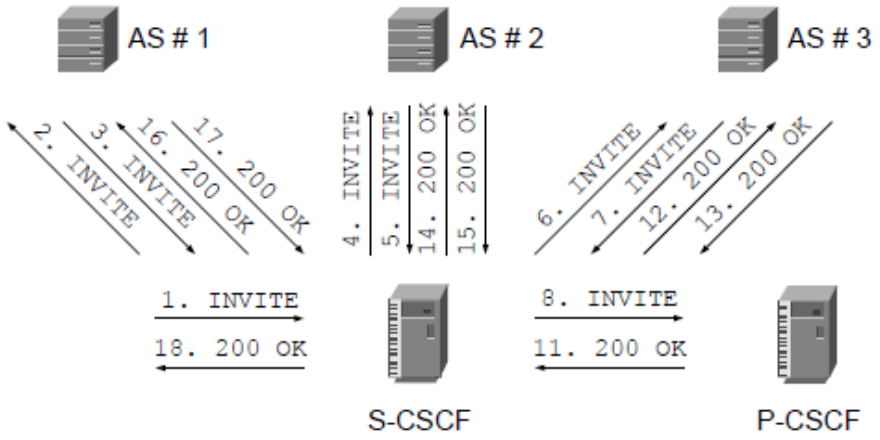


Figura 2.17: Service Chaining in IMS

tale comportamento è rappresentato nella Figura 2.17: dopo aver ricevuto la richiesta di SIP INVITE (1), il S-CSCF valuta l'iFC con la priorità più alta e inoltra la richiesta all'AS corrispondente, in questo caso AS#1. Quando il S-CSCF riceve indietro il `SIP INVITE` (3) valuta l'iFC (tra i restanti) che ha priorità massima e inoltra ancora una volta la richiesta all'AS definito in esso (4), cioè AS#2 . Le stesse operazioni (5-6) avvengono per un terzo AS, chiamato AS#3, dopodiché, ricevendo da quest'ultimo la richiesta SIP (7), il S-CSCF la inoltra infine al P-CSCF (8) che si occuperà di effettuarne la consegna al dispositivo IMS dell'utente.

Questo tipo di comportamento è detto ***service chaining***, e costituisce un meccanismo di base per la fornitura in sequenza di servizi: interazioni più complesse sono invece necessarie quando si è interessati alla composizione vera e propria di servizi, aspetto che verrà discusso nei prossimi capitoli.

Capitolo 3

Open IMS Core

Indice

L'utilizzo di *testbed* è un'operazione fondamentale nel contesto della diffusione e sviluppo delle reti IMS: la tipica architettura modulare ed espandibile implica infatti un aumento nel numero dei nodi e delle interfacce differenti chiamate ad operare insieme, con il risultato di incrementare la complessità delle operazioni. Per mezzo di un *testbed* gli operatori possono implementare e testare nuovi servizi, variando i parametri relativi al carico, robustezza e facilità d'utilizzo in un contesto molto simile a quello di reale applicazione.

A questo proposito sono state sviluppate varie piattaforme per il testing, anche da parte delle stesse aziende operanti nel campo delle telecomunicazioni: un esempio è costituito dall' *IMS Common System*, sviluppato da Ericsson, che offre un'architettura orizzontale e modulare composta dalle varie componenti dello standard IMS. [30]

Tuttavia, un ruolo importante è svolto dalle piattaforme open source: grazie ad esse, le università e le aziende possono collaborare allo sviluppo delle funzionalità offerte da IMS, facendo riferimento ad una base comune

liberamente espandibile. Inoltre, l'accesso libero a questo tipo di piattaforme consente l'interazione di figure differenti della *value chain* che possono contribuire allo sviluppo di nuove tecnologie e servizi potenzialmente proficui anche in una logica di mercato. Questo stimola la collaborazione tra le università ed enti di ricerca e le industrie, promuovendo uno scambio di *skill* certamente positivo.

E' comunque chiaro che piattaforme di questo tipo servono a fornire una base tecnologica per il testing iniziale delle componenti e delle applicazioni operanti nella rete, come supporto alla successiva creazione di soluzioni indipendenti da parte dei singoli operatori.

3.1 FOKUS IMS Playground

Il *Fraunhofer Institute FOKUS*, operante nell'ambito della ricerca sui sistemi di comunicazione, ha lanciato nel Luglio 2004 **Open IMS Playground**, un *testbed* che integra tutte le componenti fondamentali di una rete IMS e sviluppato in collaborazione con le maggiori industrie operanti nel settore. Si tratta di un ambiente di sviluppo, aperto e indipendente dal particolare operatore, che permette, tanto alle università quanto ad aziende, di testare le componenti, i protocolli e le applicazioni di una rete NGN/IMS. Ciò che si vuole incentivare è proprio questo tipo di collaborazione tra enti diversi, al fine di sviluppare componenti e applicazioni IMS che possano godere il più possibile di caratteristiche di interoperabilità (vedi Figura 3.1).

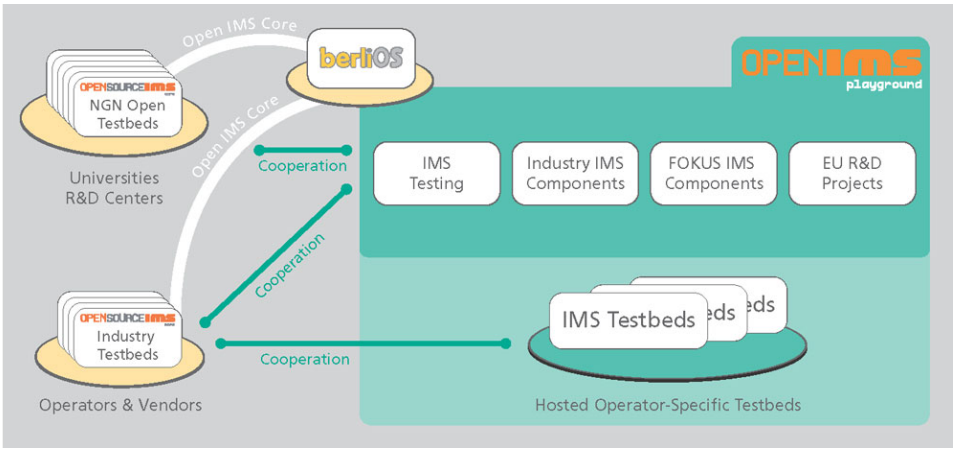


Figura 3.1: Il concept di OpenIMS Playground

Open IMS Playground è dunque utilizzato come tecnologia base da una parte per progetti industriali dell'Istituto FOKUS, dall'altra per progetti accademici o di R&D a medio termine in contesto europeo. Testimonianze

di un proficuo utilizzo del testbed di FOKUS giungono difatti dai laboratori di Ricerca e Sviluppo di molti operatori telefonici (ad esempio Portugal Telecom Inovacao, Orange Labs, Slovak Telekom) come pure da joint venture in ambito industriale per il testing di IMS [31]. Inoltre, OpenIMS Core ha spinto lo sviluppo di progetti open source nell'ambito dei client IMS, come testimoniato da [32] e [33].

Gli sviluppatori che creano nuove applicazioni IMS basate su differenti piattaforme, come IN/CAMEL, OSA/Parlay, JAIN, SIP Servlet, ecc., possono quindi testare la loro implementazione all'interno di questo ambiente. L'utilizzo di OpenIMS Playground sta di fatto indicando nuove direzioni per la ricerca e lo sviluppo, come dimostrato ad esempio da [34], [**?**].

Open IMS Playground include le seguenti componenti:

- **Open Source IMS Core (OSIMS)**: è il nucleo di Open IMS Playground, è open source e si compone di:
  - Home Subscriver Server (HSS): FHoSS;
  - Call Session Control Functions: CSCFs;
- **Open Service Enabler (OpenSE)**: implementa Parlay X Web Services, fornendo una modalità standard di accedere alle funzionalità della NGN mediante Web Service;
- **OSIMS Management Console (OMACO)**: permette di monitorare le componenti dell'IMS Core;
- **XML Document Management Server (XDMS)**: consente l'accesso alle informazioni sugli utenti e servizi da parte delle componenti che li erogano;
- **Presence Server**: fornisce il supporto presence per le applicazioni che ne richiedono l'uso, come Instant Messaging, Push To Talk over Cellular, Videochiamate/Videoconferenze, ecc.
- **Policy and Charging Control Architecture (PoCCA)**: costituisce il collegamento tra la rete d'accesso e l'IMS Core per fornire controllo dell'accesso, delle risorse e QoS;
- **Converged Open Messaging Server (COMS)**: implementa un service enabler OMA e offre funzionalità di Instant Messaging.

La Figura 3.2 seguente mostra l'architettura di Open IMS Playground.

3.2 Open Source IMS Core (OSIMS)

L'Open Source IMS Core (o OpenIMS Core, Figura 3.3) si compone dei Call Session Control Function (CSCF), elementi centrali della segnalazione

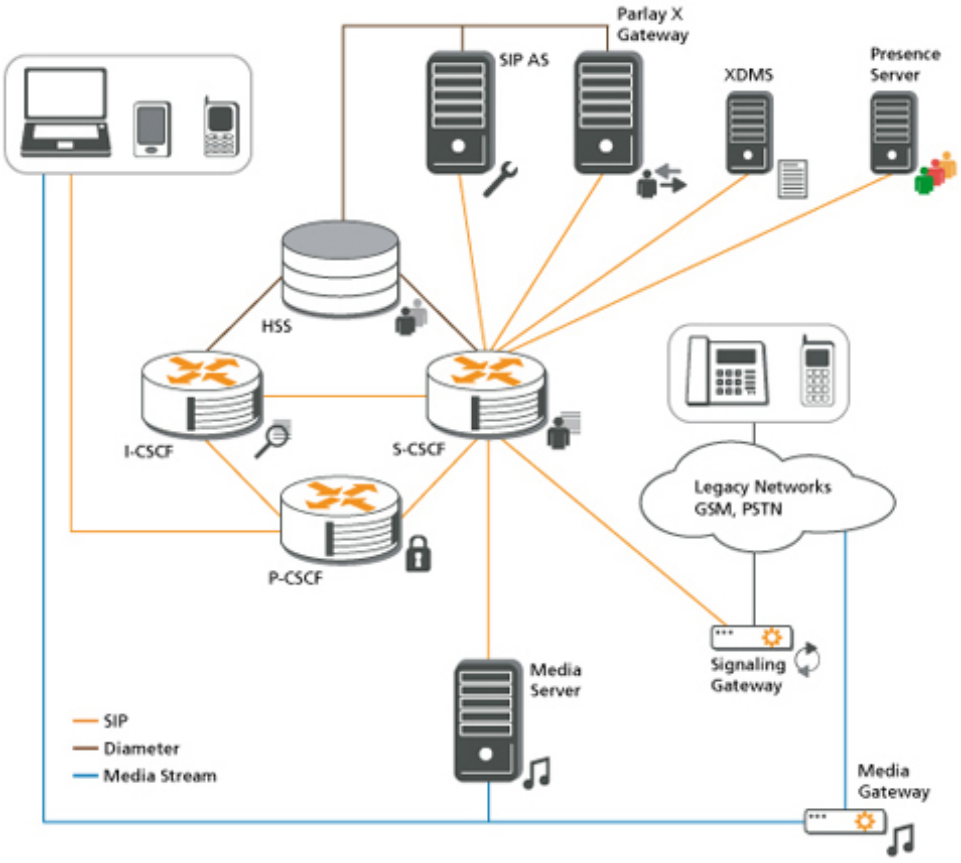


Figura 3.2: Architettura di Open IMS Playground

in IMS (vedi Par. 2.3.2.2), e di un Home Subscriber Server (HSS), che gestisce i profili degli utenti e le regole di routing associate (vedi Par. 2.3.2.1). Elementi centrali del progetto OpenIMS Core sono dunque i CSCF, sviluppati da FOKUS come estensioni del SIP Express router (SER) [36]; ma poiché anche le segnalazioni più banali di IMS richiedono un'interazione con l'HSS, è stato inserito in OpenIMS Core il FOKUS Home Subscriver Server (FHoSS).

3.2.1 Open IMS Call Session Control Function - CSCF

I moduli CSCF hanno come base il SIP Express Router (SER), che può operare come SIP registrar, proxy o redirect server e può gestire migliaia di chiamate al secondo; la sua struttura modulare ha permesso di effettuare facilmente aggiunte e modifiche alle specifiche di default, ottenendo così le funzionalità richieste dalla Release 6 di 3GPP IMS. Ogni CSCF è implementato come un modulo caricabile dinamicamente che aggiunge un determinato

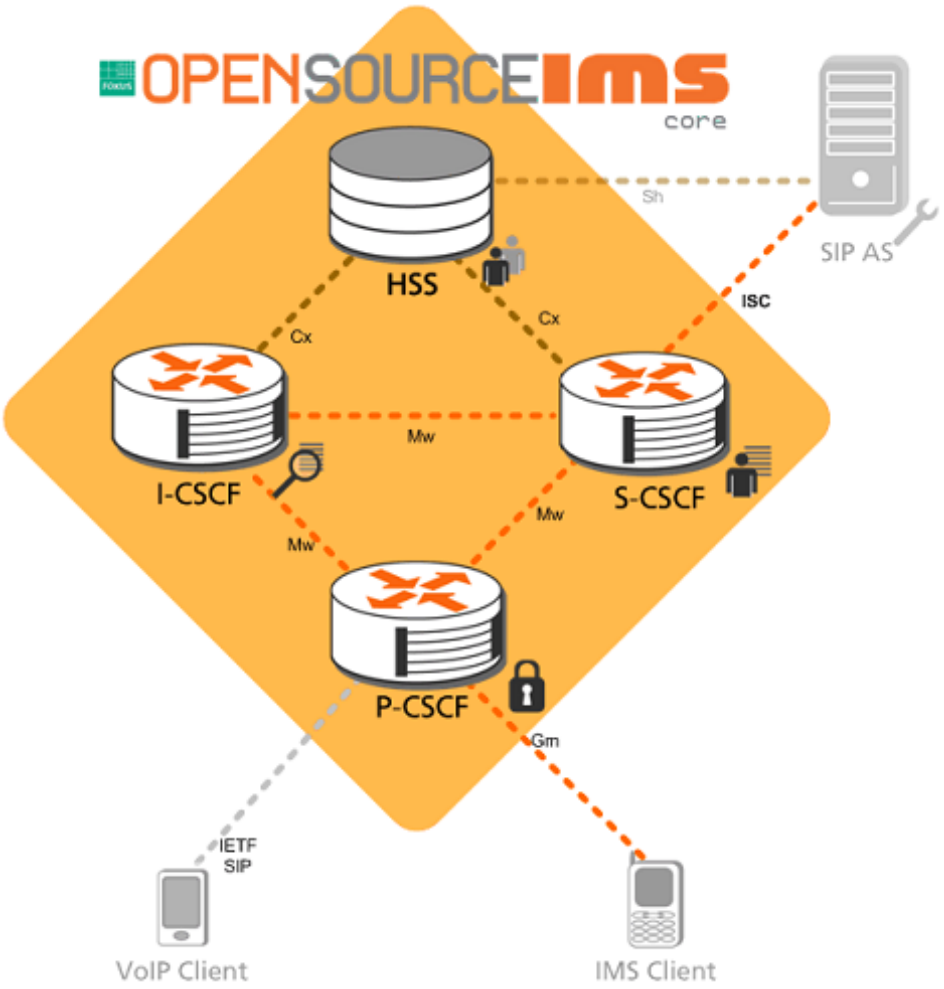


Figura 3.3: Struttura di Open IMS Core

insieme di caratteristiche al SER: inoltre, i moduli sono in grado di operare in maniera parallela e mantenere informazioni supplementari sul reciproco stato. Uno degli obiettivi primari dello sviluppo degli Open CSCF è stato mantenere il più possibile il livello di performance offerto da SER, dal momento che la vasta diffusione di quest'ultimo componente in ambito SIP ha di fatto definito il livello standard delle prestazioni.

Le modifiche principali alle quali SER è stato sottoposto per poter operare in ambito IMS riguardano soprattutto l'integrazione delle funzionalità DIAMETER: a questo proposito è stato aggiunta una componente, detta C DIAMTER Peer (CDP), che consente l'esecuzione parallela di processi riguardanti la segnalazione SIP e processi DIAMETER garantendo le performance tipiche di SER. In sostanza, dunque, i moduli CSCF di OpenIMS

costituiscono una versione potenziata di SER adeguata ad operare in una rete IMS.

Open Proxy CSCF

Nell'implementazione di OpenIMS Core, il P-CSCF svolge il ruolo di firewall a livello delle applicazioni tra la rete IMS e l'esterno: solo gli *endpoint* registrati possono inoltrare messaggi all'interno della rete e il P-CSCF assicura l'identità di tali utenti. Per questo, durante la registrazione il P-CSCF instaura un canale sicuro per ogni User Endpoint (UE); per tener traccia degli utenti registrati, dispone di un registrar interno che è aggiornato intercettando i messaggi di registrazione e, successivamente, le notifiche provenienti dal modulo S-CSCF. I dati sono mantenuti in una hash table, che permette un loro recupero veloce.

Il P-CSCF genera dei *charging vector* e vi inserisce gli identificatori della rete e dei percorsi necessari alla corretta elaborazione dei messaggi SIP; dopo una registrazione andata a buon fine, i successivi messaggi dell'utente verranno inoltrati, sulla base di informazioni del DNS Server, verso la corretta Home Network. Infine, il P-CSCF offre funzionalità di NAT per le segnalazioni SIP, qualora ve ne sia necessità.

Open Interrogating CSCF

L'I-CSCF svolge il ruolo di stateless proxy che, utilizzando le identità pubbliche del chiamante o del chiamato, interroga l'HSS e inoltra i messaggi al corretto S-CSCF; esso implementa l'interfaccia Cx, e pertanto supporta i comandi DIAMETER utili alla localizzazione dell'S-CSCF assegnato (o da assegnare).

Agisce poi da firewall per l'HSS, in quanto permette la ricezione dei soli messaggi di segnalazione provenienti da una rete trusted mediante Network Domain Security (NDS).

Open Serving CSCF

Il S-CSCF comunica anch'esso con l'HSS utilizzando il protocollo DIAMETER (sull'interfaccia Cx) per ottenere ed aggiornare le informazioni di autenticazione/registrazione degli utenti. Inoltre, può applicare a determinati profili gli *initial Filter Criteria (iFC)*, al fine di specificare le regole di segnalazione SIP nell'interazione con gli Application Server.

Questo modulo non genera i vettori di autenticazione, ma delega tale compito all'HSS: il S-CSCF si limita a confrontare questo valore con quello calcolato nell'User Equipment; sono supportati AKAv1-MD5, AKAv2-MD5 e MD5 come algoritmi di cifratura. Inoltre, il *registrar* presente all'interno del S-CSCF utilizza una complessa struttura di hash table per consentire tempi di risposta il più possibile ridotti: l'informazione necessaria

ad associare una *user identity* con un UE fisico è salvata qui ed utilizzata successivamente per l'instradamento delle chiamate.

FOKUS Home Subscriber Server (FHoSS)

Senza un Home Subscriber Server, OpenIMS Core sarebbe incompleto: FOKUS ha sviluppato una versione di HSS (appunto, FHoSS) interamente scritta in Java e basata su software open source. Scopo principale del modulo è permettere la configurazione del DBMS contenente di dati degli utenti (in questo caso MySQL) nonché l'utilizzo delle interfacce DIAMETER con i CSCF e il livello applicativo della rete IMS. FHoSS permette quindi la generazione di vettori di autenticazione e fornisce un'interfaccia HTTP-based per la gestione delle interfacce, dei profili utenti e degli iFC associati.

Di seguito è presente un'immagine dell'interfaccia utente di FHoSS, accessibile tramite il protocollo HTTP:

Le interazioni avvengono sia sul dominio della sicurezza, che su quello applicativo e infine su quello di controllo (con i differenti CSCF). Come mostrato in Figura 3.4, le comunicazioni DIAMETER per FHoSS si instaurano sulle tre interfacce Zh, Cx e Sh.

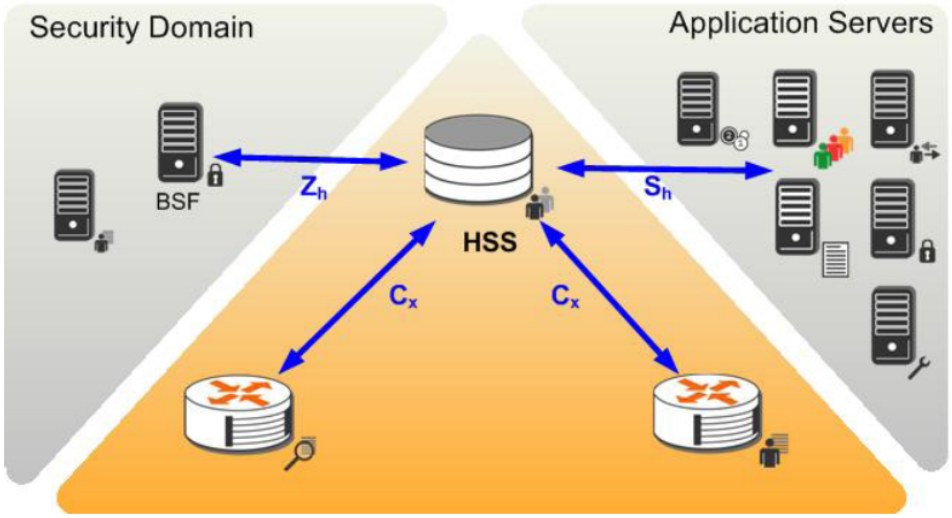


Figura 3.4: Interfacce DIAMETER per la comunicazione con l'HSS

3.3 Installazione e configurazione di Open IMS Core

OpenIMS Core fornisce le funzionalità di base necessarie all'instaurazione di una sessione di comunicazione tra due User Agent: a questo proposito, si fornisce di seguito una guida alla sua installazione e configurazione e, successivamente, verrà analizzata nel dettaglio la procedura di instaurazione, mantenimento e chiusura di una chiamata.

3.3.1 Prerequisiti

Le caratteristiche modulari di OpenIMS Core suggeriscono una sua configurazione su più macchine distinte interconnesse fra loro, in cui ognuna ricopre il ruolo di CSCF o HSS. Tuttavia, per un'analisi qualitativa delle funzionalità offerte dal sistema è sufficiente disporre di un'unica macchina Linux desktop. Per quanto riguarda invece i requisiti software, sono necessari:

- GCC3/4, `make`, JDK1.5, `ant`;
- MySQL (o altro DBMS) installato e attivo;
- `bison`, `flex`;
- `libxml2` (2.6 o superiore), `libmysql`, entrambe con relativi pacchetti dev;
- Linux kernel 2.6 e `ipsec-tools` per l'utilizzo delle funzionalità IPsec;
- Opzionale: `openssl` se si desidera testare il protocollo TLS;
- `bind` (o altro server DNS) installato e attivo;
- Browser Internet (per l'accesso all'interfaccia di FHoSS).

Una volta soddisfatti tali requisiti è possibile procedere all'installazione.

3.3.2 Installazione

Per prima cosa è necessario procurarsi i sorgenti, che sono preconfigurati per operare nel percorso `/opt/OpenIMSCore`. Al suo interno va creata la cartella `ser_ims` e utilizzare subversion per collocarvi i pacchetti relativi ai CSCF:

```
1 mkdir ser_ims
2 svn checkout http://svn.berlios.de/svnroot/repos/
    openimscore/ser_ims/trunk ser_ims
```

Allo stesso modo si crea la cartella FHoSS ed al suo interno si collocano i file relativi all'HSS:

```
mkdir FHoSS
svn checkout http://svn.berlios.de/svnroot/repos/
    openimscore/FHoSS/trunk FHoSS
```

A questo punto si procede alla compilazione dei sorgenti, dopo essersi assicurati che la versione del JDK sia la 1.5 o superiore:

```
java -version
```

Per prima cosa si compilano i file relativi ai CSCF:

```
cd ser_ims
make install-libs all
cd ..
```

quindi il modulo FHoSS in questo modo:

```
cd FHoSS
ant compile
ant deploy
cd ..
```

Se la compilazione va a buon fine è possibile procedere alla configurazione delle entità esterne ad OpenIMS Core, quali il DBMS (in questo caso MySQL) e il server DNS (bind9) e infine delle componenti interne.

3.3.3 Configurazione

Nota: alla fine della configurazione il sistema sarà predisposto per essere utilizzato in locale e con il dominio di default 'open-ims.test'. Bisogna tenere presente questo aspetto anche nella configurazione del database MySQL, che sarà accessibile solo da locale.

Per quanto riguarda la configurazione del server DNS, che in questo contesto assumiamo essere **bind9**, bisogna copiare lo *zone file* d'esempio presente in `ser_ims/cfg/open-ims.dnszone` e inserirlo nella cartella di configurazione del server DNS, che solitamente è `/etc/bind`. Pertanto:

```
cp /opt/OpenIMSCore/ser_ims/cfg/open-ims.dnszone /etc/bind/
```

Inoltre, dal momento che stiamo imponendo il funzionamento in locale, dobbiamo togliere il # di commento alla riga `prepend domain-name-servers 127.0.0.1` nel file `/etc/dhcp3/dhclient.conf`. Bisogna quindi aggiungere le seguenti righe al file `/etc/bind/named.conf.local`:

```
zone "open-ims.test" {
        type master;
file "/etc/bind/open-ims.dnszone";
};
```

Dopo aver riavviato il server DNS con il comando `sudo /etc/init.d/bind9 restart`, è necessario imporre che le entità di OpenIMS Core sappiano di dover contattare il server DNS bind per la risoluzione dei nomi; quindi si aggiungono le seguenti righe al file `/etc/resolv.conf`:

```
search open-ims.test
nameserver 127.0.0.1
```

Verifichiamo dunque che tutto sia stato fatto in maniera corretta effettuando il ping di uno dei moduli di OpenIMS Core, mediante il comando `ping pcscf.open-ims.test`.

Si deve ora configurare il database MySQL che, come prescrive l'architettura IMS (e così avviene anche in OpenIMS Core) è necessario a conservare le informazioni accedute dall'I-CSCF e dall'HSS. Con i comandi seguenti è quindi possibile collocarvi le tabelle di default, contenenti alcuni dati per poter effettuare dei test e permettervi l'accesso da parte di OpenIMS:

```
mysql -u root -p -h localhost < ser_ims/cfg/icscf.sql
mysql -u root -p -h localhost < FHoSS/scripts/hss_db.sql
mysql -u root -p -h localhost < FHoSS/scripts/userdata.sql
```

Per facilitare l'avvio delle varie componenti, copiamo gli script di esecuzione e i relativi file di configurazione dei CSCF e del FHoSS nella cartella `/opt/OpenIMSCore`, quindi mediante il comando:

```
cp /opt/OpenIMSCore/ser_ims/cfg/* /opt/OpenIMSCore
```

E' ora possibile lanciare tutte le entità di OpenIMSCore semplicemente attraverso il comando `/opt/OpenIMSCore/nome_entità.sh` , quindi per un'esecuzione parallela delle stesse si devono dare i seguenti comandi:

```
/opt/OpenIMSCore/pcscf.sh
/opt/OpenIMSCore/scscf.sh
/opt/OpenIMSCore/icscf.sh
/opt/OpenIMSCore/fhoss.sh
```

Il comportamento di default prevede che nei terminali relativi ai CSCF vengano emessi periodicamente dei messaggi di log, contenenti i dati degli eventuali utenti registrati alla rete e dei collegamenti DIAMETER aperti. A questo punto, è anche possibile accedere all'interfaccia web per il controllo e la configurazione di FHoSS, collegandosi all'indirizzo `http://localhost:8080`; alla pagina **User Profile** possiamo notare che il FHoSS ha già due utenti di base registrati, corrispondenti alle identità private `alice@open-ims.test` e `bob@open-ims.test`. E' comunque possibile eliminarle o in alternativa aggiungere nuove identità cliccando sul tasto **Create** nella colonna di sinistra, dopo aver selezionato la voce '**User Identities**' (si veda Figura 3.5); analogamente si possono compiere le stesse azioni sulle identità pubbliche degli utenti.

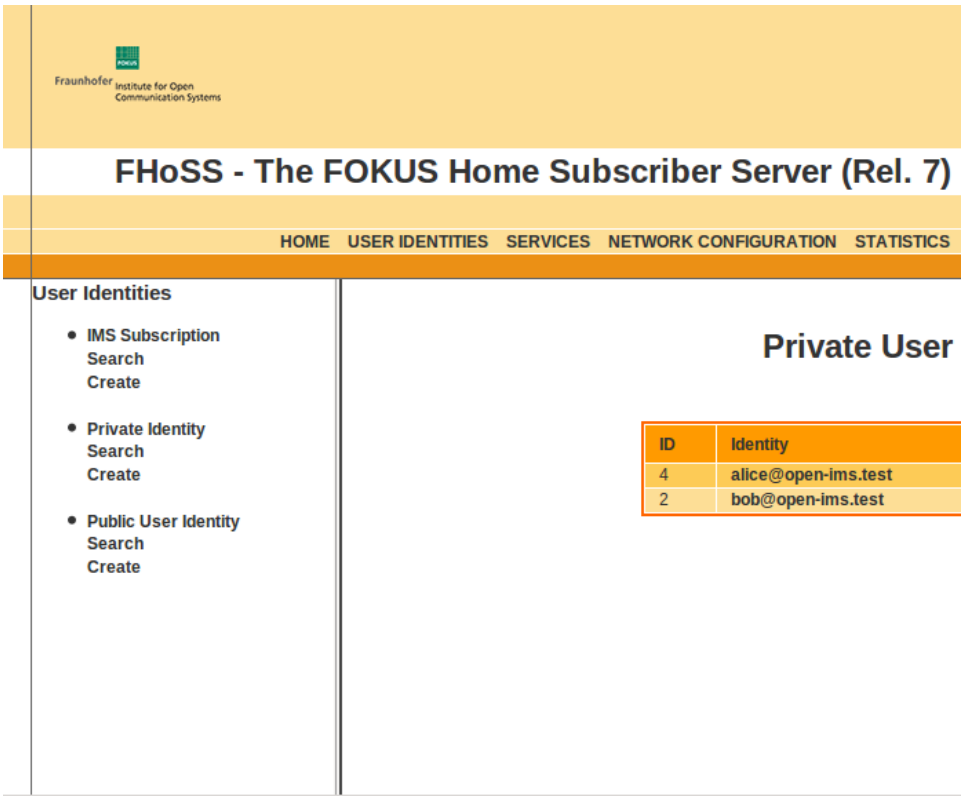


Figura 3.5: Voci della schermata User Identities nell'interfaccia Web di FHoSS

3.3.4 Test - Configurazione dei client IMS

A questo punto è possibile testare il corretto funzionamento della rete IMS appena creata mediante, ad esempio, l'instaurazione di una chiamata tra le due identità già presenti nella rete, ovvero Alice e Bob. Per fare ciò utilizzeremo il client IMS '**myMONSTER**' [37] sviluppato dallo stesso Fraunhofer Institute FOKUS: MONSTER non è altro che l'acronimo di Multimedia Open InterNet Services and Telecommunication EnviRonment. Il framework MONSTER è un insieme di classi ed interfacce Java il cui scopo è fornire una struttura applicativa omogenea per l'accesso a vari tipi di servizi da piattaforme differenti. A questo proposito si possono definire piani diversi relativi a livelli logici distinti (Figura 3.6):

Come si vede, il *device layer* prevede l'utilizzo di dispositivi differenti: dai PC Desktop (Windows, Linux, Mac) ai dispositivi mobile basati su Windows o su Android; in corso di sviluppo è il supporto per sistemi Symbian. Più sopra, il *Java Layer* permette di ottenere tali livelli di portabilità, presuppone quindi la presenza di una Java Virtual Machine sul dispositivo.

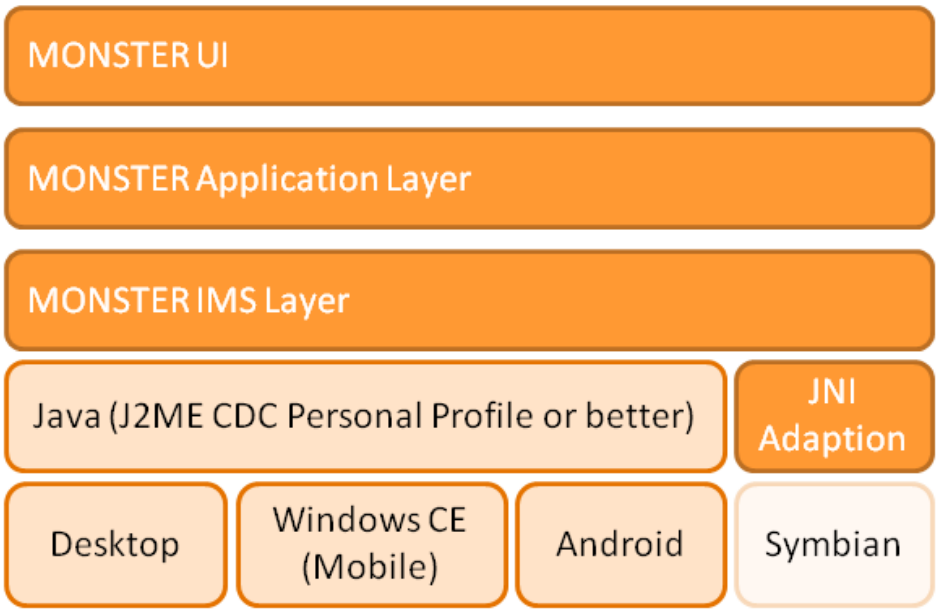


Figura 3.6: Struttura del framework MONSTER

Infine, l'*IMS layer* fornisce le funzionalità necessarie al superiore *Application layer*, contenente tutte le interfacce mostrate alle applicazioni dal livello sottostante. Ipotizziamo che ad instaurare la chiamata siano due client installati su PC Desktop Linux: dopo esserci procurati il pacchetto corretto nel sito [37], sarà sufficiente scompattare l'archivio nella locazione preferita, e avviare il client myMONSTER tramite il comando `./monster` direttamente dalla cartella appena estratta. A questo punto è necessario configurare correttamente il client con le informazioni relative alla rete IMS di appartenenza, che assumeremo essere quella appena creata con OpenIMSCore. Pertanto, nel caso ci si trovi sul PC di Alice (Figura 3.7):

```
Dominio: open-ims.test
Display Name: Alice
Public Identity: sip:alice@open-ims.test
Private Identity: alice@open-ims.test
Secret Key: alice
Proxy-CSCF: pcscf.open-ims.test
Proxy-CSCF Port: 4060
```

Dopo aver impostato il PC di Bob in maniera analoga, è possibile instaurare la chiamata tra le due parti inserendo un nuovo contatto (con le relative informazioni SIP) e cliccando su 'Audio Call' oppure semplicemente inserendo l'indirizzo SIP del destinatario nel campo presente nella schermata 'Call'.

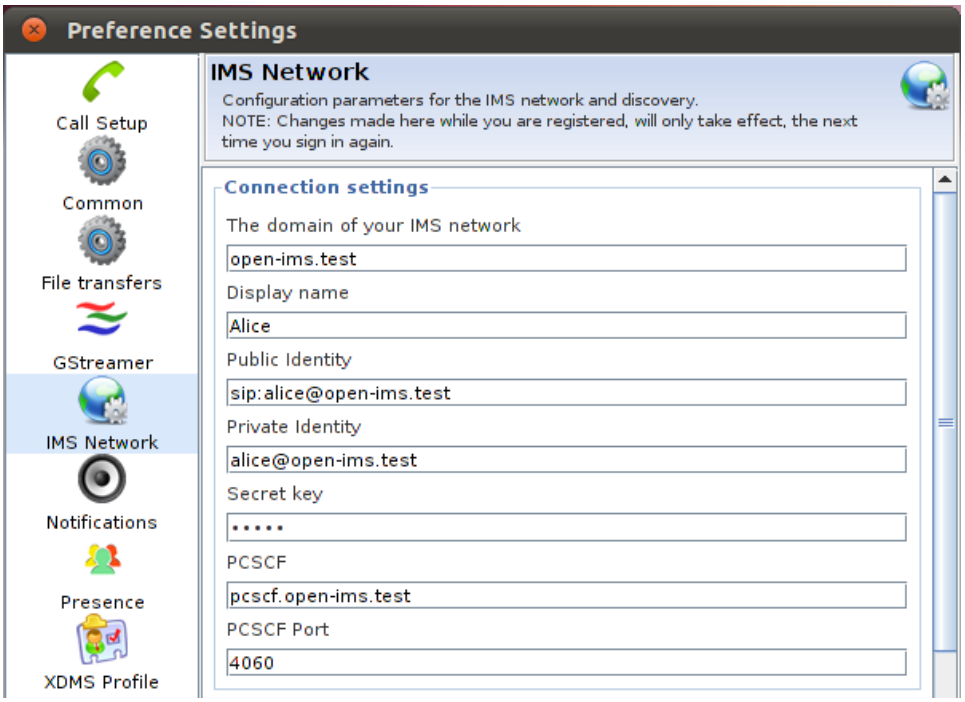


Figura 3.7: Schermata del client myMONSTER per la configurazione dei dati di rete IMS

3.3.5 Test - Analisi del flusso di messaggi SIP

Come già detto, nonostante di norma i vari CSCF e l'HSS si trovano in macchine differenti, è possibile testarne il funzionamento facendoli risiedere tutti nello stesso host e avviandoli in parallelo. Bisogna solo specificare porte differenti per le entità, che nel nostro test sono:

- P-CSCF: `4060`;
- I-CSCF: `5060`;
- S-CSCF: `6060`;
- FHoSS:
  - `8080`: server Tomcat in ascolto (per accesso all'interfaccia web);
  - `3868`: messaggi DIAMETER.

Registrazione

Il protocollo SIP e le specifiche della rete IMS obbligano gli utenti che desiderano interagire con la rete o con altri dispositivi a cominciare le loro operazioni con la fase di **registrazione** (come accennato nel Par. 2.7):

in questo modo si può identificare l'utente e conoscere i servizi ai quali può accedere. Nello specifico, la fase di registrazione prevede uno scambio

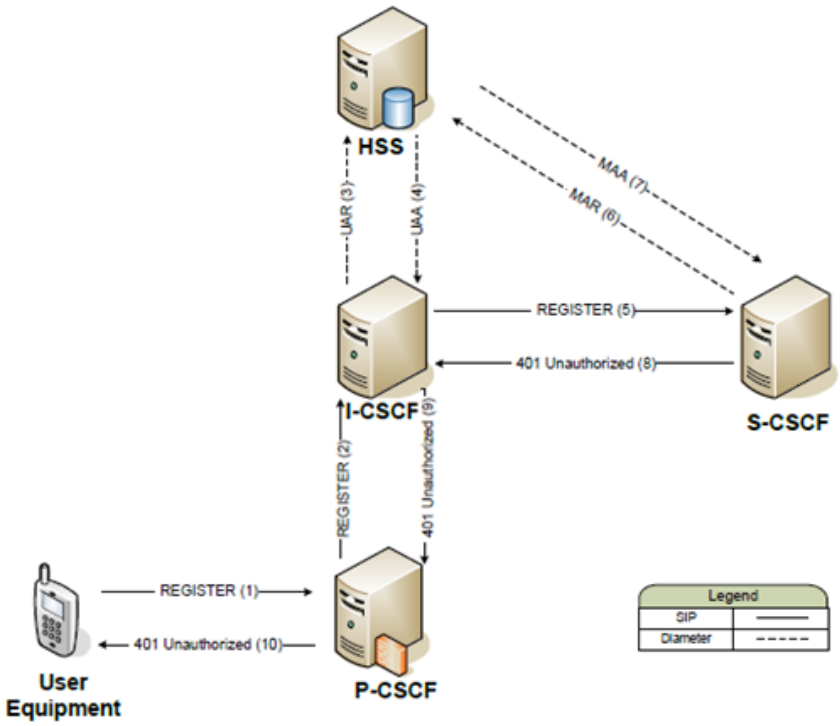


Figura 3.8: Prima fase di Registrazione

di messaggi tra l'utente, i vari CSCF e l'HSS: questo scambio avviene in due fasi dette rispettivamente *Challenge of Network to UE* e *Response of challenge by UE*. Nella prima fase (Figura 3.8), l'User Equipment conosce soltanto l'indirizzo del Proxy-CSCF (`pcscf.open-ims.test`, inserito durante la configurazione): invia quindi un messaggio di `SIP REGISTER` (1) ad esso, contenente la sua identità e il dominio dell'I-CSCF; questo perché il P-CSCF effettua un'interrogazione al server DNS per ottenere l'indirizzo di tale I-CSCF[1], e successivamente vi inoltra il messaggio di `SIP REGISTER` (2); l'I-CSCF interroga quindi l'HSS (mediante il protocollo DIAMETER) con un messaggio di User Authentication Request o `UAR` (3), cui segue la risposta User Authorization Answer (4) contenente le informazioni dell'S-CSCF al quale l'utente è assegnato. Il pacchetto `SIP REGISTER` viene quindi inol-

[1]Come già detto, un utente può trovarsi in un dominio differente rispetto a quello della sua H*ome Network*, ma l'S-CSCF che dovrà contattare dovrà appartenere a quest'ultima. Per questo motivo, nel messaggio di `SIP REGISTER` l'utente rende noto il suo dominio di appartenenza.

trato all'S-CSCF di competenza (5), che crea una sessione di autenticazione inviando un messaggio `DIAMETER MAR` (Multimedia Authorization Request) (6); ad esso segue la risposta dell'HSS, sotto forma di messaggio `DIAMETER MAA` (Media Authentication Answer) (7): è compito dell'S-CSCF recuperare i vettori di autenticazione necessari alla creazione della *Security Association* tra il P-CSCF e l'utente. Queste informazioni verranno quindi inserite all'interno di un messaggio di risposta `SIP 401` (Unauthorized) seguendo il percorso a ritroso, quindi S-CSCF, I-CSCF, P-CSCF e infine l'utente (8-9-10); il P-CSCF si occupa inoltre di eliminare la cifratura dai vettori di autenticazione qualora il messaggio sia inviato all'UE.

Provando ad effettuare la registrazione alla rete OpenIMS con l'utilizzo del client myMONSTER, si possono visualizzare tramite Wireshark (Figura 3.9) i pacchetti SIP seguenti (non vengono visualizzati i pacchetti DIAMETER):

Protocol	Info	
SIP	Request: REGISTER sip:open-ims.test	(1)
SIP	Request: REGISTER sip:open-ims.test	(2)
SIP	Request: REGISTER sip:scscf.open-ims.test:6060	(5)
SIP	Status: 401 Unauthorized - Challenging the UE	(8)
SIP	Status: 401 Unauthorized - Challenging the UE	(9)
SIP	Status: 401 Unauthorized - Challenging the UE	(10)

Figura 3.9: Prima fase di Registrazione - Pacchetti SIP

Nella seconda fase di registrazione, l'User Equipment genera quindi un messaggio di `SIP REGISTER` contenente i dati di autenticazione corretti, e lo inoltra al P-CSCF (11); quest'ultimo inoltra il messaggio `SIP REGISTER` all'I-CSCF (12), che riconosce il S-CSCF assegnato. Il S-CSCF, dopo aver ricevuto il messaggio di `REGISTER` (13) verifica se le informazioni di risposta alla *challenge* sono corrette e scarica i dati dell'utente dall'HSS mediante i messaggi `DIAMETER SAR` (Server Assignment Request) (14) e `SAA` (Server Assignment Answer) (15). Poiché l'utente risulta registrato, l'S-CSCF genera il messaggio `SIP 200 OK` che viene inoltrato a ritroso fino all'UE (16-17-18). La Figura 3.11 mostra tale flusso di messaggi: Analogamente a quanto fatto per la prima fase, riportiamo i pacchetti SIP intercettati con Wireshark (Figura 3.9). Al termine della seconda fase, l'utente risulta registrato alla rete, e pertanto potrà usufruire dei servizi messi a disposizione dagli Application Server residenti in essa. Si noti che il S-CSCF deve effettuare periodicamente un *refresh* della registrazione, affinchè questa non venga cancellata allo scadere di un determinato timeout. Infine, qualora lo desiderasse, l'utente può eliminare la propria registrazione inviando un messaggio di `SIP REGISTER` con il valore del timer della registrazione pari a 0.

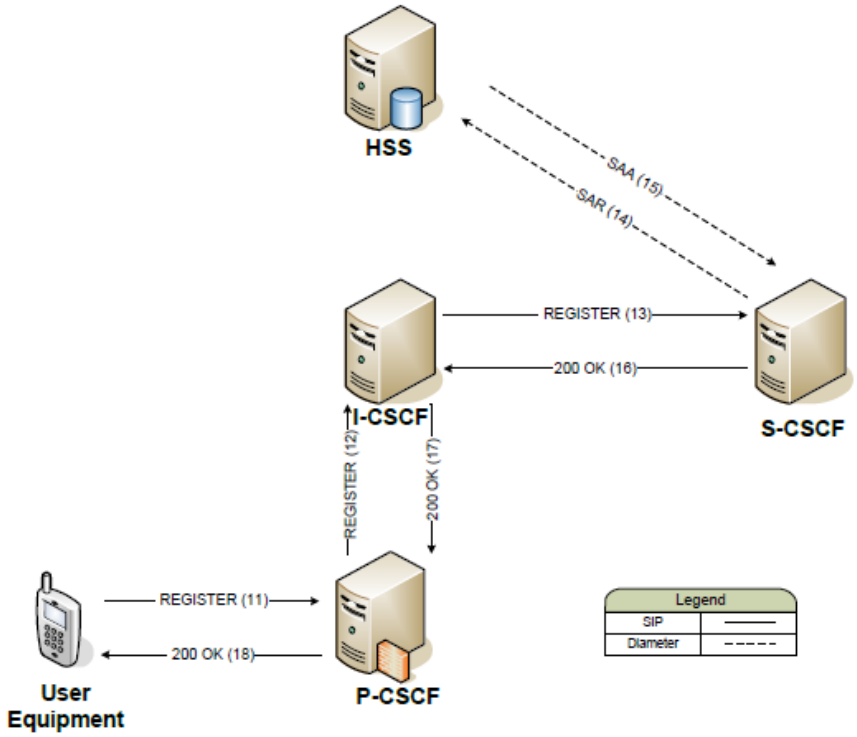


Figura 3.10: Seconda fase di Registrazione

Protocol	Info	
SIP	Request: REGISTER sip:open-ims.test	(11)
SIP	Request: REGISTER sip:open-ims.test	(12)
SIP	Request: REGISTER sip:scscf.open-ims.test:6060	(13)
SIP	Status: 200 OK - SAR succesful and registrar saved	(16)
SIP	Status: 200 OK - SAR succesful and registrar saved	(17)
SIP	Status: 200 OK - SAR succesful and registrar saved	(18)

Figura 3.11: Seconda fase di Registrazione - Pacchetti SIP

Nel nostro esempio, sia Alice che Bob devono portare a termine la procedura di registrazione per poter comunicare tra di loro.

Instaurazione di una chiamata

Ipotizziamo ora che Alice desideri contattare Bob, ad esempio per instaurare una chiamata vocale: prima di poter essere messi in contatto, sono necessarie una serie di operazioni di negoziazione delle rispettive *capabilities*, ovvero le funzionalità supportate dai due dispositivi.

Alice e Bob appartengono alla medesima Home Network, pertanto sono gestiti dal medesimo S-CSCF; inoltre assumiamo che utilizzino due diversi P-CSCF, e che entrambi abbiano installato il client IMS myMONSTER. Alice deve innanzitutto digitare nella schermata Call la SIP URI relativa

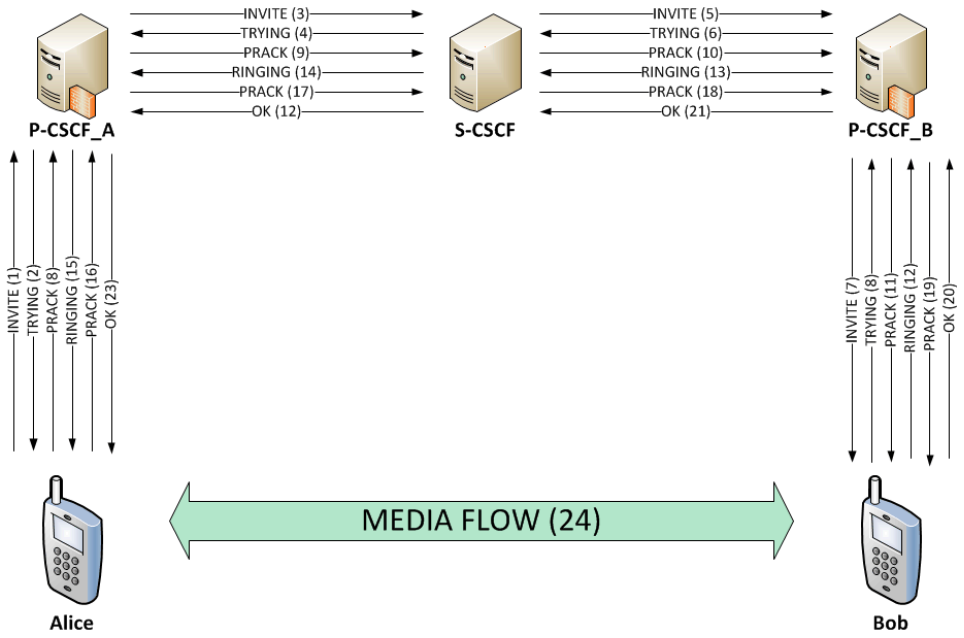


Figura 3.12: Instaurazione di una chiamata

a Bob, quindi `sip:bob@open-ims.test`; a questo punto, cliccando sul tasto 'Call' inizia la procedura vera e propria di instaurazione della chiamata. Per prima cosa Alice invia la richiesta `SIP INVITE` al P-CSCF_A ((1), Figura 3.12): quest'ultimo risponde ad Alice con il messaggio `SIP 100 TRYING` (2) ; lo scopo del messaggio `TRYING` è informare il chiamante che la richiesta è stata ricevuta dal P-CSCF, e che questo sta svolgendo le operazioni necessarie. Allo stesso modo il messaggio `INVITE` viene inoltrato al S-CSCF (3), il quale si occupa di inviarlo al P-CSCF_B (5), per poi giungere a Bob (7); in tutti questi passaggi c'è sempre un messaggio di TRYING che viene inviato all'entità precedente (4-6). Bisogna specificare che il messaggio di `INVITE` contiene come payload un pacchetto SDP, che serve per la procedura di *offer/answer* delle funzionalità: questa fase richiede l'intervento del modulo PDF (Policy Decision Function), il quale è a conoscenza dei codec supportati dai vari utenti presenti nella rete; una volta conclusa questa fase il chiamante (Alice) invia un messaggio `PRACK` verso Bob (8-9-10-11) contenente le informazioni sul codec selezionato e altre informazioni riguardanti la QoS. Se le risorse richieste sono allocabili, il chiamato (Bob) visualizza la chiamata in arrivo, e prima della risposta invierà verso Alice messaggi `SIP`

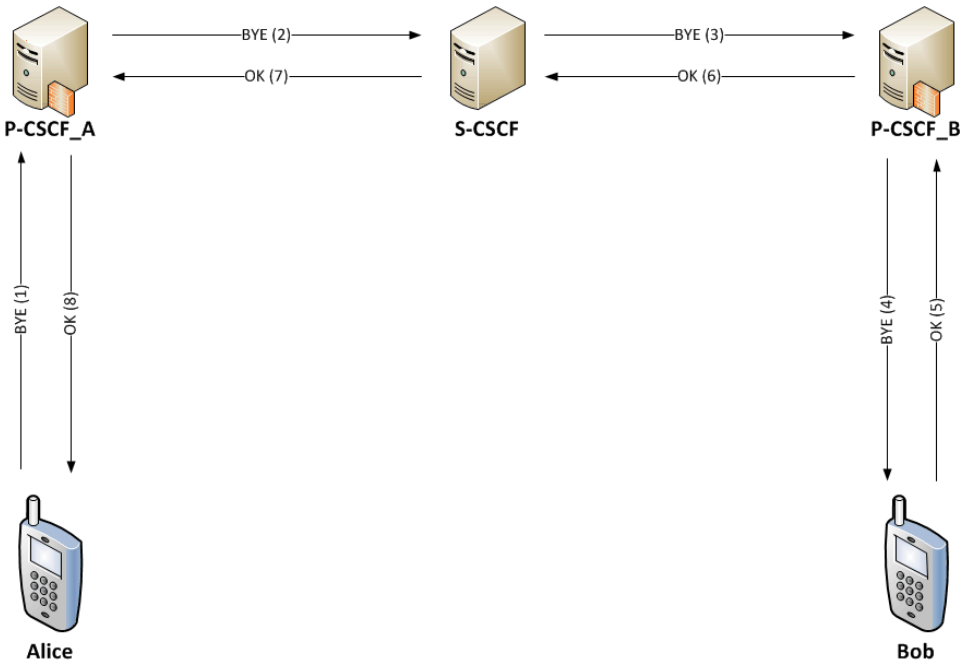


Figura 3.13: Chiusura di una chiamata

`180 RINGING` (12-13-14-15), per avvertirla che la chiamata gli sta venendo notificata. Con un meccanismo simile a quanto fatto in precedenza, Alice risponde con i messaggi `PRACK` (16-17-18-19). Quando Bob risponde alla chiamata, viene inviato ad Alice il messaggio `SIP 200 OK` (20-21-22-23), che conferma la riuscita delle operazioni e l'inizio della conversazione. A questo punto il flusso voce può avvenire con il protocollo selezionato, ad esempio RTP o RTPC (24).

Chiusura della chiamata

Se ad esempio Alice intende chiudere la sessione SIP instaurata con Bob, può inviare un messaggio `SIP BYE` al P-CSCF_A ((1), Figura 3.13) che lo inoltrerà come di consueto fino ad arrivare a Bob (2-3-4). A questo punto, Bob deve semplicemente inviare un messaggio `SIP OK` a ritroso fino a raggiungere Alice (5-6-7-8). Si noti che in questo tipo di operazione non intervengono né l'I-CSCF né l'HSS.

Capitolo 4

Lo strato dei servizi applicativi

Indice

Come già accennato, un Application Server (AS) fornisce un servizio specifico a determinati utenti della rete: applicazioni tipiche supportate in una rete IMS sono ad esempio quelle per videoconferenza, instant messaging, presence, content sharing, ecc.

A seconda dell'implementazione, un AS può ospitare una o più applicazioni: in entrambi i casi, comunque, esso riceve ed interpreta i messaggi SIP provenienti dal S-CSCF (attraverso l'interfaccia ISC), per poi tradurre i risultati delle elaborazioni in nuovi messaggi SIP diretti ancora una volta al S-CSCF, per mezzo del quale raggiungeranno poi l'utente finale.

Un IMS Service Provider può disporre di numerosi AS all'interno del medesimo dominio, ed ognuno può essere destinato ad un determinato tipo di applicazione/servizio o a specifici gruppi di utenti. L'utilizzo delle funzionalità offerte è quindi gestito dallo stesso S-CSCF, che riceve dall'HSS il *service profile* associato all'utente, ed applica gli *Initial Filter Criteria* secondo le procedure descritte nel Paragrafo 2.8.

Le specifiche di IMS non precisano un'architettura interna per gli AS: per capire come produrre applicazioni e servizi per gli utenti finali, è necessario esaminare le tecnologie che possono essere utilizzate per costruire gli AS. Non tutte sono equivalenti, dal momento che alcune vengono utilizzate per la creazione dell'intero server, mentre altre permettono di sviluppare solamente un layer di tale server; inoltre, alcune tecnologie sono *SIP-dependent* (e si possono utilizzare solo per creare SIP AS) mentre altre sono *protocol-independent*. Si può dare una classificazione di tali tecnologie in 3 famiglie [38]:

- Tecniche di programmazione SIP;
- API;
- Service Execution Environments.

Segue una panoramica delle caratteristiche di ognuna.

4.1 Tecnologie implementative

4.1.1 Tecniche di programmazione SIP

Le tecniche di programmazione SIP permettono allo sviluppatore di accedere alle funzionalità di base del protocollo SIP al fine di creare applicazioni basate su di esso. In questa categoria si può operare un'ulteriore distinzione:

- **SIP-CGI (SIP Common Gateway Interface)**: ispirata all'HTTP CGI, strumento per la creazione di contenuti dinamici per il web; quando un server riceve una richiesta SIP, esso invoca uno script SIP CGI: il server passa il corpo del messaggio allo script, e imposta delle variabili d'ambiente contenenti informazioni riguardanti l'header del messaggio, l'utente e la configurazione del server. Lo script effettua le dovute elaborazioni, dopodiché può indicare al server di generare una risposta SIP, effettuare il proxy della richiesta, crearne una nuova o modificarne l'header.

  In un AS IMS basato su SIP-CGI i servizi sono scritti sotto forma di script CGI: il server invoca dunque un certo script per ogni richiesta in ingresso, il quale genera una lista di azioni che il server deve eseguire per portare a termine tal richiesta.

- **SIP Servlet**: le SIP Servlet API sono un'estensione delle API Java per l'utilizzo in server SIP (analogamente al concetto di HTTP Servlet) [15]. Una SIP Servlet è un'applicazione Java contenuta all'interno di un *SIP Servlet Container*: quest'ultimo permette alle Servlet di comunicare con i client SIP mediante lo scambio di oggetti che rappresentano i messaggi SIP ricevuti dal container stesso. Le Servlet hanno così accesso agli header di tutti i messaggi SIP, e mediante le informazioni recuperate da essi possono creare, inoltrare o modificare messaggi SIP. La Servlet si occupa dunque della sola componente di alto livello della segnalazione, delegando al container le altre funzionalità richieste.

  Nel gergo IMS, un AS SIP è dunque il Servlet Container, mentre le Applicazioni (o Servizi) IMS sono le Servlet. Quando l'AS riceve una richiesta SIP, applica delle regole preconfigurate che gli permettono di stabilire a quale Servlet inoltrare l'oggetto che rappresenta tale richiesta; quest'ultima effettua le operazioni richieste e invoca l'intervento del container per inviare i messaggi prodotti.

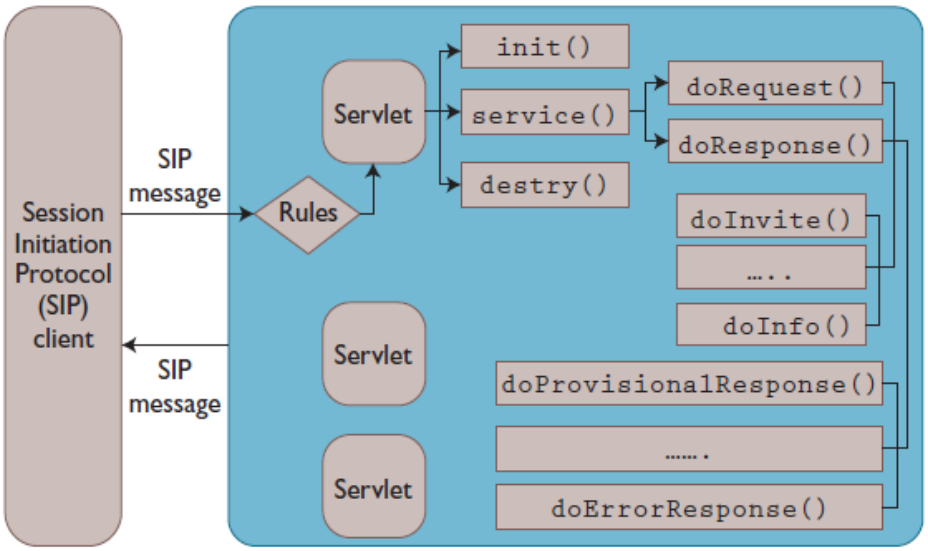


Figura 4.1: Schema di un server IMS SIP Servlet-based

4.1.2 API

Molte API destinate alla creazione di *communication server* possono essere adoperate come parte di un IMS AS. Le API forniscono interfacce object-oriented di alto livello che permettono ai programmatori di implemen-

tare applicazioni per vari tipi di comunicazione. Un esempio è rappresentato da OSA/Parlay [39], delle API che permettono lo sviluppo di servizi aventi funzionalità di controllo delle chiamate, messaging, charging, presence, ecc.

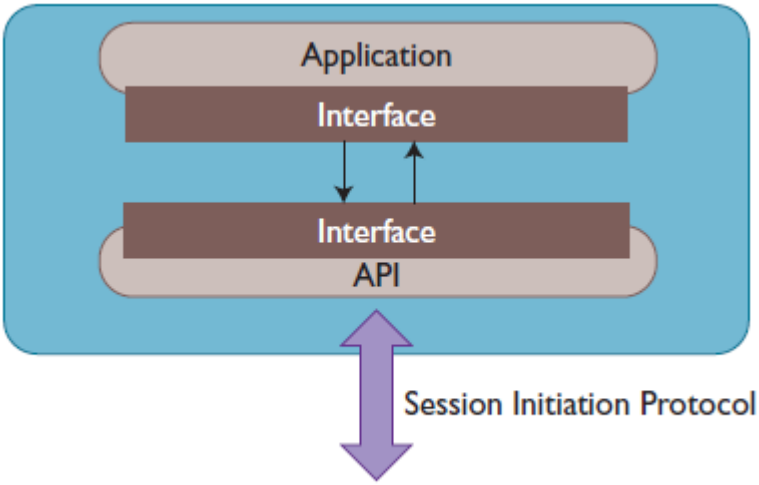


Figura 4.2: Schema di un server IMS API-based

4.1.3 Service Logic Execution Environment (SLEE)

Per Service Logic Execution Environment (SLEE) si intende un'architettura che permetta la creazione di applicazioni orientate agli eventi, dotate di caratteristiche di elevato throughput e bassa latenza, ideali quindi per l'utilizzo nell'ambito delle comunicazioni. Jain SLEE [**?**] è la specifica Java del concetto di SLEE, che comprende un container (ovvero il *runtime execution environment*) e l'architettura interna dei servizi. Un servizio Jain SLEE è una collezione di componenti riutilizzabili ed object-oriented (*service building block - SBB*) in esecuzione all'interno del container: un SBB è quindi la componente software che invia e riceve gli eventi ed effettua le elaborazioni richieste. Una risorsa esterna, come lo SLEE container o un altro SBB, può generare tali eventi.

Nel gergo IMS, un AS SLEE-based è un JSLEE container, mentre le applicazioni IMS sono rappresentate dagli SBB.

4.2 Interazione con il S-CSCF

Come già accennato, il componente della rete IMS detto S-CSCF utilizza le informazioni contenute nel service profile di un utente per instradare

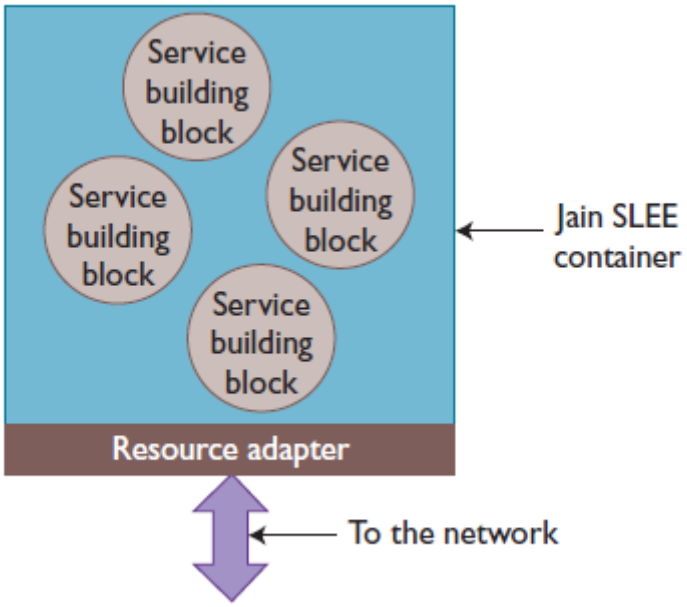


Figura 4.3: Schema di un server IMS JSLEE-based

eventuali richieste di servizio verso uno o più Application Server: questo tipo di interazione avviene mediante l'interfaccia ISC, che agisce come punto di collegamento tra le due entità. Come da specifiche 3GPP l'interfaccia ISC deve permettere:

- la notifica degli eventi tra gli AS e il S-CSCF, tra cui:
  - la registrazione di Public User Identity;
  - le funzionalità degli User Agent registrati alla rete;
- il filtraggio dei messaggi prima delle procedure di instradamento agli AS (mediante gli initial Filter Criteria - iFC);
- il riconoscimento della provenienza dei messaggi diretti o uscenti dal S-CSCF;

Per descrivere in maniera chiara queste interazioni è utile operare una distinzione dei flussi di messaggi a seconda che ci si trovi sul S-CSCF oppure su di un Application Server; i due paragrafi successivi cercheranno di raggiungere questo scopo.

4.2.1 La prospettiva del S-CSCF

Dopo aver ricevuto una richiesta SIP proveniente o diretta ad un utente collegato ad esso (in particolare una Public User Identity o una Public Ser-

vice Identity[1]), il S-CSCF esamina l'*initial Filter Criteria (iFC)* associato allo specifico profilo collegato all'identità: se il *trigger point* di un iFC risulta avere valore true, il S-CSCF inoltra il messaggio SIP al corrispondente Application Server. Come specificato in precedenza, il controllo degli iFC viene effettuato dal S-CSCF solo sui messaggi detti di *initial request*, ovvero quelli che servono ad instaurare una sessione cui faranno seguito altri messaggi di dialogo (esempi sono quindi i messaggi `INVITE`, `SUBSCRIBE`, `REFER`). Quindi un AS che vuole operare come proxy server o Back-to-Back User Agent deve specificare se intende o meno rimanere all'interno del percorso di segnalazione anche per i messaggi seguenti; questa decisione è registrata in un header specifico detto *Record Route*.

Inoltre, le specifiche 3GPP prevedono l'utilizzo di un particolare token, detto Original Dialogue Identifier (ODI), che il S-CSCF inserisce all'interno delle richieste provenienti dalla rete, prima che vengano inoltrate all'AS di competenza. Si tratta di una sorta di *secret word*, creata ed utilizzata dal S-CSCF, che ha un duplice scopo:

- Discriminare tra le richieste provenienti dalla *core network* e quelle che invece giungono dall'application layer; pertanto, quando una richiesta SIP contiene un *original dialogue identifier*, può essere associato ad una procedura che interessa l'interfaccia ISC.
- Qualora l'AS decida di agire come Back-to-Back User Agent tra due endpoint A e B, instaura un dialogo separato per ogni segmento: senza l'utilizzo di un *original dialogue identifier*, il S-CSCF non sarebbe in grado di concludere che i due canali di comunicazione sono in realtà appartenenti allo stesso dialogo. Questa informazione è utile ad esempio quando è necessario applicare un servizio di tariffazione.

Un altro comportamento da considerare è quando un **AS opera come User Agent**, ovvero è lui stesso l'iniziatore di una sessione SIP: questa eventualità si verifica ad esempio a seguito di una interazione con un utente o un web service, o ancora potrebbe essere un effetto prodotto dalla ricezione di una richiesta SIP; basti pensare al caso in cui, ricevendo un `SIP INVITE` proveniente da A e diretto a B, l'AS decide di inviare un messaggio istantaneo ad A,B e C. Qualora la richiesta sia stata iniziata per conto di un utente, l'AS inserisce l'identità pubblica di quest'ultimo nel messaggio, affinché venga effettivamente elaborato come proveniente dall'utente specificato. Tuttavia, il S-CSCF utilizza porte differenti per interfacciarsi con la core network e con gli Application Server: questi ultimi, in particolare, non sono a conoscenza della porta sulla quale inoltrare le richieste di iniziazione della sessione; questo problema viene risolto dall'AS inserendo all'interno di questi messaggi un particolare parametro, detto '**orig**', che consente al S-CSCF di trattare

[1] Per *Public Service Identity (PSI)* si intende l'identificatore di un determinato servizio erogato da un Application Server, una sua istanza o un gruppo di utenti che ne fanno uso.

le richieste provenienti dall'interfaccia ISC come se invece provenissero dalla *core network*.

Infine sorge il problema di notificare la registrazione di un client IMS agli Application Server: il messaggio `SIP REGISTER` non viene di norma inoltrato lungo l'interfaccia ISC, dal momento che il S-CSCF, agendo da *registrar*, è l'endpoint per questo tipo di comunicazioni. Tuttavia, l'informazione relativa alla registrazione/deregistrazione di un utente può essere fondamentale per un AS, e pertanto è sensato porsi il problema di come rendergliela nota. In genere, il S-CSCF invia un messaggio di `SIP REGISTER` all'AS dopo che l'utente ha effettuato la consueta registrazione: si tratta tuttavia di un messaggio di `REGISTER` 'ridotto', nel senso che non contiene tutte le informazioni del messaggio originale (ad esempio è assente la descrizione delle capabilities del device registrante). E' comunque possibile per un AS ottenere più informazioni mediante il meccanismo di `SIP SUBSCRIBE`: in questo modo egli riceverà i dettagli richiesti all'interno del corpo di un messaggio `NOTIFY` inviato dal S-CSCF; si noti tuttavia che, a differenza dell'inoltro del `SIP REGISTER` 'ridotto' da parte del S-CSCF all'AS, l'utilizzo di questo meccanismo non è una specifica propria dell'interfaccia ISC.

4.2.2 La prospettiva dell'Application Server

Nel **ricevere** un messaggio SIP, l'Application Server può comportarsi in maniera differente: in particolare può agire da:

- **SIP User Agent**, ovvero l'applicazione IMS che risiede nel server funge da endpoint per la comunicazione. Questo comportamento è tipico dei *presence server*, come pure delle richieste di cessazione per un particolare servizio erogato;

- **SIP Redirect**, cioè il server si limita a inoltrare la richiesta ricevuta verso un'altra entità;

- **SIP Proxy**, utile qualora il server debba eseguire delle operazioni *prima* dell'instaurarsi della comunicazione end-to-end, sul cui dialogo non potrà poi intervenire ulteriormente; può essere utilizzato per monitorare lo stato delle sessioni SIP attive;

- **Routeing B2BUA**, quando è invece richiesta una maggiore interazione sulla comunicazione rispetto al caso precedente; il server può decidere di apparire o meno come endpoint. Alcuni esempi possono essere la modifica di alcuni dati presenti nel corpo dei messaggi SIP (ad esempio i codec utilizzati), il trasferimento della sessione in corso, l'inserimento di contenuti multimediali nella comunicazione (ad esempio pubblicità).

Ad ogni modo, una funzionalità propria dell'interfaccia ISC è quella di distinguere tra i messaggi che provengono da una determinata identità IMS e quelli che invece terminano nella identità stessa, come pure tra utenti registrati alla rete e non: questo tipo di informazione può servire ad eseguire dei servizi differenti a seconda della **direzione di comunicazione** o allo **stato della registrazione**; si pensi ad esempio ad un servizio che permette di salvare momentaneamente un messaggio diretto ad una identità non registrata alla rete, salvo poi notificarlo non appena quest'ultima effettuerà la registrazione. Un modo per ottenere questo comportamento è ad esempio configurare gli *initial Filter Criteria* in maniera tale che l'inoltro di messaggi aventi direzioni differenti o provenienti da utenti non registrati venga effettuato verso AS distinti.

Un altro aspetto da considerare è quello riguardante l'**autenticazione** e le funzionalità correlate ad essa: entrano qui in gioco i cosiddetti **P-Header**, cioè particolari header (definiti in [40]) utili per l'interazione con gli Application Server. La richiesta che raggiunge un AS può contenere uno dei seguenti header:

- **P-Asserted-Identity**: permette di informare l'AS che l'utente cui si fa riferimento è stato autenticato nella core network IMS. Sulla base di questa informazione, l'applicazione potrà decidere di intraprendere ulteriori controlli nei confronti dell'identità o garantire l'accesso a determinate risorse ed applicare specifiche policy. Si noti che l'header *P-Asserted-Identity* può esistere anche per utenti che non appartengono alla rete dell'operatore, dal momento che il dominio di sicurezza IMS supera tali confini;

- **P-Access-Network-Info**: contiene informazioni sulla tecnologia d'accesso utilizzata dall'utente (es. xDSL, WiFi, UMTS) così da aiutare l'applicazione in merito alla modalità di consegna dei contenuti. Allo stesso modo può contenere informazioni sulla localizzazione dell'utente (es. ID della cella);

- **P-Visited-Network-ID**: rappresenta il nome della rete in cui l'utente potrebbe essere in *roaming*. In questo modo si possono applicare gli accordi tra operatori per quanto riguarda il *content delivery*;

- **P-Charging-Function-Addresses** e **P-Charging-Vector**: contengono rispettivamente l'indirizzo del nodo incaricato della tariffazione e l'ID da utilizzare per effettuare tali operazioni.

Nota: Bisogna osservare che l'utilizzo dei P-Header, così come l'implementazione delle funzionalità riguardanti la direzione della comunicazione e lo stato di registrazione, necessitano di una modifica delle caratteristiche degli Application Server SIP tradizionali, cioè non pensati esplicitamente per l'utilizzo nell'ambito IMS.

Infine, per quanto riguarda il **comportamento da User Agent dell'AS**, vale quanto già detto nel paragrafo precedente; alternativamente, l'AS può inserire nell'header *P-Asserted-Identity* le informazioni relative all'utente che vuole impersonare: poiché siamo nel *trust domain* della rete IMS, tale richiesta verrà correttamente autenticata all'interno della *core network*.

4.3 Utilizzo di un Application Server con Open IMS Core

Vediamo ora più da vicino come configurare un Application Server affinché possa interagire con la rete IMS descritta nel Paragrafo 3, ovvero OpenIMS Core.

Per effettuare i nostri test utilizzeremo la versione 2.0 di **SailFin** [41], un SIP Servlet Container basato sulla tecnologia delle SIP Servlet e sull'Application Server Sun Glassfish [42]: SailFin supporta quanto specificato nel documento JSR 289 [15], relativo alla versione 1.1 delle SIP Servlet.

4.3.1 Installazione e configurazione di SailFin

SailFin è un progetto Open Source che realizza un SIP Servlet Container e condivide molte caratteristiche con il suo equivalente commerciale sviluppato da Sun Microsystems chiamato Sun Java GlassFish Communications Server. Le differenze sostanziali tra i due prodotti risiedono nel prezzo e nelle performance: noi abbiamo scelto SailFin in quanto offre tutte le funzionalità necessarie al testing di piccole applicazioni. Requisiti per l'installazione sono:

- JDK 5 Update 8 (o superiore);
- Ant 1.6.5 (o superiore).

E' necessario prima di tutto procurarsi il file `.jar` presente nella sezione Downloads del sito ufficiale [41], dopodiché eseguire il comando per scompattare l'archivio e generare la directory '`sailfin`':

```
% java -Xmx256m -jar nome_della_versione.jar
```

Si entra nella directory appena creata e, dopo aver concesso i permessi d'esecuzione, si avvia `Ant`:

```
% chmod -R +x lib/ant/bin
% lib/ant/bin/ant -f setup.xml
```

A questo punto è possibile avviare SailFin in esecuzione sul dominio di default (`domain1`) con il comando seguente (all'interno della 'sailfin directory'/bin):

```
% ./asadmin start-domain domain1
```

In questo modo abbiamo avviato il Servlet Container, ma dobbiamo ancora configurarlo per potersi interfacciare in maniera corretta con la restante rete; in particolare è necessario inserire i dati relativi ai *SIP Listener*, ovvero i socket sui quali SailFin riceverà i messaggi SIP da inoltrare alla Servlet adeguata: uno è destinato alle comunicazioni tcp/udp (*SIP Listener 1*) mentre l'altro alle comunicazioni sicure TLS (*SIP Listener 2*). Impostiamo quindi le rispettive porte ai valori `9060` e `9061` mediante i comandi:

```
1 % ./asadmin set
2     server.sip-service.sip-listener.sip-listener-1.port
          =9060
3 % ./asadmin set
4     server.sip-service.sip-listener.sip-listener-2.port
          =9061
```

Allo stesso modo bisogna configurare le porte che potrebbero essere utilizzate da un'entità esterna qualora una Servlet decida autonomamente di inizializzare una sessione SIP (come descritto nel Paragrafo 4.2.1 quando si parla di 'AS come User Agent'): esse saranno le stesse utilizzate dai *SIP Listener*, ma affinché non risultino nascoste all'esterno è necessario eseguire i comandi seguenti:

```
1 % ./asadmin set server.sip-container.external-sip-port=9060
2 % ./asadmin set server.sip-container.external-sips-port
    =9061
```

Nota: le stesse operazioni effettuate con i comandi precedenti possono essere eseguite sfruttando l'interfaccia web offerta da SailFin: sarà sufficiente collegarsi all'indirizzo `http://localhost:4848` ed inserire come nome utente '`admin`' e password '`adminadmin`'.

E' ora possibile effettuare il *deploy* di una Servlet SIP e testarne il funzionamento utilizzando come rete IMS la precedentemente installata OpenIMS Core.

4.3.2 Interfacciamento con Open IMS Core

Una volta installato il SIP Servlet Container Sailfin, è necessario configurare OpenIMS Core affinché lo identifichi come Application Server ed invii verso esso le richieste SIP corrispondenti agli *initial Filter Criteria* specificati. Questo tipo di impostazioni vengono effettuate attraverso la schermata di amministrazione dell'HSS, quindi collegandosi all'indirizzo `http://localhost:8080`; ipotizziamo di voler configurare la rete affinché il S-CSCF inoltri all'Application Server (SailFin) i messaggi di SIP INVITE. Dobbiamo quindi:

- Aggiungere alla lista degli Application Server un nuovo server corrispondente a SailFin (Figura 4.4)

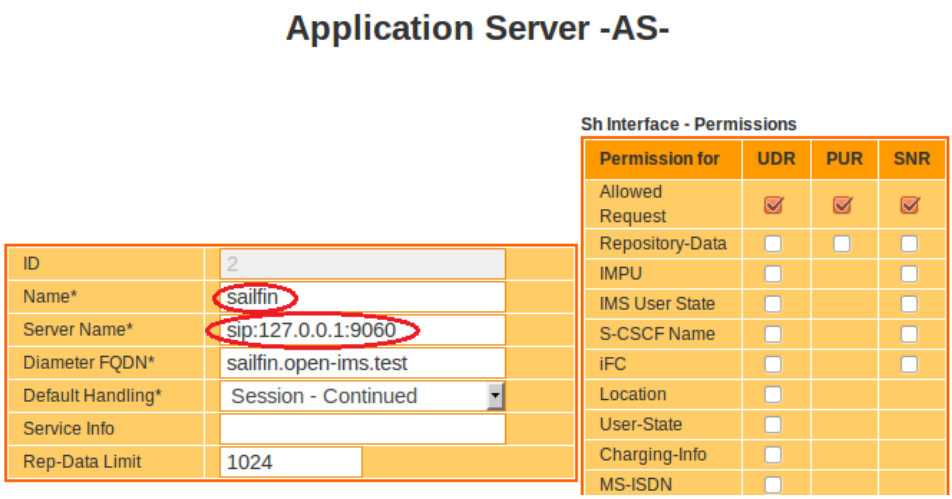


Figura 4.4: Schermata 'Application Server'

  - Selezioniamo **Services→Application Servers→Create**
  - Nel campo **Name** inseriamo `sailfin`
  - Nel campo **Server Name** inseriamo `sip:127.0.0.1:9060`
  - Nel campo **Diameter FQDN** (Fully Qualified Domain Name) inseriamo `sailfin.open-ims.test`

- Creiamo un nuovo trigger point, corrispondente alla ricezione di un messaggio di `SIP INVITE`
  - Selezioniamo **Services→Trigger Points→Create**
  - Nel campo **Nome** inseriamo ad esempio `sip_invite`
  - Nella sezione più in basso selezioniamo come **SIP Method** `INVITE` e come **Session Case** `Orig - Session`

- Creiamo un nuovo iFC, avente come trigger point quello appena specificato (Figura 4.5)
  - Selezioniamo **Services→Initial Filter Criteria→Create**
  - Nel campo **Nome** inseriamo ad esempio `sip_invite_ifc`
  - Nel campo **Trigger Point** selezioniamo `sip_invite`
  - Nel campo **Application Server** selezioniamo `sailfin`

Figura 4.5: Schermata *'initial Filter Criteria'*

- A questo punto torniamo nella schermata **Services→Trigger Point** (Figura 4.6) e cerchiamo quello chiamato `sip_invite`: al suo interno selezioniamo l'iFC appena creato alla voce **Attach iFC** e clicchiamo su **Attach**.

Abbiamo così configurato la *core network* affinché effettui l'inoltro dei messaggi di `SIP INVITE` all'Application Server SailFin precedentemente installato: a questo punto possiamo creare al suo interno una SIP Servlet che effettui delle elaborazioni su questi messaggi.

4.3.3 Struttura e funzionamento delle Servlet SIP

Le applicazioni SIP Servlet sono contenute in un archivio avente una precisa struttura e identificato dall'estensione `.sar`. Il *deploy* di una SIP Servlet in un Servlet Container implica il susseguirsi di una serie di operazioni, che vanno dalla corretta installazione dell'applicazione, alla sua esecuzione e infine alla disinstallazione. La Figura 4.7, estratta dalle specifiche Servlet 1.1, rappresenta appunto questo 'ciclo di vita'.

Inizialmente, il Servlet Container analizza il pacchetto `.sar` e cerca al suo interno la classe Java che rappresenta la Servlet: in questo momento siamo nello stato `Init()`, che consiste nel passaggio delle informazioni di configurazione dal *deployment descriptor* all'applicazione vera e propria. A questo punto la Servlet è in grado di offrire un servizio (ovvero è pronta ad essere invocata dal container quando quest'ultimo riceve uno specifico

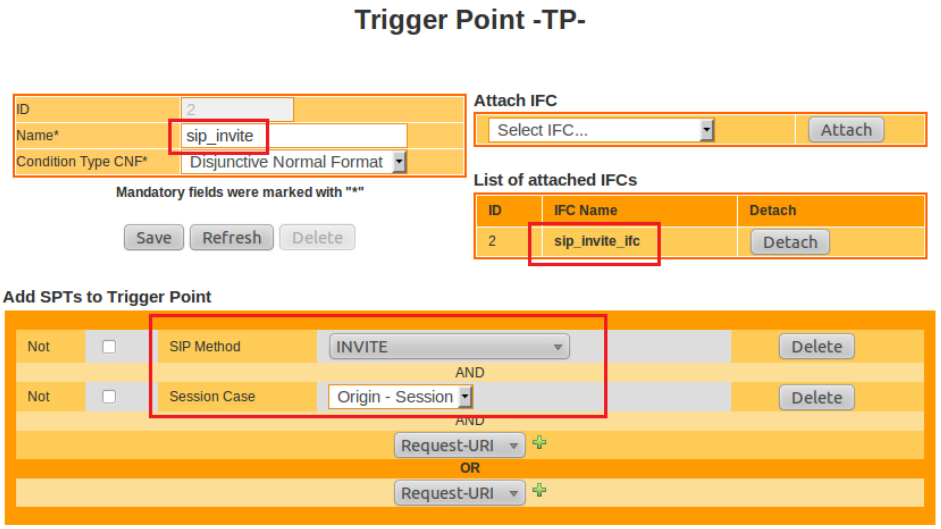


Figura 4.6: Schermata *'Trigger Point'*

messaggio SIP); in futuro il container, generalmente a seguito di una richiesta da parte dell'utente, potrà concludere l'esecuzione dell'applicazione, quindi effettuare l'*undeployment* della stessa (fase corrispondente all'arco `Destroy()`).

La specifica dell'interfaccia Servlet, della quale SIP Servlet è un'estensione (al pari delle HTTP Servlet), definisce il metodo `service`, che costituisce il veicolo primario nella ricezione e invio di messaggi: esso viene invocato ogni qual volta avvenga una richiesta o una risposta. La sua definizione è la seguente:

```
1 void service(ServletRequest req, ServletResponse res)
```

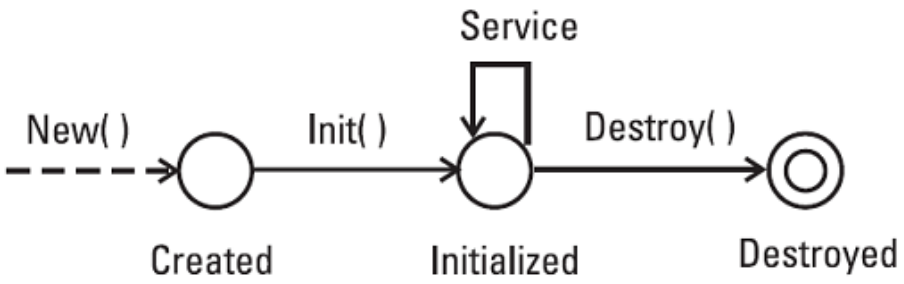


Figura 4.7: Il ciclo di vita di una SIP Servlet in un Servlet Container

```
2     throws ServletException, java.io.IOException
```

Il Servlet Container invoca quindi la versione SIP corretta di tale metodo: `SipServletRequest` per le richieste, `SipServletResponse` per le risposte. Inoltre, per evitare al programmatore di effettuare il *parsing* dei messaggi SIP al fine di scoprire il tipo di pacchetto (ad es. `INVITE`, `BYE`, ecc.), le SIP Servlet API mettono a disposizione i due seguenti metodi:

```
1 protected void doRequest(SipServletRequest req);
2 protected void doResponse(SipServletResponse req);
```

Il metodo `doRequest` analizza la richiesta appena ricevuta e invoca uno dei seguenti metodi a seconda della primitiva SIP contenuta nel messaggio (Figura 4.8):

- `doInvite`: invocato al ricevimento di un messaggio di SIP INVITE; analogamente per:
- `doAck`
- `doOptions`
- `doBye`
- `doCancel`
- `doRegister`
- `doPrack`
- `doSubscribe`
- `doNotify`
- `doMessage`
- `doInfo`
- `doUpdate`
- `doRefer`
- `doPublish`

Le specifiche SIP Servlet permettono di interagire anche con nuovi metodi di SIP non presenti tra quelli elencati: è sufficiente sovrascrivere il metodo `doRequest` e utilizzare 'super'. Ad esempio, se il parametro `NewRequest` rappresenta il nome di un nuovo metodo, un esempio di sovrascrittura può essere il seguente:

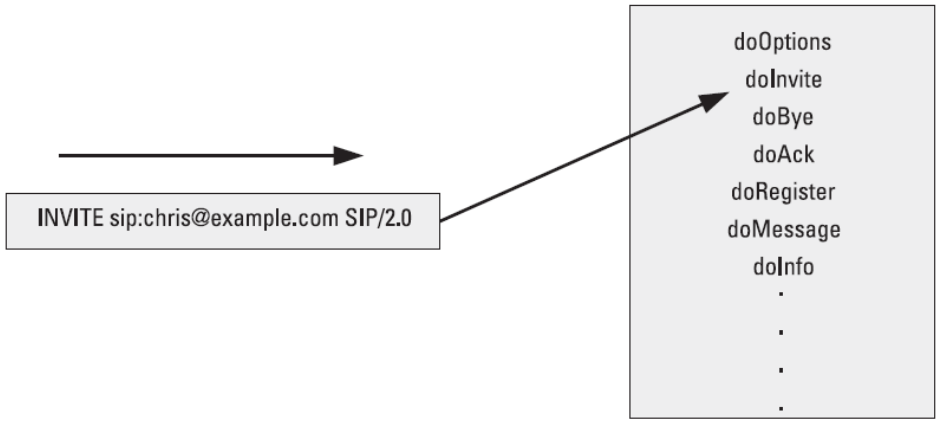


Figura 4.8: Meccanismo di invocazione del metodo corretto a seguito di una richiesta SIP

```
protected void doRequest(SipServletRequest request)
throws ServletException, IOException
{
        if (NewRequest.equals(request.getMethod()))
        {
                doNewRequest;
        }else
        {
                super.doRequest(request);
        }
}
```

Il funzionamento del metodo `doResponse` è del tutto analogo: l'ovvia differenza consiste nel fatto che quest'ultimo viene invocato alla ricezione di un messaggio di risposta SIP e, a seconda dello status code contenuto in esso, provoca l'invocazione di uno dei seguenti metodi (Figura 4.9):

- `doProvisionalResponse`: invocato alla ricezione di un messaggio SIP con codice compreso tra 101 e 199; analogamente per gli intervalli successivi:
- `doSuccessResponse`: 200-299
- `doRedirectResponse`: 300-399
- `doErrorResponse`: 400-699

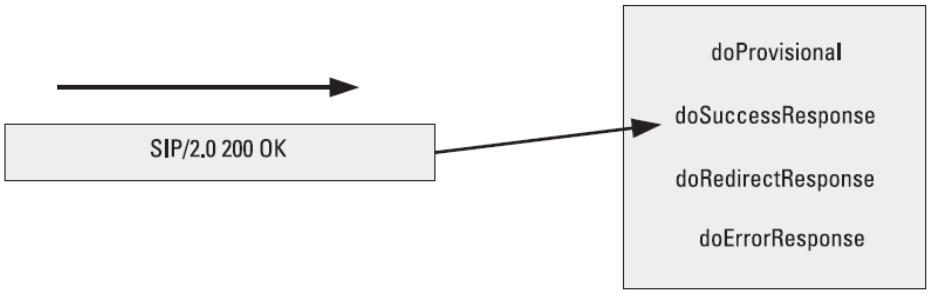


Figura 4.9: Meccanismo di invocazione del metodo corretto a seguito di una risposta SIP

4.3.4 Struttura dei pacchetti `.sar`/`.war`

I pacchetti di tipo `.sar` dispongono di una specifica struttura interna delle cartelle e dei file: in questo modo il container sul quale verranno posizionati avrà modo di accedere alle informazioni necessarie al deployment e al funzionamento della Servlet in maniera standard.

Nella root di una SIP Servlet è presente la cartella '`WEB-INF`': essa contiene tutto ciò che è strettamente collegato all'applicazione, ed il container si assicura della sua esistenza per poter effettuare le operazioni di deploy. All'interno di questa cartella vi sono poi i seguenti elementi:

- il file `sip.xml`, ovvero il *deployment descriptor*: contiene varie informazioni utilizzate sia dal container che dall'applicazione stessa;
- la cartella `classes`, contenente le classi della SIP Servlet che devono essere disponibili in fase runtime al container;
- la cartella `lib`, nella quale sono presenti gli archivi `.jar` utilizzati dalle applicazioni.

Va notato che la struttura dell'archivio `.sar` è analoga a quella dei pacchetti `.war`, utilizzati nell'ambito delle HTTP Servlet: per questo, qualora si tratti di una converged application (che prevede cioè l'interazione tra più canali di comunicazione - ad esempio Web, instant messaging, email, ecc.), all'interno della cartella `WEB-INF` potrà esserci sia il file `sip.xml` che il file `web.xml`; in questo caso l'archivio potrà essere indistintamente di tipo `.sar` oppure `.war`.

4.3.5 Implementazione di una semplice SIP Servlet

Utilizzeremo come ambiente di sviluppo la versione 6.9.1 di **Netbeans** [43], per il quale sono disponibili dei plugin che facilitano la creazione di

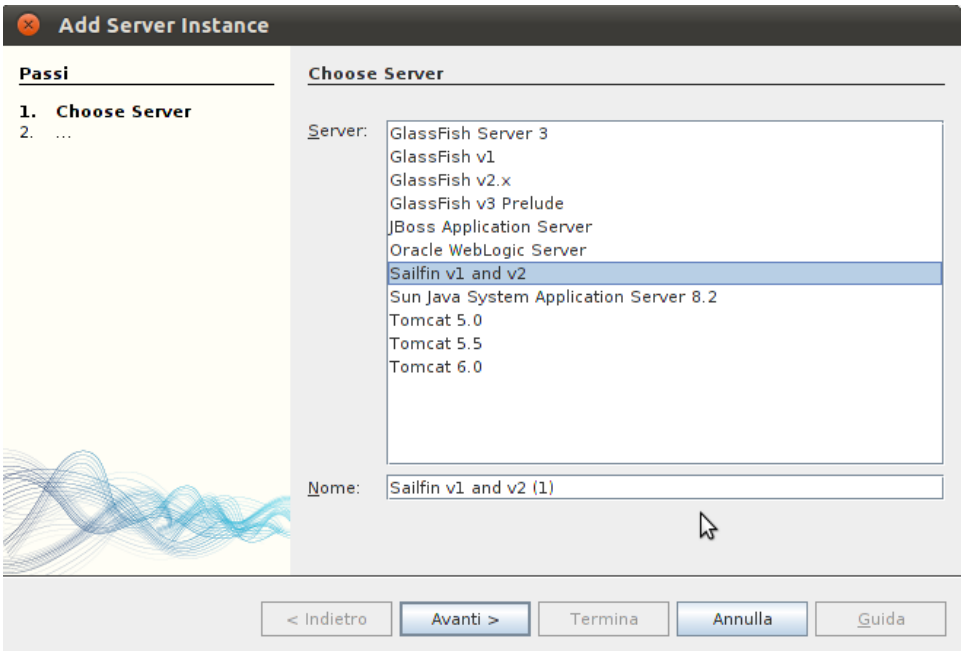


Figura 4.10: Configurazione del server Sailfin (1)

SIP Servlet; è possibile installarli direttamente dall'IDE, in particolare andando alla voce **Strumenti→Plugin** e selezionando poi **Aggiungi Plugin**: basterà selezionare i pacchetti .nbm presenti nella cartella `<sailfin dir>/lib/tools/netbeans` e procedere con l'aggiunta.

Possiamo ora aggiungere il supporto per il server SailFin, in modo che NetBeans effettui automaticamente l'avvio di quest'ultimo e il *deploy* delle Servlet create. E' sufficiente cliccare su **Strumenti→Server** e selezionare nell'elenco la voce **SailFin v1 and v2** (Figura 4.10).

Infine, dopo aver cliccato su Avanti, ci si presenterà la schermata di configurazione del server, nella quale dovremo semplicemente inserire i dati che relativi alla collocazione del server nel file system e l'informazione relativa al dominio da utilizzare (nel nostro caso `domain1` - Figura 4.11):

Il vantaggio principale nell'uso di NetBeans si ottiene quando andiamo a creare un nuovo progetto: l'IDE si occuperà infatti di produrre tutti i file necessari al corretto funzionamento della Servlet, compreso il *deployment descriptor* `sip.xml` e, successivamente, l'archivio `.sar` o `.war` necessario al *deploy* dell'applicazione nel Servlet Container.

Nella trattazione seguente descriviamo l'implementazione di una semplice SIP Servlet che effettua il redirect di una chiamata SIP, mediante l'inoltro dei messaggi di SIP INVITE ad un destinatario diverso da quello richiesto: non ha chiaramente senso utilizzare un Application Server solo per eseguire

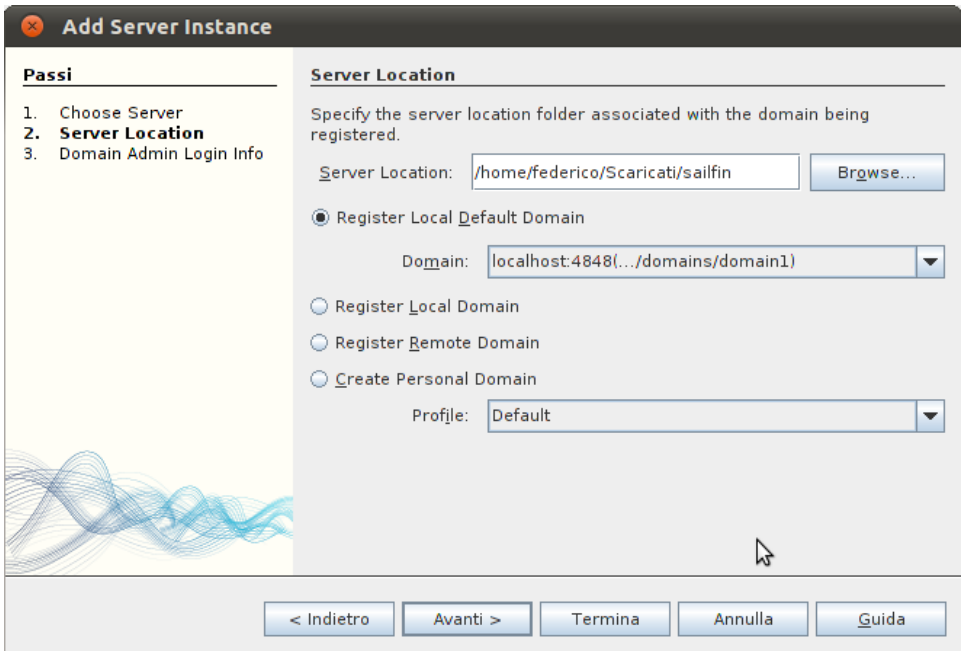


Figura 4.11: Configurazione del server Sailfin (2)

un'operazione così semplice, ma questo procedimento mette in luce le caratteristiche fondamentali delle SIP Servlet e del loro utilizzo in ambito IMS. Per questo motivo, l'AS si comporterà come Proxy, seguendo la definizione data in [5] nell'ambito SIP:

> [Per Proxy] si intende una entità intermedia che agisce sia come server che come client, allo scopo di effettuare richieste per conto di altri client. Un Proxy server svolge anzitutto compiti di instradamento, quindi assicura che una richiesta sia mandata ad un'altra entità 'più vicina' al destinatario finale. I Proxy possono svolgere funzioni di controllo (ad es. assicurarsi che un utente sia abilitato ad effettuare chiamate); inoltre interpreta, e se necessario riscrive, parti precise di un messaggio di richiesta prima di instradarlo.

Una richiesta che giunge alla Servlet attraverso il container è, come già detto, rappresentata mediante l'uso dell'interfaccia `SipServletRequest`: l'applicazione può funzionare da Proxy utilizzando l'oggetto omonimo offerto da tale interfaccia mediante l'invocazione del metodo `getProxy()`. Quindi:

```
1 SipServletRequest.getProxy();
```

restituirà un oggetto di tipo `Proxy` , che potrà essere configurato affinché l'AS svolga le operazioni di instradamento prescritte. Nel nostro caso sarà

sufficiente impostare l'URI di destinazione dei messaggi come parametro esplicito del metodo `proxyTo()`. A questo proposito dovremo ricorrere all'utilizzo dell'interfaccia `SipFactory`, che permette al programmatore di creare istanze di oggetti utili in ambito SIP, come ad esempio `Address`, `SipURI`, `SipServletRequest` e così via. Un'istanza di `SipFactory` può essere inserita nell'applicazione mediante l'uso dell'annotazione Java `@Resource`: successivamente sarà possibile invocare su di essa i metodi per la creazione di particolari oggetti SIP. Considerando quanto detto nel Paragrafo 4.3.3 relativamente ai metodi utilizzati per la gestione delle richieste SIP, la Servlet che effettuerà l'inoltro dei messaggi ad uno specifico indirizzo SIP presenterà il seguente metodo `doInvite`:

```
1  @Resource
2  SipFactory sf;
3  protected void doInvite(SipServletRequest request)
4          throws ServletException, IOException {
5                  if (request.isInitial()) {
6                          Proxy proxy = request.getProxy();
7                          proxy.setRecordRoute(true);
8          proxy.proxyTo(sf.createSipURI("bob","open-ims.test"))
               ;
9                  }
10         System.out.println("ProxyServlet:\n" + request);
11 }
```

In questo esempio la Servlet eseguirà l'inoltro dei messaggi SIP INVITE ricevuti dal container all'entità avente indirizzo `sip:bob@open-ims.test`, come specificato nel metodo `proxyTo()`. Inoltre:

- il metodo `isInitial()` ha valore `true`/`false` a seconda che la richiesta SIP sia iniziale o meno;
- il metodo `setRecordRoute()` serve per indicare al container di inserire un header SIP '`Record Route`' alla richiesta SIP uscente, in modo che anche le richieste successive (`ACK`, `BYE`) appartenenti allo stesso dialogo passino attraverso il Proxy;

  Nota: per quanto riguarda le risposte, il comportamento è differente: esse infatti attraversano sempre lo stesso percorso effettuato dalle richieste, pertanto le risposte riguardanti una determinata richiesta passeranno sempre per l'AS, indipendentemente dalla presenza o meno del suo indirizzo nel campo `Record-Route`.
- dopo la ricezione di un messaggio di `SIP INVITE` viene stampato un messaggio nel log del Servlet Container.

Il *deploy* e l'esecuzione della Servlet nel server Sailfin precedentemente installato avvengono in maniera automatica: cliccando alla voce 'Run' di NetBeans, quest'ultimo effettuerà tutte le operazioni necessarie, e potremo te-

stare l'effettivo funzionamento della Servlet instaurando una chiamata tra due utenti della rete IMS.

4.3.6 Analisi del traffico

Ipotizziamo di voler impostare l'Application Server affinché effettui il redirect di tutte le chiamate provenienti al S-CSCF della sua rete verso l'utente `sip:bob@open-ims.test`: questo è in effetti quanto viene svolto dalla Servlet d'esempio precedentemente illustrata. Siano rispettivamente `5062` e `5064` le porte corrispondenti ai terminali IMS di Alice e Bob (già registrati alla rete) e sia `9060` la porta sulla quale l'AS Sailfin è in ascolto; su di esso è in esecuzione la Servlet per il redirect creata in precedenza. Avvengono dunque le seguenti operazioni:

1. Alice chiama l'utente `sip:pippo@open-ims.test`: la richiesta di `SIP INVITE` viene inoltrata dal client al P-CSCF (1) (porta `4060`), il quale inserisce la sua SIP URI all'interno del campo `Record Route` con la dicitura `lr` alla fine (corrispondente a *loose routing*), cosicché le successive risposte passino nuovamente per tale nodo. Esso inoltra quindi la richiesta al S-CSCF (2).

2. Trattandosi di una *initial request*, il S-CSCF effettua il controllo degli *initial Filter Criteria* corrispondenti al service profile del mittente: poiché abbiamo impostato un *Trigger Point* per i messaggi di `SIP INVITE`, il S-CSCF inoltra il messaggio all'AS (3), effettuando le stesse operazioni svolte dal P-CSCF in merito al campo `Record Route` (ovviamente con il proprio indirizzo) ed in aggiunta inserendo l'indirizzo dell'AS e l'*Original Dialog Identifier (ODI)* nel campo `Route`: in questo modo, dopo le elaborazioni dell'AS, il messaggio tornerà ad esso (come spiegato nel paragrafo 2.8 relativo agli iFC).

3. Poiché nel codice abbiamo inserito `setRecordRoute(true)`, l'AS inserirà all'interno del messaggio appena ricevuto il suo indirizzo nel campo `Record Route`, al di sopra di quelli già presenti, per poter restare nel path delle richieste successive. A questo punto la SIP Servlet sostituisce la `Request-URI` del messaggio (attualmente `sip:pippo@open-ims.test`) con quella impostata per il *redirect*, ovvero `sip:bob@open-ims.test`. Il messaggio così modificato viene inviato al S-CSCF (4).

4. Poiché il messaggio ricevuto contiene nel campo `Route` l'*Original Dialog Identifier*, il S-CSCF sa che esso è stato inviato dall'AS a seguito di una precedente richiesta, e pertanto esamina l'eventuale altro iFC in ordine di priorità: nel nostro caso, non essendo impostati altri iFC, il S-CSCF inoltra quindi il messaggio all'utente richiesto tramite il P-CSCF (5).

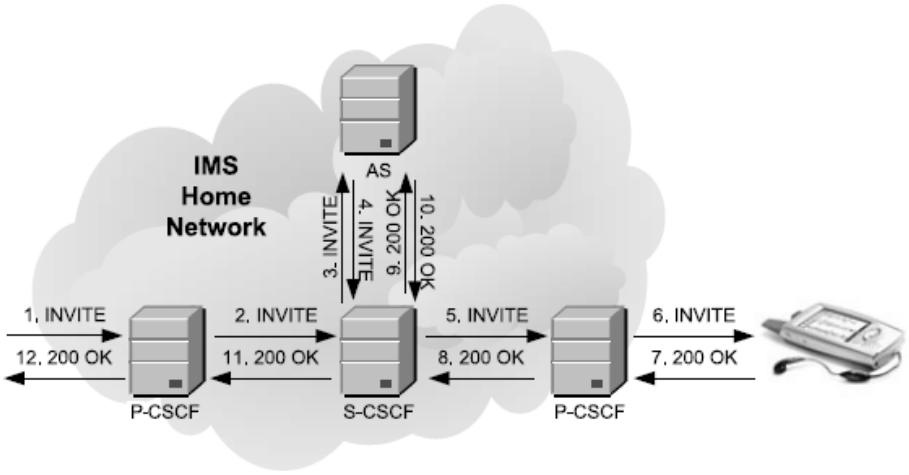


Figura 4.12: Segnalazione SIP relativa all'operazione di redirect

5. Il P-CSCF inoltra il messaggio di `SIP INVITE` all'utente `sip:bob@open-ims.test` (6).

6. Il messaggio `SIP OK` proveniente dall'utente chiamato attraverserà ora i nodi della rete secondo l'ordine inverso rispetto alla richiesta (7-8-9-10-11-12).

Nota: come già accennato, i messaggi di risposta non vengono influenzati dall'impostazione `setRecordRoute(true)`: essi, come comportamento di default, seguono sempre il path indicato all'interno del campo `Via` del messaggio di richiesta ricevuto.

La Figura 4.12 presenta lo schema della segnalazione SIP illustrata (il P-CSCF è unico nonostante venga raffigurato due volte). Le successive figure contengono invece il dettaglio degli header di due messaggi SIP inviati in questo processo: la Figura 4.13 mette in evidenza il contenuto del campo `Route` del messaggio inviato dal S-CSCF all'AS (3), composto dall'URI di quest'ultimo e dall'ODI. La Figura 4.14 mostra invece i campi `Record-Route` e `Via` presenti nel messaggio consegnato all'UA Bob (6).

Durante la chiusura del dialogo, innescata dall'invio del messaggio `SIP BYE`, l'AS rimarrà nel percorso della segnalazione, proprio perché l'UA che effettua tale operazione utilizzerà il campo `Record-Route` (in cui è presente l'URI dell'AS) per la costruzione della voce `Route` del messaggio di SIP BYE.

```
Request-Line: INVITE sip:pippo@open-ims.test SIP/2.0
Message Header
  Record-Route: <sip:mo@scscf.open-ims.test:6060;lr>
  Route: <sip:127.0.0.1:9060;lr>, <sip:iscmark@scscf.open-ims.test:6060;lr
  Record-Route: <sip:mo@pcscf.open-ims.test:4060;lr>
```

Figura 4.13: Header del messaggio SIP inviato dal S-CSCF all'AS (3)

```
Request-Line: INVITE sip:bob@127.0.0.1:5064 SIP/2.0
Message Header
  Record-Route: <sip:mt@pcscf.open-ims.test:4060;lr>
  Record-Route: <sip:mt@scscf.open-ims.test:6060;lr>
  Record-Route: <sip:127.0.0.1:9060;fid=server_1;lr>
  Record-Route: <sip:mo@scscf.open-ims.test:6060;lr>
  Record-Route: <sip:mo@pcscf.open-ims.test:4060;lr>
  P-Asserted-Identity: <sip:alice@open-ims.test>
  Via: SIP/2.0/UDP 127.0.0.1:4060;branch=z9hG4bKc5a5.bc5864c1.0
  Via: SIP/2.0/UDP 127.0.0.1:6060;branch=z9hG4bKc5a5.cb3ef407.0
  Via: SIP/2.0/UDP 127.0.0.1:9060;branch=z9hG4bKdaac099199e4566
  Via: SIP/2.0/UDP 127.0.0.1:6060;branch=z9hG4bKc5a5.bb3ef407.0
  Via: SIP/2.0/UDP 127.0.0.1:4060;branch=z9hG4bKc5a5.ac5864c1.0
  Via: SIP/2.0/UDP 127.0.0.1:5062;rport=5062;branch=z9hG4bKaaa7
```

Figura 4.14: Header del messaggio SIP inviato dal P-CSCF all'UA (6)

Capitolo 5

Orchestrazione di servizi

Indice

Nella trattazione precedente c'è già stato modo di sottolineare come IMS costituisca una soluzione interessante per l'erogazione di servizi, soprattutto dal punto di vista degli operatori telefonici: in particolare, riveste grande importanza la possibilità di combinare più servizi di base e generalmente autonomi in un cosiddetto '**servizio composito**', che offra all'utente maggiori funzionalità e che permetta allo stesso tempo il riutilizzo del codice delle applicazioni di base. Prima di considerare l'interazione fra servizi in ambito specificatamente IMS, è bene introdurre alcuni concetti relativi alla composizione di servizi in generale, ed in particolare all'**approccio ad eventi**, cioè quello stile architetturale che consente l'invocazione di applicazioni a seguito del verificarsi di eventi, tipico dei servizi *telco*: ad esso faranno riferimento molti dei modelli proposti nel seguito.

Tuttavia, anche al di fuori di IMS, non esiste ad oggi un sistema universalmente riconosciuto efficace per l'orchestrazione di servizi, specialmente se questi sono realizzati con tecnologie diverse; un tentativo in questo senso è stato realizzato da Ericsson, che propone una piattaforma per il supporto alla composizione di servizi eterogenei: una descrizione dettagliata del suo funzionamento è presente nel Paragrafo 5.4.

5.1 Architettura *Event-Driven*

Con i termini *Event-Driven Architecture (EDA)*, si intende identificare un paradigma architetturale all'interno del quale diverse componenti software *reagiscono* a seguito di eventi, ovvero cambiamenti significativi dello stato di qualche entità. Un sistema basato su EDA consiste generalmente in una serie di entità che **generano eventi** (*agent*) e da altre che **consumano eventi** (*sink*): alle ultime spetta la responsabilità di produrre una *reazione* non appena l'evento viene loro presentato da parte delle prime. Le azioni intraprese possono essere eseguite per intero dalla stessa entità consumatrice di eventi, oppure questa può effettuare delle trasformazioni sull'evento e inoltrarlo verso un'altra entità. L'utilizzo di una architettura di questo tipo offre i seguenti vantaggi:

- più facile sviluppo e manutenzione di applicazioni/servizi distribuiti, anche caratterizzati dalla presenza di eventi asincroni e non prevedibili;
- facile assemblaggio, riassemblaggio e configurazione di servizi;
- facile aggiunta di nuovi servizi;
- permette il riutilizzo dei componenti di base, consentendo quindi la creazione di un ambiente più dinamico.

L'architettura *event-driven* si può considerare complementare a quella *Service Oriented (SOA)* [44], poiché i servizi possono essere attivati tramite operazioni svolte a seguito del verificarsi di un determinato evento. L'unione di questi due paradigmi ha portato alla creazione dell'architettura chiamata **Event-Driven SOA**, che offre modularità (caratteristica di SOA) e il supporto all'interazione fra servizi indipendenti (caratteristica di EDA).

I servizi possono essere invocati quando uno specifico evento raggiunge un consumatore (*sink*): quest'ultimo può far uso di dati presenti all'interno dell'evento stesso per richiamare il servizio desiderato; in alternativa un servizio può comportarsi da generatore di eventi (*agent*) e creare un nuovo evento basandosi sul risultato di un'altra invocazione. Le Figure 5.1 e 5.2 corrispondono rispettivamente alla prima e alla seconda modalità di selezione di servizi.

Questo modello di esecuzione è tipico dei servizi telefonici, che spesso richiedono anche la presenza di elementi contenenti la descrizione di uno stato (*stateful*), e si discosta da quello realizzato con linguaggi per la descrizione di *workflow*, come ad esempio WS-BPEL [45] (specifiche tecniche in [46]), basato piuttosto sulla logica *request/response*.

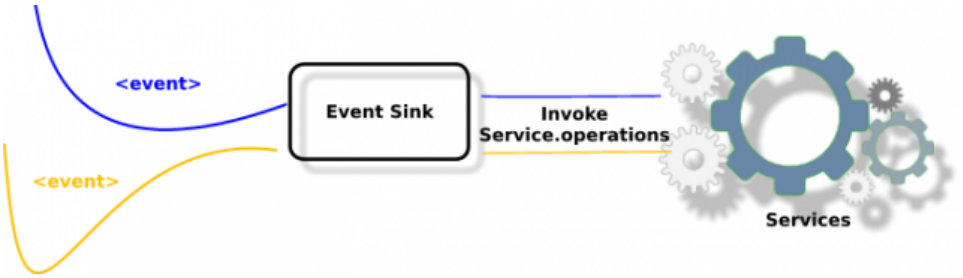


Figura 5.1: Invocazione di servizi nell'architettura Event-Driven SOA

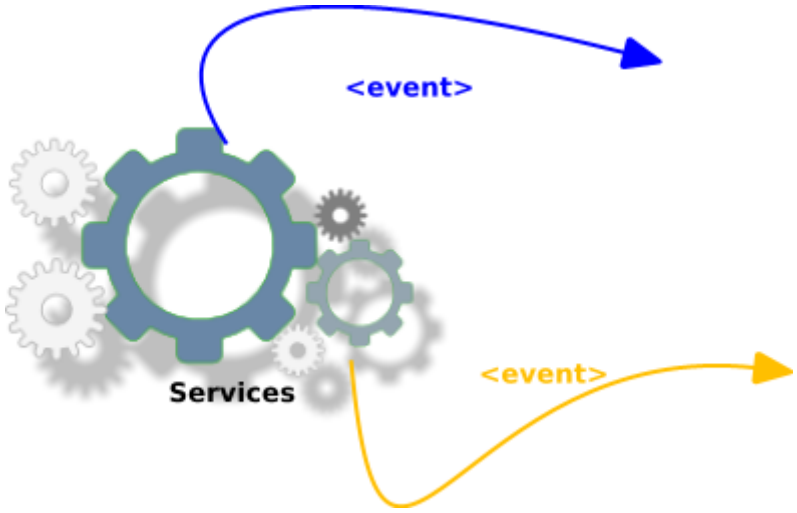


Figura 5.2: Invocazione di servizi nell'architettura Event-Driven SOA: Servizio come *generatore di eventi*

5.2 Web Service e Servizi Telefonici

Nel dominio delle applicazioni Web, l'Architettura Orientata ai Servizi (SOA - Service Oriented Architecture) promuove la composizione di web service al fine di offrire funzionalità più complesse, utilizzando standard come WS-BPEL. Tale paradigma incoraggia la modularità del software, nonché il suo riutilizzo e l'indipendenza tra piattaforme e linguaggi. Allo stesso tempo, nel dominio delle telecomunicazioni, standard come IMS e SIP Servlet permettono una analoga composizione di applicazioni SIP di base in servizi complessi, per ottenere i medesimi vantaggi. Ma soprattutto, c'è interesse ad integrare questi due domini, al fine di erogare servizi telefonici più ricchi, che facciano uso della grande quantità di dati accessibile mediante le API esposte dai Web service.

Tale intento si scontra con varie problematiche, in primis quella di gestire adeguatamente l'interazione tra i servizi stessi, data anche la sostanziale differenza tra i servizi Web e quelli telefonici; tali differenze sono state esposte ed analizzate in [50].

5.3 SIP Servlet 1.1: l'Application Router

Lo standard SIP Servlet è sicuramente uno dei più usati quando si ha a che fare con servizi basati sul protocollo SIP: con la versione 1.1 dello standard è stato introdotto un nuovo componente, chiamato Application Router, responsabile della selezione di applicazioni SIP in un determinato server. Di seguito viene descritto il suo funzionamento, in quanto la tecnologia SIP Servlet è utilizzata in Ericsson Composition Engine, la piattaforma per l'orchestrazione di servizi descritta nel paragrafo successivo.

L'Application Router viene invocato dal container al fine di selezionare la corretta SIP Servlet a seguito della ricezione di una richiesta iniziale (ovvero una richiesta per la quale il container non ha una conoscenza pregressa). Il routing di un tale messaggio stabilisce dunque l'ordine di esecuzione delle applicazioni, secondo le indicazioni che l'Application Router fornisce al proprio container: in particolare, se l'applicazione appena invocata restituisce una richiesta iniziale per continuare la chiamata, quest'ultima verrà nuovamente inviata dal container all'Application Router, il quale ancora una volta stabilirà la successiva applicazione da invocare. In questo modo si viene a creare una *catena* di invocazioni, che va a costituire il servizio completo. Ogni richiesta successiva (*subsequent request*) appartenente al dialogo SIP in corso sarà poi instradata lungo il tragitto delle applicazioni selezionate (eventualmente in ordine inverso rispetto alla richiesta originaria), **ma l'Application Router non verrà più contattato**.

La Figura 5.3 rappresenta questo tipo di interazioni: una richiesta iniziale (*incoming request*) raggiunge il Servlet Container, e quest'ultimo decide

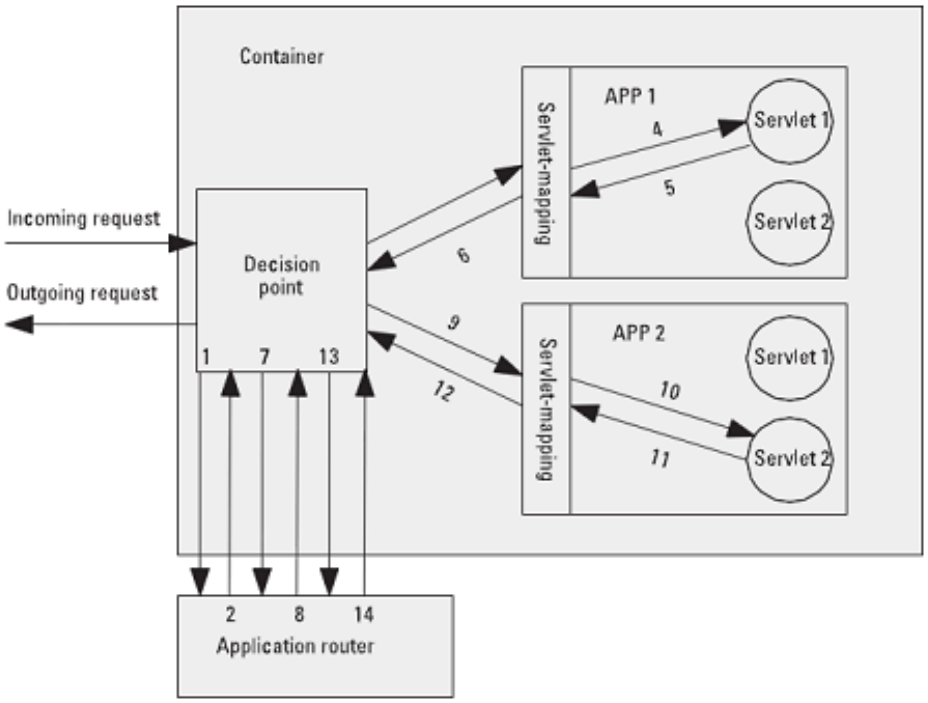


Figura 5.3: Selezione delle applicazioni mediante Application Router

dunque di contattare l'Application Router per poter stabilire verso quale applicazione instradare il messaggio (1). L'Application Router può far uso di **qualsiasi informazione** per decidere quale dovrà essere l'applicazione successiva da contattare: tale risposta (2) viene dunque passata al container, che provvederà ad inoltrarla verso la servlet corretta (3,4). Una volta compiute le elaborazioni, l'applicazione restituisce il messaggio (eventualmente modificato) al container e, se si tratta ancora di una richiesta iniziale, quest'ultimo contatterà nuovamente l'Application Router: tale processo si svolge fino a che non vi saranno più applicazioni da inserire nella composizione, e pertanto l'Application Router informerà il container di tale situazione (14) e la richiesta verrà inoltrata all'esterno (*outgoing request* in figura) secondo le consuete regole del protocollo SIP.

Bisogna specificare che ogni applicazione invocata opera come un'entità SIP indipendente dalle altre: potrà infatti inviare richieste e risposte, sebbene nella realtà gli scambi di messaggi avverranno solo con le applicazioni adiacenti nella catena di invocazioni. Inoltre, a differenza di quanto avviene per un orchestratore di Web Service, la responsabilità dell'Application Router consiste nel comporre una sequenza lineare di applicazioni per un determinato sottoscrittore, che opera come chiamante o come chiamato:

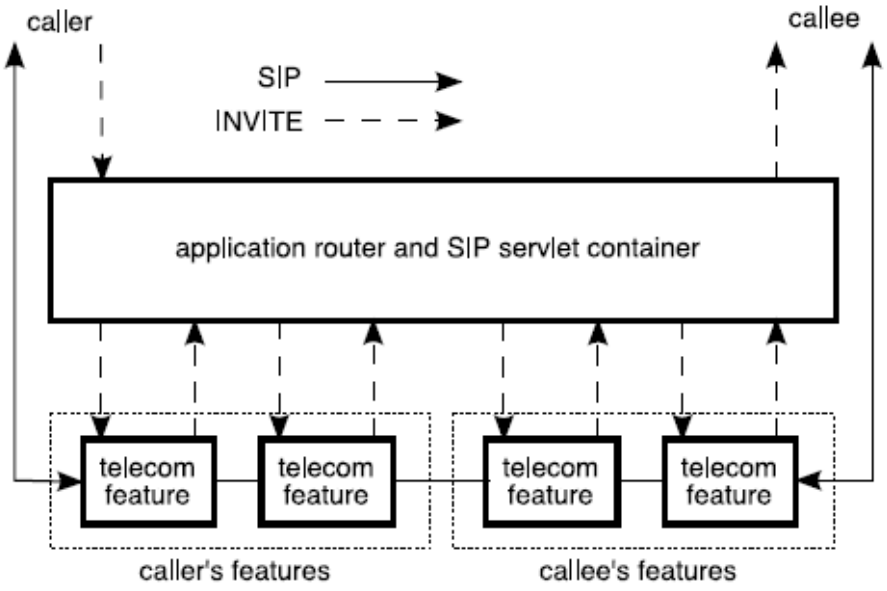


Figura 5.4: Composizione di applicazioni in un SIP Servlet Container

pertanto, i servizi associati a colui che effettua la chiamata verranno composti *prima* di quelli del ricevente. La Figura 5.4 evidenzia questo fatto.

Un Application Router si basa su quattro parametri per poter decidere quale applicazione selezionare:

1. L'**URI del sottoscrittore** (chiamante o chiamato) presente nel messaggio di richiesta iniziale;

2. La **routing region**, che può essere *originating* o *terminating*, e permette all'AR di distinguere il ruolo dell'utente (chiamante o chiamato) dal momento che a seconda di esso potrebbe utilizzare servizi differenti;

3. La **routing directive**, che può essere `NEW`, `CONTINUE` o `REVERSE`, e permette all'AR di capire se la richiesta ricevuta è appartenente ad una catena di invocazioni già in corso (`CONTINUE`) oppure se si tratta di una nuova (`NEW`); nel primo caso l'AR fa uso della *state information* associata alla richiesta ricevuta per riprendere il processo di selezione. Mediante questo meccanismo l'Application Router rimane *stateless*;

4. La **state information**, contenente le applicazioni selezionate fino a quel momento per conto della richiesta alla quale è associata.

Infine, un AR può far uso di qualsiasi altra informazione durante le sue scelte, come ad esempio la data e ora, lo stato della rete e così via.

Va comunque precisato che le specifiche JSR 289 non definiscono dettagli implementativi di un AR: è compito del produttore del SIP Servlet Container realizzare un AR che implementi *almeno* i metodi descritti all'interno dell'Appendice C delle specifiche Servlet 1.1 (il cosiddetto *Default Application Router (DAR)*). Ovviamente, resta possibile fornire una realizzazione più ricca di quella descritta, che possa ad esempio recuperare informazioni da fonti differenti e svolgere compiti di orchestrazione più complessi.

5.4 Ericsson Composition Engine (ECE)

Attualmente, tra le proposte più valide all'orchestrazione di servizi si colloca l'Ericsson Composition Engine (ECE) [13], un approccio alla composizione di servizi che permette di creare applicazioni cosiddette **convergenti**, ovvero che possono contenere componenti appartenenti a domini differenti, e delle quali si può usufruire attraverso reti e tecnologie d'accesso diverse (ad esempio reti wireless, cellulari, ecc.).

Per poter raggiungere tale scopo, il nucleo della piattaforma è **technology agnostic**, ovvero non si basa su di una singola tecnologia o sistema, ma piuttosto è focalizzato sulle funzionalità: in questo modo il supporto per nuove tecnologie può essere aggiunto in maniera semplice. Ovviamente, tale piattaforma consente l'utilizzo di componenti di base costituiti da Web Service così come applicazioni SIP, e ne permette una loro composizione in accordo con i principi SOA. Il paragrafo seguente analizzerà nel dettaglio le caratteristiche di Ericsson Composition Engine.

5.4.1 Struttura della piattaforma

Ericsson Composition Engine (ECE) è un sistema per la composizione di applicazioni convergenti che fa uso di servizi provenienti dai domini Web, telecomunicazioni ed enterprise, basati sia su commutazione di circuito (CAP/INAP) che sul protocollo IP (SIP), in grado di controllare le interazioni tra le varie componenti superando le barriere imposte dalle diverse tecnologie. Esso si identifica con un orchestratore centrale che fa da mediatore tra i differenti servizi: tale idea è rappresentata nella Figura 5.5.

ECE si basa sui principi SOA, pertanto tutti i servizi sono considerati come unità autonome e debolmente accoppiate (*loosely coupled*). Da un punto di vista di IMS, esso ricopre il ruolo di un Application Server SIP: in particolare, il Composition Engine informa il core IMS della necessità di invocare un determinato servizio SIP, operazione consentita dalle specifiche dell'Application Router presenti nel documento JSR 289 relativo a SIP Servlet 1.1. Qualora fosse necessario invocare un servizio *non-SIP*, sarà invece

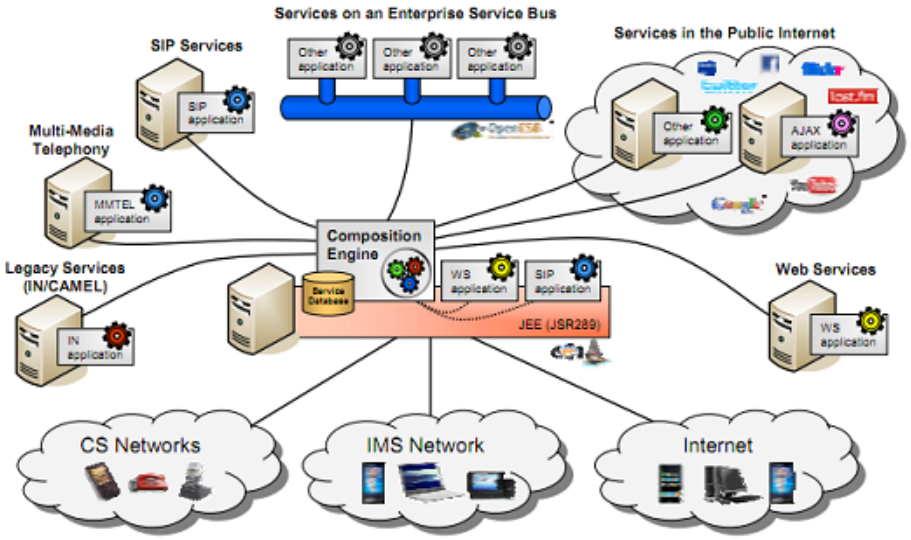


Figura 5.5: Ericsson Composition Engine come mediatore tra le diverse tecnologie

il Composition Engine stesso ad effettuale tale operazione, utilizzando eventuali risultati ricevuti dall'applicazione come parametri per le invocazioni successive. Questo tipo di interazioni, rappresentato in Figura 5.6, costituisce un esempio tipico di quanto avviene per una applicazione convergenti: le linee tratteggiate in figura corrispondono alle invocazioni di servizi, in particolare quelle dirette verso gli ingranaggi blu sono relative a applicazioni SIP, mentre le altre ad applicazioni *non-SIP*.

In questo modo, il Composition Engine realizza un singolo orchestratore, che può effettuare sia invocazioni di servizi SIP basandosi sui meccanismi di un Application Router, che selezione di WS mediante il consueto paradigma *request-response*. Una applicazione composta da più servizi viene realizzata tramite la definizione di un **application skeleton**, ovvero un modello della struttura d'esecuzione che specifica i servizi costituenti. Tuttavia, i dettagli relativi ai protocolli usati ed alle interazioni tra i componenti sono delegati ai cosiddetti **Composition Execution Agents (CEA)**, i quali devono assicurare l'esecuzione delle decisioni relative alla composizione, nei modi stabiliti dalla particolare tecnologia cui fanno riferimento (ogni CEA corrisponde infatti ad una tecnologia). Infine, è definito uno **shared state**, ovvero un insieme di informazioni utilizzate dall'*application skeleton* e dai diversi CEA per un'esecuzione coordinata dei servizi.

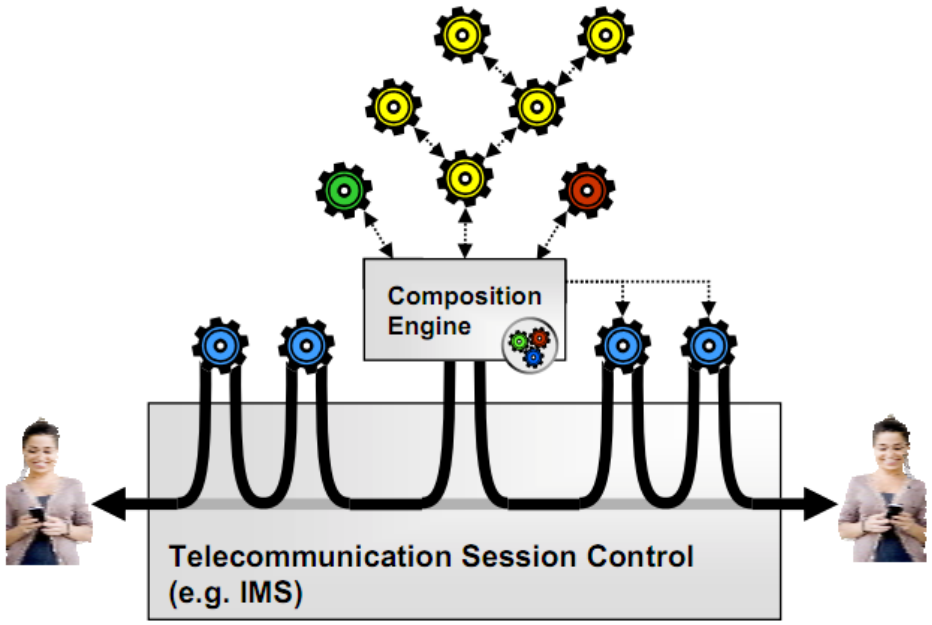


Figura 5.6: Orchestrazione di una applicazione converged nel contesto IMS

5.4.2 Descrizione dei servizi ed *Application Skeleton*

Le modalità con le quali vengono descritti ed identificati i vari servizi è una delle caratteristiche essenziali di ECE, che permette tra l'altro di ottenere la desiderata caratteristica di *loose coupling* tra l'applicazione composita e i suoi componenti. Difatti, i servizi possono essere descritti specificando una collezione di proprietà astratte delle quali godono, che riflettono le caratteristiche e funzionalità offerte dal servizio stesso. Ad esempio, un servizio che permette di effettuare l'accesso ad un calendario online di un utente avrà una proprietà astratta chiamata `type` con il valore `calendar_login`. Pertanto, un'applicazione composita conterrà un insieme di proprietà astratte corrispondenti alle funzionalità da lei richieste: sarà il Composition Engine, in fase run-time, a selezionare gli specifici servizi che disporranno delle caratteristiche richieste. In questo modo il progettista dell'applicazione dovrà semplicemente specificare di quali funzioni avrà bisogno, delegando all'ECE la scelta effettiva dei servizi che le realizzeranno, siano essi basati su SIP piuttosto che su WS o altre tecnologie. La decisione relativa alla tecnologia da utilizzare non rientra quindi più nei compiti di chi progetta una applicazione composita.

Queste informazioni andranno quindi a costituire i cosiddetti *application skeleton*, che oltre alla descrizione dei servizi da utilizzare conterranno anche informazioni relative al loro ordine di esecuzione. Per poter raggiungere tale

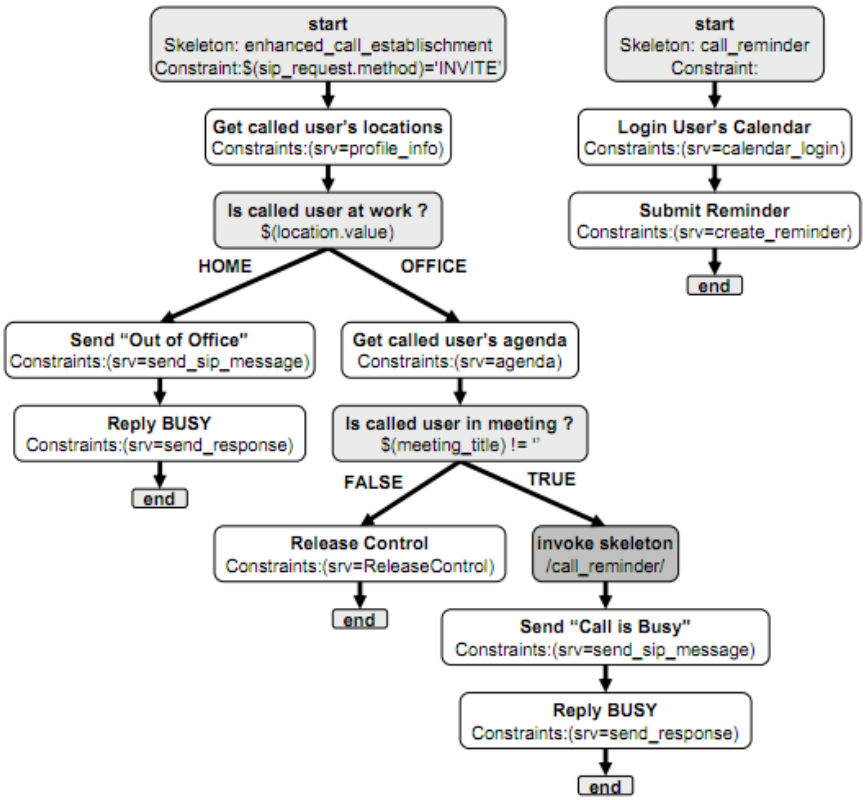


Figura 5.7: Un esempio di Application Skeleton per un servizio composito

obiettivo, si utilizzano elementi grafici che rappresentano i servizi costituenti nonché le loro interazioni e gli eventuali costrutti condizionali (un esempio è rappresentato in Figura 5.7).

E' inoltre possibile per un *application skeleton* invocare altri *sub-skeleton*, al raggiungimento di particolari condizioni all'interno del flusso d'esecuzione. Caratteristica importante del Composition Engine è poi la capacità di selezionare in fase di esecuzione le applicazioni corrispondenti a quanto specificato nell'*application skeleton* secondo politiche di load balancing o di costo; questa possibilità è permessa proprio dal particolare tipo di descrizione adottato per i servizi. Ad esempio, nella figura precedente è raffigurato il riquadro 'Get called user's location', avente come vincolo (*constraint*) il valore `srv=profile_info`: il Composition Engine, in fase di esecuzione, cercherà un servizio che conterrà nella sua descrizione la proprietà `srv=profile_info` e, qualora ve ne sia più di uno, selezionerà quello che meglio risponde alle esigenze di costo e carico impostate.

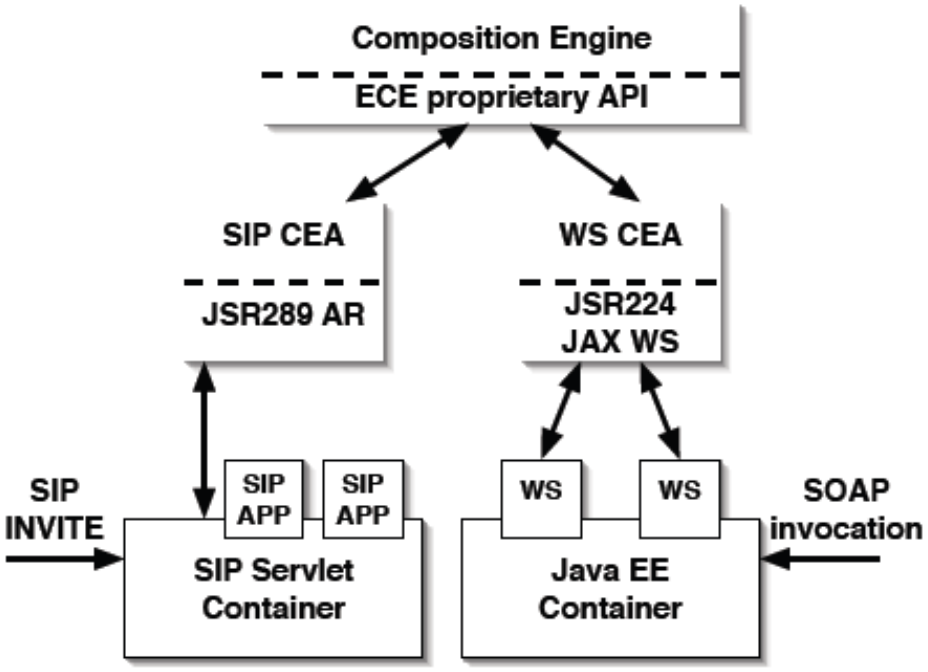


Figura 5.8: Distinzione tra le API del Composition Engine e quelle proprie dei CEA

5.4.3 L'esecuzione dei servizi

In ECE il processo di selezione dei servizi è dunque indipendente dalla tecnologia che costituisce le singole applicazioni: il Composition Engine seleziona dapprima un servizio senza curarsi dei dettagli relativi alla sua esecuzione, che avverrà successivamente in base alle regole ed ai protocolli specifici del servizio selezionato. Sono i Composition Execution Agent (CEA) gli addetti all'esecuzione vera e propria dei servizi che il Composition Engine ha selezionato: inoltre, essi si occupano di notificare ad esso eventuali richieste di servizi compositi provenienti dall'esterno.

E' evidente che, in un contesto di in cui si vogliano eseguire servizi compositi creati mediante l'utilizzo di tecnologie differenti, saranno necessari più CEA, ognuno dedicato ad una particolare implementazione. Il Composition Engine espone a tutti i CEA delle API generiche, indipendenti cioè dalla particolare tecnologia che identifica ogni CEA; un insieme di API complementari sono poi esposte dai singoli CEA, e sono adoperate dal Composition Engine (si veda Figura 5.8). In questo modo una nuova tecnologia di servizi può essere integrata senza grandi difficoltà, ovvero creando un nuovo CEA basato su tali API e senza modificare il nucleo dell'orchestratore.

Per quanto riguarda l'utilizzo al di sopra della rete IMS, il Compositon Engine non è altro che un Application Server dotato di un SIP CEA che supporta l'interfaccia ISC. In particolare, gli *initial Filter Criteria* impostati nel core di IMS causano l'inoltro dei messaggi SIP al SIP CEA, il quale effettua la richiesta di composizione direttamente al Composition Engine. Il SIP CEA opera dunque come un Application Router che utilizza il Composition Engine come **fonte di informazioni** riguardo a quali applicazioni invocare e in che ordine. Inoltre, nell'*application skeleton* potrebbero essere presenti servizi non-SIP: la loro esecuzione avverrà in seguito ad una richiesta della Composition Engine rivolta al CEA che si occupa della particolare tecnologia di servizio *non-SIP*.

Bisogna tuttavia specificare che le funzionalità richieste dal SIP CEA sono in realtà una **estensione delle specifiche di un Application Router**; le seguenti caratteristiche sono quelle aggiuntive rispetto a quanto descritto nelle specifiche JSR 289:

- Composizione di applicazioni SIP che prenda in esame non solo informazioni specifiche del protocollo SIP, ma anche quelle provenienti da moduli differenti. Ad esempio, l'indirizzo di inoltro di un messaggio SIP può essere recuperato tramite una rubrica di contatti online invocata come WS;

- Essere invocato non solo da *initial request*, bensì anche dai *subsequent message* che vengono scambiati dopo l'instaurazione della sessione. Tali messaggi potrebbero non influenzare direttamente la segnalazione SIP in corso tra gli utenti, ma innescare eventualmente delle attività appartenenti a domini non-SIP. Pertanto, la composizione potrebbe proseguire anche dopo l'instaurazione della sessione SIP;

- Gestire le sessioni di composizione, che possono essere composte tanto da sessioni SIP quanto da sessioni relative ai WS;

- Permettere lo scambio di messaggi tra i differenti servizi interessati nella composizione, quindi mediante la condivisione di dati tra tecnologie di servizi diverse.

Per quanto riguarda l'ultimo punto, le specifiche di ECE prevedono appunto la presenza di uno *shared state*, ovvero un insieme di tutti i dati raccolti dai vari CEA coinvolti nella sessione di composizione (si veda Figura 5.9). Pertanto, la Composition Engine sarà a conoscenza ad esempio delle informazioni contenute nei parametri dei messaggi SIP, e potrà utilizzarli come input per i servizi che ne fanno richiesta. Ciò che è più rilevante in merito allo shared state è quindi il fatto di offrire una modalità per l'interazione tra i vari servizi che sia slegata da una particolare tecnologia, e pertanto il più possibile generica.

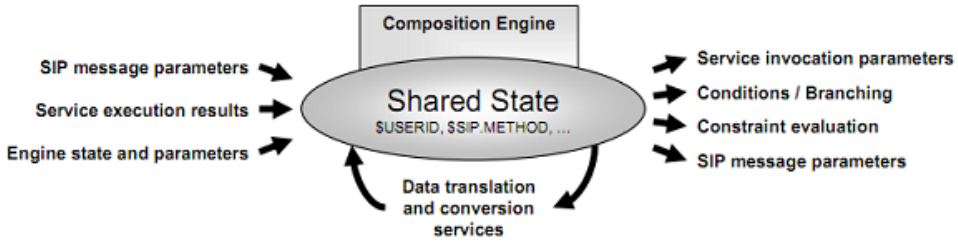


Figura 5.9: *Shared State* di una sessione di composizione

La necessità di un meccanismo che permetta l'interazione tra i vari servizi si è reso necessario proprio nel contesto delle applicazioni SIP in quanto **una loro realizzazione mediante Servlet non offre questo tipo di funzionalità**: difatti, sebbene un singolo messaggio possa innescare l'invocazione di più servlet, queste saranno viste dal container come entità distinte, e quindi prive della possibilità di cooperare. Un tentativo di risolvere questa problematica era stato proposto già in [51], dove gli autori descrivono un'architettura per la composizione di servizi chiamata **Alcatel-Lucent Service Broker**: nel documento, con il termine ***steplet*** si indica la versione delle servlet che permette l'utilizzo delle funzionalità aggiuntive sopra esposte, e realizzata in un Servlet Container proprietario.

Capitolo 6

Modelli per l'orchestrazione in ambito IMS

Indice

In questo capitolo verranno esaminati vari modelli architetturali candidati all'orchestrazione di servizi in ambito specificatamente IMS: in particolare, ci baseremo sul paradigma event-driven presentato nel capitolo 5 per proporre delle soluzioni che consentano un interfacciamento efficiente tra lo strato applicativo in cui risiedono i servizi (e i moduli per la loro orchestrazione) e il core della rete IMS.

L'orchestratore non è infatti in grado di comunicare direttamente con i servizi da invocare, proprio perché questi sono stati sviluppati come unità indipendenti, e non presentano nativamente delle funzionalità che permettano

tale interazione: a questo proposito esamineremo i problemi che l'interazione fra essi può causare, e ci baseremo su di essi per proporre una soluzione il più possibile efficiente. Per questo si presentano due sfide complementari:

- la gestione delle **interazioni negative**, cioè quelle che causano conflitti tra i vari servizi: nel seguito ci riferiremo ad essa con i termini **gestione dei conflitti**;
- la gestione delle **interazioni positive**, cioè quelle che permettono la composizione dei servizi di base: in seguito ci riferiremo ad essa con i termini **gestione della composizione**.

Nel seguito del capitolo l'interesse sarà dunque rivolto verso una tipologia di componenti che svolga le operazioni di 'mediazione' tra l'entità orchestratrice ed i servizi veri e propri: nel caso di Ericsson Composition Engine sono i CEA, e come abbiamo visto ne esiste uno per ogni tipo di tecnologia supportata. Essendo lo standard IMS basato principalmente sul protocollo SIP, è naturale che le soluzioni proposte si basino su componenti che appartengono a tale contesto: l'obiettivo sarà quello di valutare l'approccio migliore, sia in termini di efficienza che di semplicità di utilizzo ed aderenza agli standard. In particolare, possiamo suddividere le soluzioni in due categorie:

- soluzioni relative al *call-processing*, ovvero alla **composizione di *feature* telefoniche**;
- soluzioni relative alla **composizione di servizi generici** (di seguito chiamati servizi IT).

La prima tipologia riguarda la creazione di servizi compositi nei quali l'instaurazione di una chiamata telefonica rappresenta l'aspetto centrale: i servizi saranno basati per la maggior parte sul protocollo SIP, e si tratterà generalmente di funzionalità fornite a seguito della ricezione di un messaggio `SIP INVITE`. Nel secondo caso, invece, l'orchestrazione riguarda servizi di qualsiasi genere, indicati con il termine generico di Mashup: l'evento che avvia il processo di composizione non è necessariamente collegato ad una chiamata, nonostante *feature* telefoniche possano essere inserite nella fase di creazione del servizio composito. I due approcci faranno uso di:

- **SIP Servlet 1.1**;
- **librerie** che implementano lo stack SIP (JAIN SIP);
- **orchestratore** realizzato secondo specifiche proprietarie.

Tali tecnologie verranno utilizzate per proporre un componente che consenta la comunicazione tra il core di IMS e la piattaforma per la composizione di servizi.

6.1 La gestione dei conflitti in IMS

Le reti basate su IP (e quindi IMS) devono affrontare il problema del conflitto fra servizi, presente anche nelle reti telefoniche tradizionali e generalmente indicato con ***'feature interaction problem'***. Esso si presenta quando servizi (ovvero *feature*) differenti invocati durante una sessione si comportano in maniera corretta se sono eseguiti indipendentemente l'uno dall'altro, ma causano invece problemi quando la loro esecuzione non avviene più in modo separato.

Un esempio di interazione indesiderata tra *feature* è costituita da una chiamata tra due utenti Alice e Bob che hanno attivi rispettivamente un servizio di *Selective Call Rejection* (o *Call Screen*, cioè il rifiuto di chiamate provenienti o dirette a dei numeri presenti in una blacklist) e di *Call Forwarding Incondizionato*. Se Alice effettua una chiamata verso Bob e quest'ultimo ne effettua la redirezione verso un terzo utente Eva presente nella blacklist di Alice, si presenterà una situazione di errore: il corretto funzionamento del servizio di *Selective Call Rejection* prevede infatti che la chiamata non sia inoltrata ad Eva, ma questo comportamento va a scontrarsi con quello del *Call Forwarding Incondizionato* di Bob; viceversa, il corretto funzionamento di quest'ultimo servizio causerebbe l'inoltro della chiamata ad un numero presente nella blacklist di Alice, circostanza ancora una volta indesiderata.

Sebbene l'esempio sia molto semplice, esso mette in luce la necessità di adottare dei meccanismi che permettano la **prevenzione** di tali situazioni patologiche o, qualora non sia possibile prevenirle, una loro **individuazione e risoluzione**. In letteratura sono presenti due approcci a questo tipo di problema: il primo, quello *offline*, prevede il riconoscimento e la risoluzione di possibili conflitti *prima* che essi avvengano; ma poiché spesso i servizi hanno un comportamento imprevedibile durante la fase di run-time, esiste anche l'approccio *online*, che deve consentire l'eliminazione dei conflitti verificatisi *durante* l'esecuzione stessa dei servizi. Bisogna comunque considerare che la stessa risoluzione dei conflitti può avvenire in maniera *statica*, ovvero mediante delle linee guida già delineate e predefinite per ogni servizio, oppure in modo *dinamico*, basandosi cioè sulle informazioni ottenute in tempo reale.

6.1.1 Tecniche per la prevenzione dei conflitti

Varie tecniche sono state proposte per la *prevenzione* dei conflitti tra servizi; le più rilevanti sono:

- **Service Negotiation** [52]: basato su servizi SIP, gli utenti finali negoziano i servizi disponibili *prima* dell'instaurazione della sessione. Per questo il chiamante aggiunge alla richiesta di `SIP INVITE` una lista contenente i suoi servizi nonché una serie di azioni che il ricevente può

effettuare qualora non ne accettasse. In alternativa, proprio quest'ultimo può inviare una lista modificata al chiamante, che dovrà dunque scegliere quali opzioni preferire. Il processo si conclude quanto entrambe le parti hanno raggiunto un accordo.

Il principale difetto di questo metodo è rappresentato dalla mancanza di trasparenza nei confronti degli utilizzatori finali dei servizi.

- **Service Description** [53]: il metodo si basa sulla presenza di alcune tabelle, legate ai terminali, che contengono una descrizione di vari servizi; mediante l'uso di diagrammi di transizione è possibile determinare i potenziali conflitti tra servizi, e intraprendere quindi delle misure cautelative.

  Tale metodo è tuttavia poco flessibile, dal momento che le tabelle devono essere costantemente aggiornate per far fronte all'introduzione di nuovi servizi.

- **Formal Models** [54]: un modello formale che descrive i servizi SIP e i loro conflitti è realizzato mediante uno *Specification and Design Language (SDL)* che interessa alcune tra le più importanti caratteristiche di un messaggio SIP, quali il caller ID, gli header di indirizzamento e il sequence number. Sulla base della classificazione dei vari tipi di servizi e delle corrispondenti strategie di prevenzione dei conflitti, il numero di effettive interazioni negative dovrebbe diminuire.

  Tuttavia, questo sistema è limitato dalla quantità di possibili conflitti nonché dai diversi comportamenti dei servizi.

6.1.2 Tecniche per l'individuazione e la prevenzione dei conflitti

Come per la prevenzione, esistono varie tecniche finalizzate all'individuazione e risoluzione dei conflitti tra servizi. In particolare:

- **Feature Interaction Manager** [55]: gli utenti informano un Feature Interaction Manager (FIM) della loro volontà di eseguire un servizio: tale entità è situata tra gli stessi utenti e gli eventuali SIP proxy. Compito del FIM è determinare se l'esecuzione del nuovo servizio provocherà delle interazioni con gli altri appartenenti alla medesima sessione: nel caso venga individuato un conflitto, il FIM avverte i proxy di interrompere la chiamata e lancia la procedura di risoluzione.

- **Macchina a stati finiti** [56]: si tratta di un modello formale basato sulla presenza di una macchina a stati finiti che specifica i servizi e di una metodologia per individuare e risolvere i conflitti basandosi su una serie di possibilità. I conflitti sono definiti come stati della macchina

non desiderabili, e il metodo di risoluzione prevede che tali stati si comportino in maniera accettabile.

- **Distribuited Conflict Resolution Mechanism** [57]: in questa proposta è definito un componente che contiene una lista dei servizi e dei potenziali conflitti tra di essi; quando una applicazione identifica che più servizi possono essere richiesti simultaneamente, invia una richiesta a tale componente, il quale specificherà (sulla base di policy espresse mediante formule logiche) i servizi che potranno essere eseguiti.

6.1.3 Requisiti per la risoluzione dei conflitti in ambito IMS

Le proposte elencate in precedenza sono d'aiuto per la formulazione di soluzioni analoghe in ambito più specificatamente IMS; in particolare, un meccanismo di gestione dei conflitti in tale contesto dovrà sicuramente avere le seguenti caratteristiche:

- **Essere SIP-based**: poiché i conflitti tra servizi avvengono al momento della loro invocazione, ed essendo l'instaurazione e l'invocazione di servizi basata su SIP in IMS, qualsiasi soluzione deve basarsi su SIP;
- **Trasparenza agli utenti finali**: invece che richiedere l'intervento degli stessi utenti per la risoluzione dei conflitti, tale meccanismo dovrebbe essere presente nell'architettura base di IMS. Il vantaggio principale di questo approccio consiste in una maggiore flessibilità delle funzioni offerte senza dover coinvolgere l'utente finale;
- **Basso impatto sul tempo di instaurazione della sessione**: poiché gli interventi di risoluzione avvengono durante la creazione della sessione SIP, essi devono rispettare le caratteristiche di tempo reale richieste dal protocollo SIP.

Per poter soddisfare tali requisiti, è dunque richiesta la presenza di una entità che possa orchestrare le diverse invocazioni di servizi, nonché risolvere gli eventuali conflitti: per come è costituita l'architettura di IMS, un tale componente dovrebbe poter intercettare i messaggi provenienti dal S-CSCF e diretti agli Application Server. Uno sforzo in questo senso è stato svolto dallo stesso ente 3GPP, che in [58] ha definito le specifiche (per il momento ad un livello di astrazione piuttosto elevato) di un possibile componente standard chiamato **Service Broker**, le cui funzioni possono essere suddivise in due categorie: *online* e *offline*. Le funzioni offline sono rappresentate da:

1. Identificazione di tutti i servizi richiesti da un utente (associati al profilo)
2. Comprensione delle modalità di interazione tra i servizi per risolvere eventuali conflitti

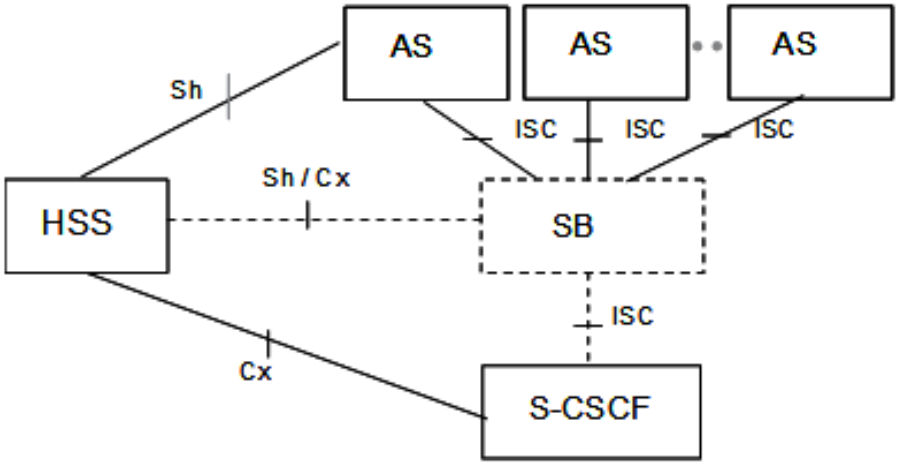


Figura 6.1: Collocazione del Service Broker nell'architettura IMS

3. Definizione del comportamento della composizione di servizi.

Nelle funzioni online rientra invece un meccanismo volto ad assicurarsi che, durante una sessione, i servizi richiesti dall'utente si compongano nella maniera corretta, e che pertanto il servizio composito erogato corrisponda a quanto atteso dall'utilizzatore. Da un punto di vista architetturale, tale componente si troverebbe dunque a metà tra gli Application Server e il S-CSCF: in effetti, 3GPP specifica che una eventuale implementazione di tale componente potrebbe avvenire sia come aggiunta di funzionalità al S-CSCF, che come parte di un Application Server, o addirittura come componente a sé stante. La Figura 6.1 rappresenta come riquadro tratteggiato il Service Broker, e ne evidenzia la collocazione e le interazioni con le altre componenti di IMS.

Tuttavia, gli sforzi di 3GPP non hanno portato alla realizzazione di un componente universalmente accettato per la gestione della composizione di servizi, e si sono limitati a indicare una possibilità tra le tante disponibili: piuttosto, vari enti e gruppi di ricerca hanno proposto delle soluzioni ad hoc per affrontare tale problema, spesso relative ad una specifica tecnologia di servizi. Nel seguito, esporremo con maggiore dettaglio le soluzioni più rilevanti, evidenziando le potenzialità e le carenze e specificandone i confini di utilizzo.

6.2 Composizione di *feature* telefoniche

In questo paragrafo esamineremo delle architetture che permettono di effettuare la composizione di servizi in stretta collaborazione con l'infrastruttura IMS sottostante: in particolare, l'interesse è rivolto principalmente all'orchestrazione di *feature* telefoniche, quindi ad una esecuzione di servizi che pone come punto di partenza l'instaurazione di una chiamata tra due utenti, alla quale si vogliono associare delle funzionalità aggiuntive. Per questi motivi l'evento che provoca l'avvio della procedura di composizione è la ricezione, da parte della piattaforma di orchestrazione, di un messaggio di `SIP INVITE`.

6.2.1 Soluzione 1: Nessuna modifica dell'Application Router

La soluzione seguente, basata sullo standard SIP Servlet 1.1, prevede l'adozione di una struttura 'gerarchica' della composizione, e non richiede di modificare in maniera sensibile l'Application Router. A questo scopo, è necessario operare una distinzione tra:

1. le **Servlet SIP** che realizzano dei servizi di tipo telefonico, quindi ad esempio *Call-Blocking*, *Call-Forwarding* e così via;
2. le **Servlet non-SIP** che realizzano servizi di qualsiasi genere, ad esempio WS.

Esse sono viste dal Container allo stesso modo, e la differenza sostanziale risiede nel tipo di operazioni che eseguono una volta invocate: le Servlet di tipo (2) contengono infatti al loro interno l'invocazione del metodo `notifyEvent`, che utilizzano per richiedere l'esecuzione dei servizi *non-SIP* che rappresentano. E' per questo che s'è parlato di **struttura gerarchica**: ogni Servlet di questo tipo invoca infatti (indirettamente) altri servizi, a seconda di quanto specificato nella descrizione del Mashup.

Alla ricezione di un `notifyEvent` da parte di una Servlet di tipo (2), l'orchestratore si occupa autonomamente dell'invocazione dei servizi *non-SIP*, per poi restituire il controllo all'AR quando il processo di selezione (per quella Servlet) è concluso. Per mantenere la corrispondenza tra le successive invocazioni dell'orchestratore da parte delle Servlet, è possibile utilizzare come identificatore della sessione di composizione il call-ID presente nella richiesta SIP (che è lo stesso per tutti i messaggi del dialogo SIP). La Figura 6.2 rappresenta un esempio di Mashup composto da 6 Servlet, delle quali:

- 4 (colorate in verde) sono Servlet di tipo (1), quindi implementano servizi telefonici;
- 2 (colorate in rosso) sono Servlet di tipo (2), quindi richiamano servizi *non-SIP*.

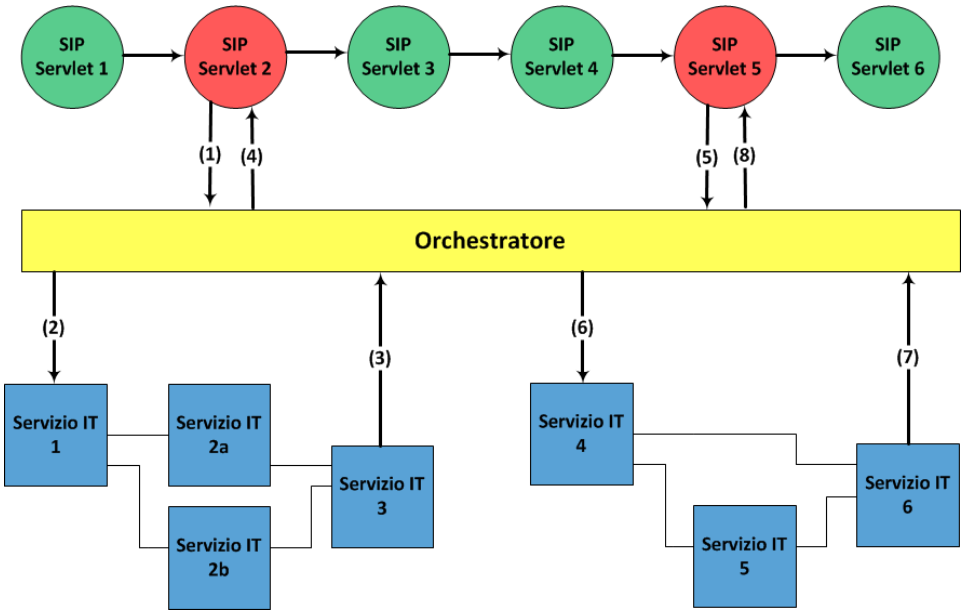


Figura 6.2: Esempio di struttura 'gerarchica' della composizione

In una ipotetica esecuzione, sarà compito della prima Servlet non-SIP (in questo caso la n°2) inserire all'interno dell'oggetto `SipSession` appartenente alla richiesta attuale un attributo contenente il `call-ID` del dialogo: questo sarà poi inserito opportunamente all'interno delle *properties*, che costituiranno un parametro del metodo `notifyEvent` alla sua prossima invocazione (1), e potrà essere usato dall'orchestratore per identificare la sessione di composizione. Quest'ultimo si comporterà nella maniera consueta per quanto riguarda la selezione dei servizi *non-SIP* (quindi attraverso il meccanismo `notifyEvent`/`invokeAction`) e quando la loro esecuzione sarà terminata (3) si occuperà di far riprendere l'esecuzione della Servlet in attesa tramite `invokeAction` (4). Nel caso in cui dovessero esserci ulteriori servizi non-SIP da invocare, il Mashup conterrà un'altra Servlet di tipo (2) (in figura la n°5) che ne permetterà l'esecuzione secondo le stesse modalità (5-6-7-8). Bisogna specificare che le Servlet SIP presenti nel servizio composito possono accedere alle *properties* delle quali hanno bisogno (o aggiungere le loro) tramite l'oggetto `SipSession`, che è legato alla richiesta in circolazione; tuttavia, le Servlet non-SIP dovranno necessariamente contenere la logica che permetta una loro 'traduzione' in parametro properties che segua le specifiche dell'orchestratore.

Infine, un aspetto da non trascurare è quello relativo all'assegnazione delle coppie <attributo, valore> che vanno a costituire le informazioni contenute nelle cosiddette `properties`: il nome degli attributi deve essere definito in maniera che le singole Servlet (ovvero i servizi di base) possano acce-

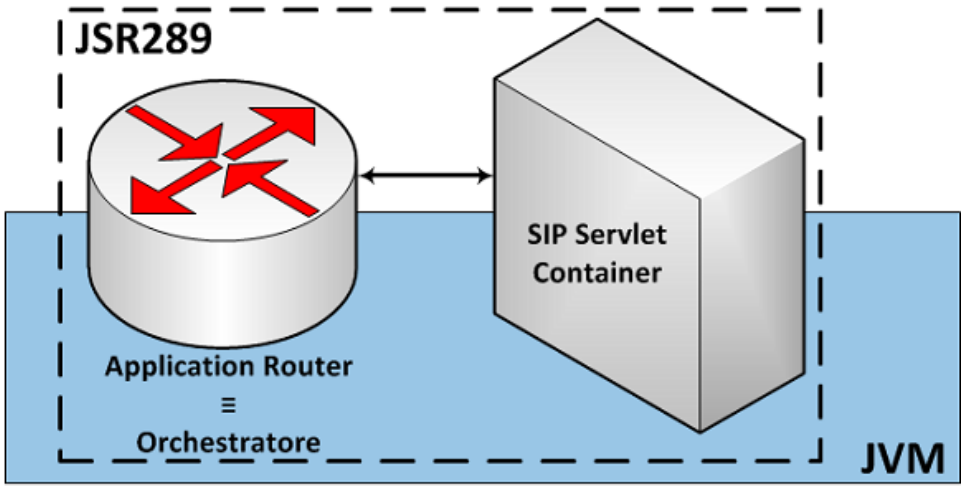


Figura 6.3: Struttura della soluzione proposta

dervi secondo un meccanismo standard. Questo aspetto, comune a molte delle soluzioni che proporremo, verrà esaminato nel seguito della trattazione.

Nota: questo tipo di soluzione nasconde in realtà un problema relativo all'esecuzione dei thread di ogni Servlet di tipo (2): ad ogni invocazione del metodo `notifyEvent` il thread che rappresenta tale Servlet resterà inattivo, perlomeno finché l'orchestratore non avrà concluso l'esecuzione dei servizi *non-SIP* associati a quella Servlet. Può quindi verificarsi la situazione in cui il *thread pool* relativo alle Servlet SIP non può più eseguire nuovi task, ed è costretto ad inserire le ultime richieste di esecuzione in attesa.

6.2.2 Soluzione 2: Application Router come orchestratore

E' possibile integrare all'interno dell'Application Router le stesse funzioni svolte dall'orchestratore; ancora una volta, il punto di collegamento tra IMS e lo strato dei servizi sarà rappresentato da un SIP Servlet Container. La Figura 6.4 illustra la struttura di questa soluzione: il componente Application Router sarà ora una versione 'potenziata' di quello esposto nella soluzione precedente, dal momento che al suo interno conterrà la logica stessa dell'orchestratore.

L'orchestratore, per poter comunicare con il SIP Servlet Container, dovrà subire una modifica alla sua interfaccia di comunicazione: se infatti nelle soluzioni precedenti esso comunicava con i SP tramite i metodi invokeAction/notifyEvent, ora dovrà utilizzare l'interfaccia standard presente tra AR e Container, definita nelle specifiche JSR289. Il Container invocherà dunque il metodo `getNextApplication` (sull'oggetto `SipApplicationRouter`)

al ricevimento di una *initial request*: tale invocazione conterrà come parametri la richiesta ricevuta, la *routing region*, la direttiva, ed eventuali altre informazioni di stato (`stateInfo`), e l'AR restituirà al Container un oggetto di tipo `SipApplicationRouterInfo` contenente le informazioni desiderate.

In questo approccio **i SP sono sostituiti dalle SIP Servlet**: ognuna conterrà, oltre alla logica del servizio, le istruzioni per il recupero delle *properties* (ovvero alle informazioni necessarie per la propria esecuzione) e per l'eventuale aggiornamento delle stesse. Poiché tuttavia le specifiche SIP Servlet 1.1 non consentono all'AR di interagire direttamente con le applicazioni, tali dati dovranno essere inseriti all'interno dell'oggetto `SipSession` associato alla richiesta iniziale (che l'AR ha a disposizione) attraverso il metodo `setAttribute`. Per chiarire il funzionamento delle varie componenti, si può ad esempio considerare un servizio composito che, prima di inoltrare la chiamata al destinatario, esamina il suo calendario online degli impegni ed in base ad esso decide se rifiutare o meno la chiamata (ad esempio perché in riunione). Tale servizio composito può essere visto come costituito da 2 servizi di base:

1. il servizio dedicato al controllo degli impegni sul calendario, che produce come *output property* un oggetto Boolean al valore true se il destinatario risulta libero, viceversa false;

2. il servizio di *Call-Blocking* che riceve come *input property* un oggetto `Boolean`: se `true` inoltra la chiamata secondo quando specificato nel campo `to` della richiesta SIP, viceversa risponde al mittente con un messaggio `SIP BUSY`.

Nella Figura 6.4 è illustrata la sequenza di azioni intraprese a seguito del ricevimento di un messaggio di `SIP INVITE` (1) da parte del Container: per prima cosa, questo invoca il metodo `getNextApplication` (2), del quale la richiesta costituisce un parametro; a questo punto l'AR consulta la struttura del Mashup selezionato dal mittente del `SIP INVITE`, imposta eventuali *properties*[1] all'interno dell'oggetto `SipSession` (specificate in fase di creazione del Mashup) e infine crea l'oggetto `SipApplicationRouterInfo` contenente il nome dell'applicazione che accede al calendario online, che viene ricevuto dal Container (3). Quest'ultimo invocherà su di esso il metodo `getNextApplicationName()`, e inoltrerà il messaggio `SIP INVITE` alla SIP Servlet corrispondente al servizio di controllo del calendario online (4). L'applicazione estrarrà dall'oggetto `SipSession` le informazioni necessarie alla verifica degli impegni (ad esempio il nome dell'utente da controllare) ed effettuerà le operazioni richieste: il risultato di tale esecuzione sarà una *output property* (un valore `Boolean`) che verrà inserito nell'oggetto `SipSession`, e costituirà l'*input property* per il servizio (ovvero Servlet) successivo. Ancora

[1] Una di esse sarà certamente l'ID della sessione di orchestrazione

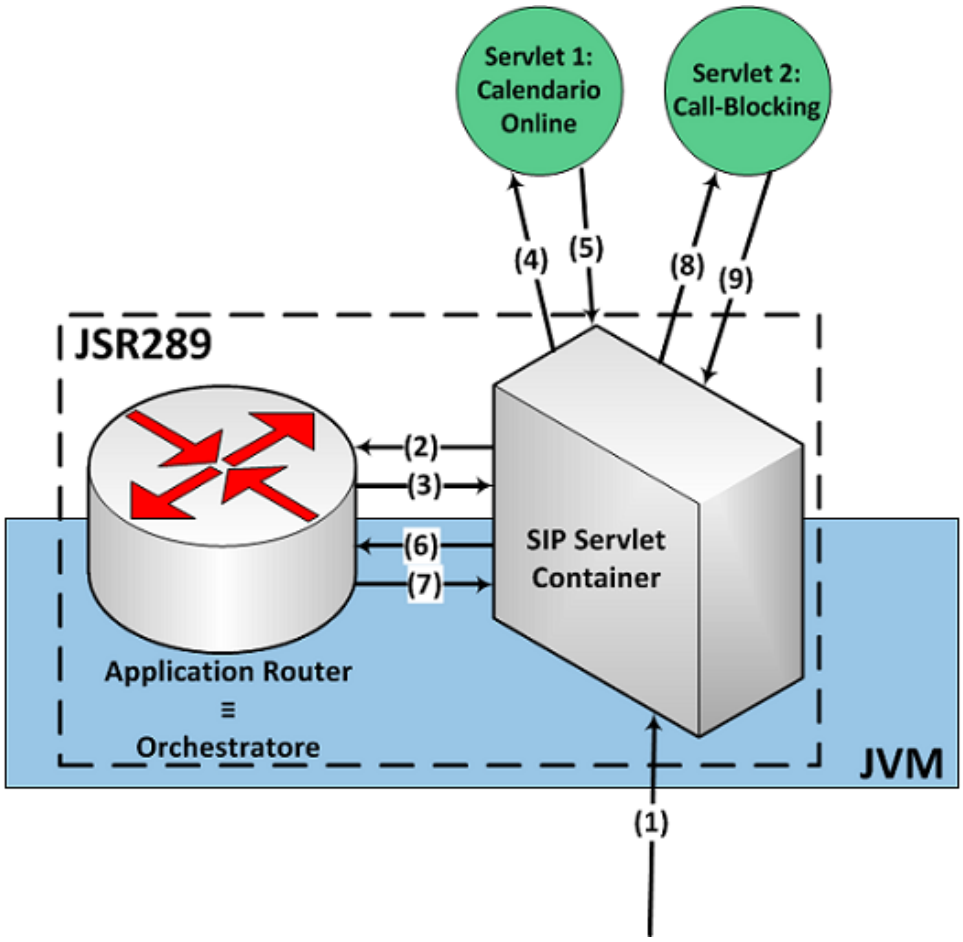


Figura 6.4: Esempio di servizio composito operante sull'architettura proposta

una volta la richiesta iniziale inoltrata al Container (5) passerà all'AR (6), che estraendo l'ID della sessione di orchestrazione dall'oggetto `SipSession` sarà in grado di stabilire che il prossimo servizio da invocare è quello di *Call-Blocking*: la selezione della Servlet corretta avviene in maniera analoga alla precedente (7-8). La Servlet dovrà solamente estrarre nel modo consueto il valore della *property* necessaria alla sua esecuzione (oggetto `Boolean`) e in base al suo valore inoltrare la richiesta oppure inviare un messaggio `SIP BUSY` al mittente (9).

6.3 Composizione di servizi generici

A differenza di quanto esposto nel paragrafo 6.2, le soluzioni che seguono non riguardano una composizione di servizi strettamente legata all'ambito telefonico: piuttosto, si considera l'orchestrazione di servizi generici, ovvero appartenenti ad un dominio qualsiasi, per i quali sia definibile un Mashup; sebbene al suo interno possano essere presenti anche *feature* telefoniche, l'evento che avvia la procedura di composizione non è necessariamente la ricezione di un `SIP INVITE` che rappresenta l'instaurazione di una chiamata.

6.3.1 Soluzione 3: Ericsson Composition Engine (ECE)

La già citata Ericsson Composition Engine (ECE) costituisce una soluzione dotata di uno spettro di tecnologie supportate molto ampio, tanto quanti sono i CEA sviluppabili. Nel contesto delle interazioni con IMS è tuttavia rilevante considerare solamente il modulo indicato come SIP CEA, che è in realtà una particolare implementazione dell'Application Router definito nelle specifiche JSR 289, dotato di alcuni 'potenziamenti', come ad esempio la possibilità di intercettare non solo le *initial request*, bensì anche i *subsequent message*. A questo proposito, bisogna sottolineare che le specifiche non prevedono un tale comportamento da parte dell'AR, che dovrebbe essere invocato dal container *solo* a seguito della ricezione da parte di quest'ultimo di messaggi di richiesta iniziali.

Dunque, in una implementazione che si basi su tali caratteristiche, l'orchestratore (chiamato in questo contesto 'Composition Engine' (CE)) dovrà:

1. fornire informazioni relative alla sequenza di composizione di servizi SIP (specificata nell'*Application Skeleton*) all'AR;

2. occuparsi direttamente dell'invocazione di servizi *non-SIP*.

Il punto (1) viene generalmente eseguito successivamente alla ricezione da parte del SIP CEA (che costituisce il punto di collegamento tra il core di IMS e il CE) di un messaggio `SIP INVITE` (o comunque iniziale): quest'ultimo viene inoltrato secondo il consueto meccanismo basato sull'impostazione di specifici *initial Filter Criteria* all'interno dell'HSS.

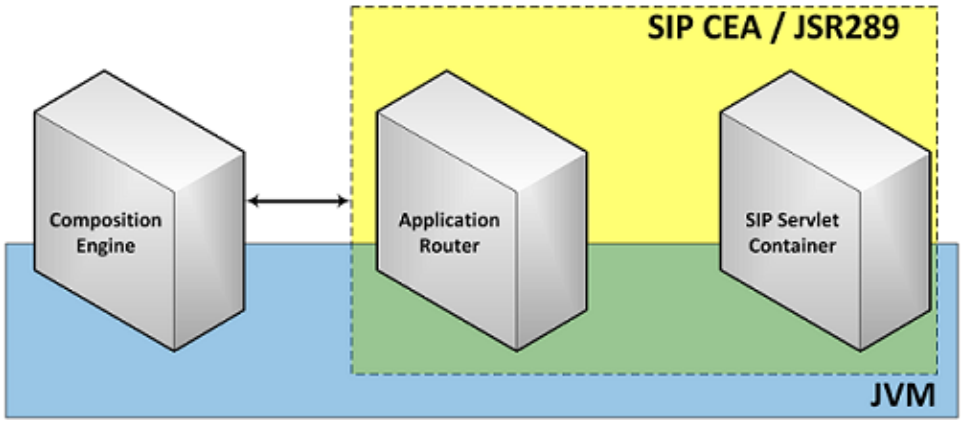


Figura 6.5: ECE: interazione tra Composition Engine e SIP CEA

Nella Figura 6.5, in cui è evidenziata la struttura logica delle interazioni tra il CE e il SIP CEA, i vari componenti risiedono dunque sulla medesima JVM: pertanto, lo scambio di messaggi tra le due entità avviene attraverso l'utilizzo delle API comuni definite per il CE e di API specifiche per quanto riguarda il SIP CEA.

La principale caratteristica architetturale di ECE risiede dunque nella distinzione tra la funzione di **composizione** di servizi, svolta dal Composition Engine, e la funzione di **esecuzione** degli stessi, delegata ai differenti CEA sulla base dello specifico servizio selezionato in fase di run-time: questa separazione permette una facile integrazione, nello stesso sistema, di domini differenti (telecomunicazioni, IT, ecc.); il CE si propone appunto come *mediatore* tra tali domini. La presenza di quest'ultimo permette poi di condividere, secondo delle regole standard definite per il già citato *shared state*, tutte le informazioni ricavate dai CEA: in questo modo si realizza la possibilità di comunicazione tra servizi appartenenti a domini differenti e realizzati con qualunque tecnologia. Dunque, informazioni come i parametri dei messaggi SIP possono essere utilizzati dal CE come input per servizi che richiedono il profilo dell'utente o la sua posizione geografica; allo stesso modo questi stessi dati possono essere usati per modificare un messaggio SIP prima del suo inoltro all'interno della rete sottostante.

Dettagli di esecuzione

Alla ricezione di una richiesta iniziale, il SIP Container crea le varie parti necessarie a costituire lo stato della selezione di servizi nel seguente modo:

- Se la richiesta proviene da una entità SIP esterna, viene impostata la direttiva (in precedenza chiamata *routing directive*) `NEW`. Viceversa,

se la richiesta proviene da una applicazione, la direttiva è impostata implicitamente o esplicitamente da essa;

- Se la richiesta proviene da una applicazione, e la direttiva è `CONTINUE` o `REVERSE`, il parametro stateInfo viene impostato al valore che aveva nella richiesta originale (alla quale questa fa riferimento). Viceversa, tale parametro non viene impostato;
- I valori Subscriber URI e Routing Region non vengono attualmente impostati.

A questo punto si esegue la seguente procedura:

1. Viene invocato il metodo `SipApplicationRouter.getNextApplication()` dell'Application Router: questo metodo serve al Container per ottenere informazioni in merito alla successiva applicazione da contattare. Tali informazioni sono fornite dall'AR sotto forma di oggetto di tipo `SipApplicationRouterInfo`, che nel seguito chiameremo `result`;
2. Viene invocato metodo `getRouteModifier()` sull'oggetto result: questo restituirà indicazioni su come interpretare i valori ottenuti con `result.getRoutes()`;
3. Viene invocato il metodo `result.getNextApplicationName()`: questo permette di ottenere il nome della prossima applicazione da contattare, che potrà essere selezionata secondo i consueti metodi specificati in ambito Servlet[2]. Se tuttavia l'invocazione del metodo restituisce `null`, significa che non ci sono ulteriori applicazioni da contattare, e pertanto il Container può procedere all'inoltro del messaggio SIP nella rete secondo quanto prescritto dal protocollo.

In questa descrizione si è dato per assunto che l'AR possa accedere alle informazioni relative alla composizione dei servizi contenute nel Composition Engine, operazione preliminare a quella di selezione vera e propria delle applicazioni.

6.3.2 Soluzione 4: SIP Servlet e piattaforma proprietaria

Architettura di riferimento

In [59] viene proposta una piattaforma per l'esecuzione di servizi (Service Execution Platform (SEP)) che permette la composizione degli stessi in domini caratterizzati dalla presenza di eventi asincroni. Sebbene la specifica

[2]Per poter identificare le applicazioni SIP presenti nel server, il Container utilizza le informazioni contenute nei *deployment descriptor* di ogni applicazione, rappresentati dai file `sip.xml`. In particolare l'elemento <app-name> conterrà il nome dell'applicazione cui il file fa riferimento.

implementazione delle componenti differisca da quella realizzata da Ericsson, la piattaforma presenta entità dotate di caratteristiche funzionali molto simili a quelle presenti nell'ECE. In particolare, i componenti principali sono:

- **Service Proxy (SP)**: ogni SP permette l'interazione tra un tipo di servizi (esterni rispetto alla piattaforma) e l'orchestratore, secondo un modello ad eventi. Questi componenti sono necessari in quanto i servizi sono progettati per essere entità indipendenti, e quindi non dispongono di interfacce per la reciproca interazione. Gli SP espongono un metodo, chiamato `invokeAction`, che può essere chiamato per attivare le funzionalità del SP. Un solo SP è necessario per poter utilizzare una risorsa esterna, sebbene per motivi di performance e fault tolerance se ne possano utilizzare in numero maggiore. In sostanza, questi componenti svolgono funzionalità analoghe ai CEA in ECE.

- **Orchestration Management System (OMS)**: permette agli utenti di attivare/disattivare i servizi compositi richiesti, ovvero la particolare istanza di Mashup: in tale circostanza, l'OMS richiede all'utente di inserire eventuali proprietà richieste dal Mashup, e procede all'attivazione del primo SP mediante invocazione del metodo `invokeAction`. L'esecuzione vera e propria del Mashup avverrà alla notifica di un evento iniziale da parte del primo SP verso l'OMS stesso: quest'ultimo gestisce questo tipo di eventi in modo da poter selezionare uno specifico *Orchestration Node (ON)* secondo le politiche di gestione del carico adottate.

- **Orchestration Node (ON)**: questo componente si occupa di combinare i differenti SP al fine di eseguire la composizione di servizi definita nel Mashup. Esso espone il metodo `notifyEvent` attraverso il quale un SP può notificare nuovi eventi all'ON: sarà compito di quest'ultimo occuparsi delle successive invocazioni, secondo l'algoritmo di routing riportato di seguito. L'ON è logicamente corrispondente alla Composition Engine descritta in ECE.

```
1  Parametri di Input: Event_Id,
2                      SenderSP_Id,
3                      Session_Id,
4                      Properties
5  {
6         1. Recupera la Routing Table (RT), generata
7            dall'ON, contenente le informazioni
8            relative all'ordine di esecuzione dei
9            servizi del Mashup
10        2. Recupera l'entry della RT necessaria ad
11           invocare la/e azione/i successiva/e,
12           utilizzando la coppia <SenderSP_Id,Event_id>
```

```
13              come chiave
14          3. Per ogni azione trovata (ovvero ogni arco
15              della descrizione grafica del Mashup) fa
16              quanto segue:
17              a. Aggiorna l'array di Proprietà basandosi
18                  sulle assegnazioni corrispondenti al
19                  particolare arco considerato
20              b. Invoca il metodo invokeAction del
21                  prossimo SP con i seguenti parametri
22                  di input:
23                  ON_Endpoint,
24                  Action_Id,
25                  TargetSP_Id,
26                  Session_Id,
27                  UpdateProps.
28  }
```

Il parametro `ON_Endpoint` presente al punto 3.b. dell'algoritmo è fornito al SP invocato in modo che quest'ultimo sappia quale ON contattare per le notifiche successive (qualora vi siano più ON).

Dettagli

Una soluzione alternativa a quella proposta da Ericsson, ma comunque basata sull'utilizzo di un SIP Servlet Container, può far uso dell'architettura presentata nel paragrafo 6.3.2, alla quale tuttavia vanno apportate delle modifiche per poter sfruttare le potenzialità offerte dall'Application Router.

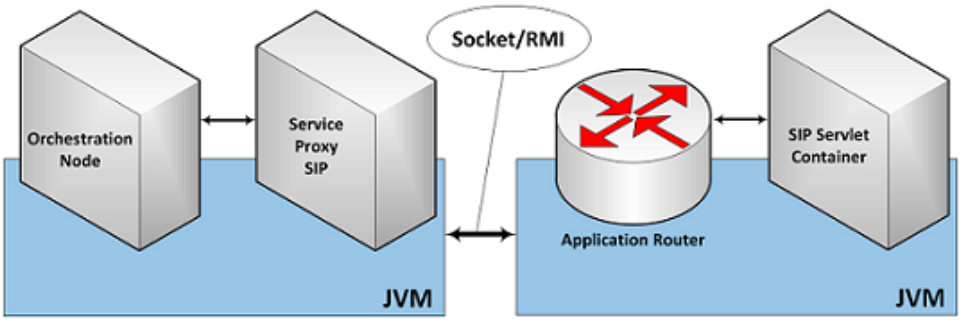


Figura 6.6: Struttura della soluzione proposta: SIP Servlet e piattaforma proprietaria

Da un punto di vista funzionale, data la corrispondenza di caratteristiche con ECE, l'AR che si dovrà implementare sarà del tutto analogo a quello descritto per la soluzione precedente, salvo tuttavia tener presente che in questo caso si richiede una specifica modalità di interazione con l'orchestra-

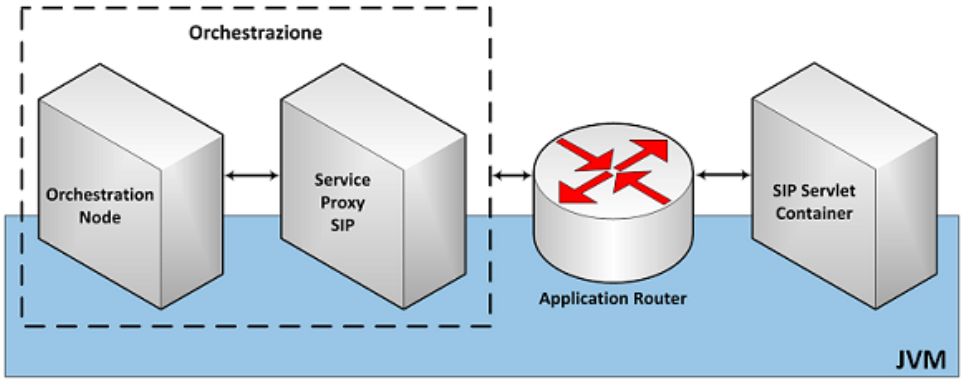


Figura 6.7: Variante della soluzione precedente

tore, ovvero tramite l'invocazione dei metodi `notifyEvent` e `invokeAction`. A questo proposito, la comunicazione tra le due componenti sarà resa possibile dal Service Proxy SIP, che effettuerà lo scambio di messaggi mediante l'apertura di un socket UDP che consenta il passaggio della segnalazione SIP, oppure con l'utilizzo di un sistema di Remote Method Invocation (RMI). Si veda la Figura 6.6.

Le possibilità di miglioramento delle prestazioni, in questo approccio, risiedono sostanzialmente nella scelta della particolare modalità di interazione tra le coppie orchestratore/Service Proxy SIP e Application Router/SIP Servlet Container: difatti, le comunicazioni all'interno delle coppie avvengono tramite metodi consolidati che non ha senso modificare (meccanismo `invokeAction`/`notifyEvent` per la prima, utilizzo del metodo `getNextApplication()` per la seconda). Questa soluzione l'utilizzo di due entità distinte, ovvero un SIP Servlet Container (dotato di Application Router) e una componente per l'orchestrazione e l'esecuzione dei servizi: si è quindi mantenuta una distinzione tra le funzioni più specificatamente *telco* (AR e SIP Servlet) e quelle genericamente identificate come IT.

E' comunque possibile adottare una soluzione che parta invece dal presupposto di unire questi due domini non solo da un punto di vista logico, ma anche implementativo: la Figura 6.7 illustra tale proposito rappresentando tutti i componenti precedentemente elencati come appartenenti ad una stessa JVM.

Punto di vista privilegiato è dunque quello dell'operatore *telco*, che vuole integrare all'interno della sua infrastruttura tutte le componenti necessarie alla creazione di servizi compositi. Il comportamento dei singoli moduli è esattamente equivalente a quanto specificato in precedenza: quello che cambia è l'interazione tra l'AR e le altre componenti dedicate all'orchestrazione dei servizi, che non dovrà più avvenire tramite socket/RMI ma secondo le API specificate dall'orchestratore per la notifica degli eventi esterni.

6.3.3 Soluzione 5: basata sulle librerie JAIN SIP

Considerando sempre come modello di riferimento l'architettura presentata nel paragrafo 6.3.2, è possibile proporre una soluzione alternativa a quelle descritte in precedenza, che non necessita di un SIP Servlet Container. In particolare, il SP destinato all'esecuzione dei servizi SIP conterrà al suo interno della logica che permette di gestire in maniera *diretta* (cioè senza l'intervento di un Application Router e quindi di un SIP Servlet Container) le interazioni con IMS ed i servizi SIP, mediante l'utilizzo delle librerie JAIN SIP [14] (queste ultime costituiscono una implementazione completa di quanto specificato nella RFC 3261, relativa al protocollo SIP). Tale SP interagisce con l'ON nella maniera consueta, quindi attraverso l'invocazione dei metodi `notifyEvent` e `invokeAction`, ma utilizzando in aggiunta delle strutture dati che consentono di stabilire la corrispondenza tra l'**utente** che richiede un servizio composito e la particolare **sessione dell'orchestratore**. Difatti, a seguito di un `invokeAction`, l'orchestratore fornisce al SP un identificatore della sessione che potrà essere inserito da quest'ultimo nella successiva invocazione di `notifyEvent`, così da permettere all'orchestratore di continuare con la selezione dei servizi per quella particolare sessione. Analogamente a quanto succede in ECE, è quindi il SP (CEA nell'ECE) a costituire il punto di ingresso dei messaggi SIP provenienti dal core di IMS, esponendo un socket UDP che permette di ricevere la segnalazione SIP e lo pone logicamente al livello di un Application Server: è dunque l'opportuna configurazione degli iFC nell'HSS a provocare l'inoltro di tali messaggi verso il SP, il quale provvederà all'invocazione del `notifyEvent` che consentirà l'avvio della procedura di orchestrazione.

Nota: resta inteso che sarà compito dell'entità chiamata Orchestration Management System (OMS) attivare/disattivare le istanze di Mashup selezionate precedentemente dagli utenti, (cosicché l'ON venga a conoscenza della loro struttura e ne costruisca la Routing Table), nonché aggiungere nell'HSS gli *initial Filter Criteria* che permettano l'inoltro dei messaggi SIP al relativo SP.

Il mantenimento della corrispondenza tra utente e sessione dell'orchestratore può essere rafforzato per esempio tramite l'utilizzo del cosiddetto **call-ID**, cioè quel campo presente in ogni messaggio SIP che ne specifica l'appartenenza ad un determinato dialogo in corso. Quindi, nel momento in cui il SP riceve dall'orchestratore l'`invokeAction` (indicato con (1) nella Figura 6.8) contenente il session-ID ed eventuali altre informazioni nel campo `properties` (come l'identità del sottoscrittore al servizio), procede all'aggiunta di una entry <session-ID, user-ID> nella struttura dati di supporto, dove il secondo campo costituisce la chiave. In questo modo, alla ricezione di un messaggio SIP (2), il SP esaminerà al suo interno l'identità del mittente e la utilizzerà per poter estrarre dalla struttura dati il relativo valore `session-ID`, ed aggiungere alla entry un terzo campo, corrispondente al va-

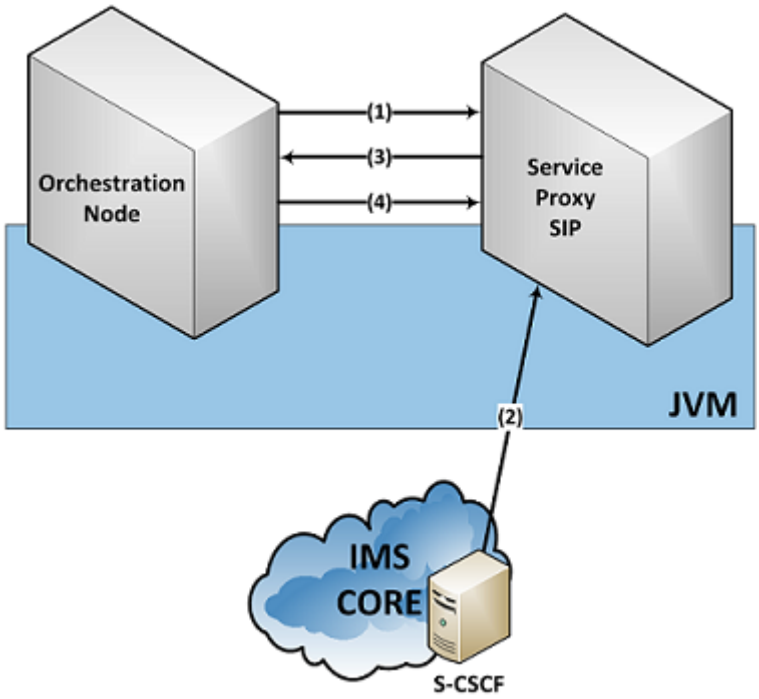


Figura 6.8: Interazioni fra le componenti dedicate alla composizione di servizi

lore del call-ID presente nel messaggio; dopo queste operazioni la struttura dati conterrà la tripla <session-ID, user-ID, call-ID>, ed il SP potrà invocare il metodo `notifyEvent` (3), con parametro di ingresso il `session-ID` appena estratto, e l'orchestratore provvederà a fornirgli una risposta tramite il metodo `invokeAction` (4).
Nota: non è necessario effettuare un controllo sull'identità del mittente: difatti, avendo impostato precedentemente gli *initial Filter Criteria*, il SP avrà la certezza che tutti i messaggi SIP da esso ricevuti appartengono effettivamente ad utenti che hanno sottoscritto qualche servizio composito, e per il quale dunque sarà già presente una entry nella struttura dati.

L'utilizzo del parametro `call-ID` consente al SP di identificare anche i messaggi *subsequent* appartenenti ad un dialogo che interessi uno dei sottoscrittori; difatti, dopo aver invocato il metodo `notifyEvent`, il SP riceverà dall'orchestratore l'indicazione del servizio successivo al quale inoltrare il messaggio[3]: mediante l'inserimento del proprio indirizzo nel campo `Record-Route`, il SP si inserirà quindi nel path del dialogo in corso, e avrà

[3]E' competenza dell'orchestratore fornire al SP le informazioni per poter contattare il servizio successivo, ad esempio mediante un indirizzo SIP che identifichi l'Application Server nel quale questo è collocato.

eventualmente modo di effettuare operazioni *anche* a seguito della ricezione di messaggi SIP *non* iniziali. Questo aspetto costituisce una delle differenze sostanziali rispetto all'approccio basato su SIP Servlet e quindi Application Router: quest'ultimo, infatti, secondo le specifiche JSR 289, non ha la possibilità di ottenere dal Container messaggi che non siano *initial request*, e pertanto può operare decisioni basandosi esclusivamente su di essi. E' per questo motivo che ECE propone un 'potenziamento' dell'AR affinché possa ricevere anche messaggi *subsequent*, cosa che tuttavia sembra costituire una violazione dello standard. La soluzione appena descritta fa invece uso di tali messaggi in maniera perfettamente lecita, dal momento che il suo comportamento è quello di un proxy che inserisce il suo indirizzo nel campo `Record-Route`, circostanza corretta in una segnalazione SIP.

La Figura 70 rappresenta tutti gli attori interessati dalla situazione descritta: essa può essere suddivisa in due fasi, indicate nell'immagine dal colore delle frecce. Le frecce nere sono relative alla fase di attivazione del servizio composito: come si vede, un utente (User Agent) effettua la selezione del Mashup contattando l'Orchestration Management System (OMS); questo si occupa quindi di informare l'HSS sugli *initial Filter Criteria* da impostare (tramite l'interfaccia Sh, basata sul protocollo DIAMETER) e di segnalare all'orchestratore quale Mashup è stato selezionato per l'utente.

Le frecce rosse fanno invece riferimento alla seconda fase, che consiste nell'erogazione vera e propria del servizio composito: l'orchestratore inizializza la procedura di composizione tramite l'invocazione del metodo `invokeAction`, contenente i parametri specificati in precedenza; il S-CSCF, sulla base delle informazioni ricevute dall'HSS, inoltrerà al Service Proxy SIP i messaggi che interessano il Mashup attivato, e quest'ultimo si occuperà di segnalare il loro arrivo mediante l'invocazione del metodo `notifyEvent`. L'ultima azione di questa fase consiste nell'attivazione vera e propria del primo servizio SIP del Mashup, ovvero l'invio verso questo del messaggio interessato da parte del SP.

6.4 Confronto tra le soluzioni proposte

Nei paragrafi precedenti sono state presentate le soluzioni ritenute più valide per il problema della composizione di servizi nel contesto IMS e, più in generale, per l'interazione tra *feature* telefoniche e servizi di altro genere. Ogni architettura ha sia pregi che difetti, e l'obiettivo di questo paragrafo è proprio quello di metterli in luce: sulla base di queste informazioni, lo sviluppatore di servizi compositi ha quindi modo di scegliere la soluzione più adatta alle proprie esigenze.

Facendo riferimento a **Ericsson Composition Engine**, nell'approccio presentato per l'interazione con IMS riveste grande spessore la presenza di un SIP Servlet Container ed in particolare dell'Application Router, suo

componente interno: tale soluzione ha il privilegio di essere 'naturale' per un utilizzo al di sopra della rete IMS, dal momento che fa uso di componenti dotati di funzionalità consolidate per poter interagire con il protocollo SIP. La composizione di servizi per mezzo dell'AR viene svolta in maniera aderente a quanto richiesto nel contesto della telefonia, ovvero tramite una loro concatenazione in sequenza (*feature chaining*). Inoltre, le SIP Servlet API sono una tecnologia conosciuta da un gran numero di sviluppatori, dal momento che si basano sul modello di programmazione delle HTTP Servlet: attualmente vi sono numerose implementazioni di queste specifiche, sia commerciali che open source, il che rende tale standard appetibile.

Bisogna tuttavia considerare, per questa soluzione come pure per le altre basate sulle SIP Servlet, che l'utilizzo di un SIP Servlet Container potrebbe non essere strettamente necessario: sulla base di questa osservazione è stata proposta una soluzione che prevedesse lo sviluppo di un componente in grado di gestire la segnalazione SIP tra IMS e l'orchestratore mediante l'utilizzo di librerie che realizzano lo stack SIP. Lo svantaggio di quest'ultima proposta è costituito dal dover realizzare all'interno di tale componente la logica per la gestione del protocollo SIP a basso livello, operazioni che invece un SIP Servlet Container esegue di default.

Analizzando più in generale le soluzioni proposte, si nota che uno dei problemi più significativi è costituito dalle modalità di **passaggio delle informazioni** (spesso chiamate *properties*) tra i differenti servizi: lo standard SIP Servlet non permette infatti un meccanismo di comunicazione diretto tra le differenti applicazioni SIP, anche se installate nel medesimo Container. Il meccanismo presentato in precedenza, basato sull'utilizzo dell'oggetto `SipSession`, prevede infatti la definizione dei parametri utilizzati dai servizi secondo una nomenclatura standard: facendo riferimento ancora una volta all'esempio di chiamata e consultazione di un calendario online, si nota che gli attributi utilizzati dal primo servizio (ad esempio *nome utente* e *data*) devono essere inseriti nell'oggetto `SipSession` con un nome noto ad esso; allo stesso modo, l'output di tale servizio dovrà essere rappresentato con un nome che possa essere riconosciuto dalla successiva applicazione di *Call Blocking*. In generale, dunque, il nome di un determinato attributo identificherà un particolare tipo di contenuto, e le varie applicazioni dovranno basarsi su tale corrispondenza durante le fasi di estrazione e inserimento delle *properties* all'interno delle invocazioni. La soluzione Ericsson risolve il problema della nomenclatura, così come quello del passaggio di informazioni, facendo uso del cosiddetto *shared state*, ovvero l'insieme di tutte le informazioni relative alla sessione di orchestrazione (sotto forma di variabili) che possono essere oggetto di scambio tra i vari servizi di base.

Altra importante differenza tra gli approcci risiede nel **modo di costruire la sequenza che descrive il Mashup di servizi**: ECE si basa sulla rappresentazione tramite *application skeleton*, più ricca rispetto ad una semplice sequenza di servizi eventualmente corredati da informazioni di in-

put. La prima soluzione permette infatti di selezionare durante la creazione del Mashup una 'funzione' piuttosto che un servizio vero e proprio, dal momento che il binding dell'applicazione viene effettuato durante l'esecuzione, e può basarsi su politiche di load balancing.

Inoltre, in tutte le soluzioni considerate si è sempre resa **necessaria l'introduzione di un orchestratore**, come un componente a sé stante oppure integrato all'interno di altre entità standard: a seconda dell'approccio, sono cambiate le reciproche modalità di comunicazione, e quindi si è resa necessaria la modifica di qualche altro componente. La soluzione 4 obbliga lo sviluppatore di servizi a crearli come SIP Servlet, sforzo comunque comparabile a quello di realizzare dei Service Proxy che implementino la logica `invokeAction`/`notifyEvent`, utilizzata nelle altre soluzioni basate su Servlet.

E' poi sensato privilegiare le soluzioni che minimizzano il numero delle componenti da contattare per ogni richiesta: ad ogni passaggio corrispondono infatti delle operazioni di *marshalling* e *unmarshalling* dei dati in transito (o comunque una loro traduzione secondo gli standard richiesti) che devono essere implementate in tutte le componenti in comunicazione. Ad esempio la Soluzione 2 prevede, in aggiunta rispetto alle altre, l'apertura di un socket UDP (o di un canale di comunicazione qualsiasi) tra i Service Proxy e il Servlet Container, e ogni richiesta dovrà attraversarlo sia in seguito ad un `notifyEvent`, che a seguito di un `invokeAction` che faccia uso di un servizio SIP.

Infine, alcune soluzioni trovano una immediata integrazione in quelle che sono le strutture già esistenti dell'operatore *telco*, mentre altre propongono un approccio che tende a differenziare l'ambito delle *feature* telefoniche da quello dei servizi IT in senso generale: appartengono alla prima categoria le soluzioni 3 e 5, mentre le altre (seppur in misura differente) appartengono alla seconda.

Capitolo 7

Implementazione di un Service Proxy interagente con IMS

Indice

Nel capitolo precedente sono state descritte varie soluzioni architetturali al problema dell'orchestrazione di servizi, sia in ambito strettamente telefonico che nel dominio più generico dei servizi Web; in ogni soluzione è possibile riconoscere due tipologie di componenti:

1. che realizzano funzionalità di controllo ed orchestrazione;
2. che realizzano i servizi veri e propri.

La prima categoria contiene quindi tutti quegli elementi il cui scopo è la gestione della corretta interazione fra le invocazioni e i flussi di dati tra i servizi: un esempio è l'orchestratore, sia esso realizzato da un Application

Router all'interno di un Servlet Container, che come componente a sé stante. In generale la logica realizzata da queste entità è identica per tutte le soluzioni, e le differenze risiederanno soprattutto nelle modalità di interfacciamento con gli altri moduli presenti nella piattaforma.

La seconda categoria riguarda invece i servizi veri e propri, quindi applicazioni che possono eseguire le operazioni più disparate: nella precedente trattazione (Paragrafo 6.3.2) queste entità sono state definite *Service Proxy*. Questi sono costituiti per la maggior parte da codice dedicato alle operazioni specifiche svolte dal servizio, e in misura minore dalla logica necessaria all'interazione con gli altri componenti: quest'ultima può essere considerata comune a tutti i *Service Proxy*, dal momento che le procedure per la comunicazione tra le diverse entità interessate nell'orchestrazione è definita dal particolare modello architetturale scelto, ed è identica per ogni tipo di servizio implementato. In questo capitolo l'attenzione sarà rivolta a quest'ultima categoria di componenti, ed in particolare verranno descritte nel dettaglio le caratteristiche di un *Service Proxy* in grado di interagire con la rete IMS.

7.1 Comunicazione tra AS e HSS: l'interfaccia Sh

La comunicazione tra un Application Server e l'Home Subscriber Server è definita nei documenti 3GPP TS 29.328 [60] e TS 29.329 [61], e le operazioni effettuabili tramite questa interfaccia sono suddivise in due gruppi funzionali:

1. Procedure per la gestione dei dati, quali:
   - recupero dei dati presenti nell'HSS da parte dell'AS;
   - aggiornamento dei dati presenti nell'HSS;
2. Procedure di sottoscrizione/notifica:
   - un AS può effettuare la sottoscrizione per ricevere notifiche qualora avvenga una modifica dei dati presenti nell'HSS;
   - l'HSS può informare l'HSS dei cambiamenti nei dati per i quali l'AS aveva precedentemente effettuato la sottoscrizione.

Nello sviluppo di un *Service Proxy*, verranno utilizzate solamente le procedure indicate nel primo gruppo, in particolare quelle che permettono all'AS di ricevere i messaggi SIP rilevanti per il servizio erogato, operazione che richiede la manipolazione degli *initial Filter Criteria* salvati nell'HSS.

Il protocollo utilizzato per la comunicazione attraverso l'interfaccia Sh è Diameter, definito nella RFC 3588, successore del protocollo RADIUS (Remote Dial-IN User Service - RFC 2865).

7.1.1 Il protocollo Diameter

Diameter è un protocollo AAA (Authentication, Authorization, Accounting) composto da un protocollo di base (quello definito nella RFC 3588) e da un set di estensioni chiamate 'applicazioni Diameter': queste aumentano le funzionalità del protocollo di base (comunque non utilizzabile autonomamente) fornendo procedure e servizi aggiuntivi. Esempi di applicazioni Diameter sono Diameter Mobile IPv4 (RFC 4004) e Diameter Network Access Server Application (RFC 4005). Tutte le applicazioni Diameter devono comunque supportare le funzionalità di base del protocollo ed utilizzare le procedure di instradamento e la struttura dei messaggi definite nell'RFC 3588.

Messaggi Diameter

I messaggi Diameter sono costituiti da un **Header**, che descrive il tipo di messaggio, e da una serie di **Attribute Value Pairs (AVP)** che contengono dati differenti a seconda del tipo di messaggio specificato. La Figura 71 illustra questa struttura.

Diameter definisce un solo tipo di messaggi, pertanto la distinzione tra essi avviene principalmente attraverso il **Command Code** e la presenza di una flag di Richiesta/Risposta: le applicazioni Diameter possono ad esempio estendere la lista di Command Code definita per il protocollo base, così da permettere nuove operazioni. I Command Code da 0 a 255 sono riservati per la retro compatibilità con RADIUS, mentre la Tabella 7.1 indica quelli definiti per il protocollo di base.

Command Name	Abbreviazione	Codice
Abort-Session-Request	ASR	274
Abort-Session-Answer	ASA	274
Accounting-Request	ACR	271
Accounting-Answer	ACA	271
Capabilities-Exchange-Request	CER	257
Capabilities-Exchange-Answer	CEA	257
Device-Watchdog-Request	DWR	280
Device-Watchdog-Answer	DWA	280
Disconnect-Peer-Request	DPR	282
Disconnect-Peer-Answer	DPA	282
Re-Auth-Request	RAR	258
Re-Auth-Answer	RAA	258
Session-Termination-Request	STR	275
Session-Termination-Answer	STA	275

Tabella 7.1: Command Code definiti per il protocollo base Diameter

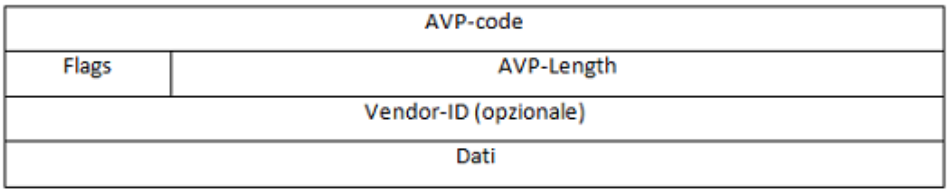


Figura 7.1: Struttura di un AVP

Per ogni Command Code è specificato un insieme di AVP da aggiungere obbligatoriamente al messaggio Diameter: gli AVP rappresentano i dati inviati attraverso Diameter, e possono essere utilizzati dal protocollo stesso oppure inviati ad applicazioni che lo supportino. Inoltre, un messaggio Diameter può contenere, oltre agli AVP obbligatori, altri AVP arbitrari, sempre che questi non siano esplicitamente esclusi dalle specifiche. In generale, comunque, gli AVP servono i seguenti scopi:

1. Trasporto di informazioni di autenticazione;
2. Trasporto di informazioni specifiche di autorizzazione che permettano di accettare o meno la richiesta di un utente;
3. Scambio di informazioni riguardanti l'uso di risorse;
4. Instradamento e reindirizzamento di messaggi Diameter in una gerarchia di server.

Ogni AVP è identificato da un codice (AVP-code) e da un Vendor-ID: ad ogni applicazione Diameter è assegnato un codice che rappresenta la compagnia ed un insieme di valori utilizzabili per il codice degli AVP. Se l'indicazione Vendor-ID non è presente, significa che l'AVP-code appartiene al protocollo base Diameter. Ad esempio, se il Vendor-ID è impostato al valore 10415 e l'AVP-code è 700, significa che la compagnia è 3GPP e l'AVP è User-Identity. La Figura 7.1 mostra la struttura di un AVP.

Entità Diameter

Il protocollo Diameter può essere considerato peer-to-peer, dal momento che qualsiasi client che supporta Diameter può inviare una richiesta (Diameter Request) ad un altro, il quale, se possibile, fornirà una risposta (Diameter Answer); tali ruoli, pertanto, sono interscambiabili. Nelle specifiche del protocollo base sono definiti una serie di entità presenti all'interno dell'infrastruttura; i principali sono:

- **Diameter Node**: un processo che implementa il protocollo base Diameter;
- **Diameter Peer**: un Diameter Node che possiede una connessione diretta con un altro Diameter Node;
- **Diameter Agent**: un dispositivo che effettua operazioni di instradamento, reindirizzamento e traduzione per i messaggi Diameter;
- **Diameter Server**: un nodo che gestisce le richieste di autenticazione, autorizzazione e accounting per un particolare dominio; oltre al protocollo base, deve supportare applicazioni Diameter;
- **Diameter Client**: un dispositivo che accede alla rete in maniera controllata.

Nella terminologia di Diameter, con il termine **connessione** si intende un collegamento a livello di trasporto tra due peer, utilizzato per inviare e ricevere messaggi Diameter. Una **sessione** è invece un concetto logico riguardante il livello applicativo, è condivisa tra un Diameter Client e un Diameter Server ed è identificata attraverso un `Session-Id` AVP (si veda la Figura 73).

Ogni nodo Diameter tiene traccia delle richieste in uscita salvando gli identificatori (univoci) *end-to-end* e *hop-by-hop*. Per ogni sessione, un peer mantiene tali identificatori finché non viene ricevuta la corrispondente risposta. Ogni nodo Diameter mantiene poi una lista di Peer conosciuti (Peer Table): al suo interno sono presenti informazioni quali l'identità, le informazioni di stato e le funzionalità dei peer. Per ogni sessione Inoltre, i Diameter Agent che si occupano dell'instradamento dei messaggi, mantengono tabella relativa al dominio (Realm-Routing Table) utilizzata per inoltrare i messaggi sulla base del meccanismo *hop-by-hop*.

Le sessioni aperte devono essere mantenute tali attraverso l'invio di messaggi detti `Device-Watchdog`: se un peer non risponde al messaggio (inviato periodicamente) di `Device-Watchdog-Request` (DWR) con un `Device-Watchdog-Answer` (DWA) esso viene considerato inattivo, e la sessione viene chiusa.

Scambio di messaggi Diameter

Il routing dei messaggi Diameter è basato principalmente sull'utilizzo degli AVP `Destination-Realm` e `Destination Host`: questi AVP sono utilizzati quando la richiesta non viene inviata direttamente al peer che la deve ricevere (ad esempio per i messaggi CER o DWR - si veda la Tabella 7.1). Qualora un messaggio sia destinato ad uno specifico server di un particolare dominio, verranno utilizzati entrambi gli AVP; viceversa, se non è specificato il server, l'AVP `Destination-Host` può essere omesso.

Se il Diameter Peer è presente nella Peer Table, il messaggio viene inviato direttamente al destinatario, mentre in caso contrario sarà un Diameter Agent ad effettuare il corretto instradamento controllando i due AVP appena citati. Ad ogni richiesta Diameter corrisponde sempre una risposta o un eventuale messaggio d'errore: una risposta ha lo stesso Command Code e identificatore *end-to-end* della richiesta; inoltre, essa conterrà un `Result-Code` AVP che indica se la richiesta è andata a buon fine o se ha invece dato luogo ad un errore. Gli errori Diameter possono essere di due tipi:

1. **Errori di protocollo**: errori di trasmissione o instradamento. Quando un nodo Diameter rileva tale errore, risponde con un messaggio in cui viene impostato l'*Error-bit*: in messaggi di questo tipo è presente anche un AVP `Result-Code` nel quale viene specificata l'entità dell'errore;

2. **Errori di applicazione**: errori dipendenti da configurazioni errate degli AVP o da problemi nella logica dell'applicazione. Messaggi di questo tipo **non** contengono mai l'*Error-bit* impostato ad 1 (come negli errori di protocollo), ma in essi è comunque presente l'AVP `Result-Code` che descrive il tipo di errore. Ad esempio, un errore di questo tipo potrebbe verificarsi se un AVP contiene un valore non ammesso oppure se il messaggio Diameter contiene un AVP non supportato.

L'AVP `Result-Code` è in realtà presente in ogni messaggio Diameter di risposta, e informa il mittente circa il risultato della sua richiesta: tale AVP contiene un numero che specifica la causa dell'errore, secondo quanto indicato nella Tabella 7.2.

7.1.2 L'applicazione Diameter per l'interfaccia Sh

Il protocollo base Diameter è esteso per poter svolgere operazioni sull'interfaccia Sh attraverso una applicazione Diameter per tale interfaccia: essa definisce nuovi Command Code utilizzabili nelle interazioni tra HSS e AS, come pure nuovi AVP da inserire all'interno dei messaggi Diameter. IANA (Internet Assigned Numbers Authority, [62]) ha definito nella RFC 3589 l'intervallo di Command Code utilizzabile per applicazioni 3GPP come [300,. . . ,313]: nello specifico dell'applicazione Diameter per l'interfaccia Sh, si tratta dei quattro Command Code 306, 307, 308, 309, indicati nella Tabella 7.3.

Categoria	Intervallo di codici	Descrizione
Informal	1000-1999	Errori appartenenti a questa categoria informano il mittente che la richiesta non è andata a buon fine in quanto richiede ulteriori operazioni
Success	2000-2999	La richiesta è andata a buon fine
Protocol Errors	3000-3999	Errori di protocollo
Transient Failures	4000-4999	La richiesta non è andata a buon fine quando è stata ricevuta, ma potrebbe essere elaborata successivamente
Permanent Failures	5000-5999	La richiesta non è andata a buon fine e non dovrebbe essere ritrasmessa senza delle modifiche (Es.: mancanza di un AVP obbligatorio)

Tabella 7.2: Codici di errore Diameter

Command Name	Abbreviazione	Codice
User-Data-Request	UDR	306
User-Data-Answer	UDA	306
Profile-Update-Request	PUR	307
Profile-Update-Answer	PUA	307
Subscribe-Notification-Request	SNR	308
Subscribe-Notification-Answer	SNA	308
Push-Notification-Request	PNR	309
Push-Notification-Answer	PNA	309

Tabella 7.3: Command Code relativi all'interfaccia Sh di IMS

Sempre da specifiche IANA, i messaggi Diameter relativi all'interfaccia Sh dovranno contenere l'AVP `Vendor-Id` con il valore `10415` (corrispondente a 3GPP) e l'AVP `Application-Id` con il valore `16777217` (che identifica l'applicazione Diameter per l'interfaccia Sh). Le operazioni effettuabili attraverso l'interfaccia Sh possono essere raggruppate in quattro categorie:

- **Sh-Pull** o **Data read**: procedure invocate dall'AS e utilizzate per effettuare la lettura di dati relativi ad un determinato utente presenti all'interno dell'HSS. Appartengono a questa categoria le operazioni `User-Data-Request` (UDR) e `User-Data-Answer` (UDA). Il messaggio `User-Data-Request` deve contenere **obbligatoriamente** gli AVP indicati nella Tabella 7.4 come tipo M (Mandatory - obbliga-

tori) mentre può contenere gli AVP indicati nella tabella come tipo C (Conditional - a condizione) a seconda del tipo di richiesta.

Nome	AVP	Tipo	Descrizione
User Identity	`User-Identity`	M	Public User Identity IMS, Public Service Identity dell'utente cui vengono richiesti i dati
Requested Data	`Data-Reference`	M	Indica che informazione è richiesta
Application Server Identity	`Origin-Host`	M	Indica l'AS che origina la richiesta, e viene utilizzato per controllare i permessi di accesso all'HSS
Requested Domain	`Requested-Domain`	C	Indica il dominio al quale è applicabile l'operazione richiesta
Service-Indication	`Service-Indication`	C	Identifica, insieme alla User Identity inclusa nell'`User-Identity` AVP, l'insieme di dati relativi a un servizio che vengono richiesti
Application Server Name	`Server-Name`	C	Usato come chiave per identificare i Filter Criteria, insieme alla User Identity inclusa nell'`User-Identity` AVP e all'AVP `Data-Reference`
DSAI Tag	`DSAI-Tag`	C	Usato come chiave per identificare l'istanza di Dynamic Service Activation Info (DSAI) richiesta, insieme alla User Identity inclusa nell'`User-Identity` AVP e all'AVP `Data-Reference`

Tabella 7.4: AVP presenti nella richieste di tipo Sh-Pull

Per quanto riguarda il corrispondente messaggio di risposta `User-Data-Answer` (UDA), devono essere inseriti gli AVP indicati nella Tabella 7.5.

Nome	AVP	Tipo	Descrizione
Result	`Result-Code`	M	Risultato della richiesta. L'AVP Result-Code deve essere usato per gli errori definiti nel protocollo Diameter di base. L'AVP Experimental-Result deve essere usato per gli errori dell'estensione Diameter per l'interfaccia Sh
Data	`User-Data`	C	I dati richiesti dall'AS. Questo elemento deve essere presente qualora i dati richiesti sono presenti nell'HSS e l'AS ha il permesso di accedervi in lettura

Tabella 7.5: AVP presenti nelle risposte di tipo Sh-Pull

- **Sh-Update** o **Data-Update**: procedure invocate dall'AS utilizzate per consentire ad esso di aggiornare i dati presenti nell'HSS. Appartengono a questa categoria le operazioni `Profile-Update-Request` (PUR) e `Profile-Update-Answer` (PUA). Gli AVP da inserire e le relative descrizioni sono indicate nella Tabella 7.6.

Per quanto riguarda il corrispondente messaggio di risposta `Profile-Update-Answer` (PUA), deve essere inserito l'AVP indicato nella Tabella 7.7.

- **Subscription to notifications** o **Sh-Subs-Notif**: procedure invocate dall'AS e utilizzate per sottoscrivere la ricezione di notifiche di aggiornamento dei dati presenti nell'HSS. Appartengono a questa

Nome	AVP	Tipo	Descrizione
User Identity	`User-Identity`	M	La Public User Identity o Public Service Identity della quale si vogliono aggiornare i dati
Requested Data	`Data-Reference`	M	Indica che informazione va aggiornata
Data	`User-Data`	M	Contiene i dati che vanno aggiornati
Application Server Identity	`Origin-Host`	M	Identifica l'AS che origina la richiesta e viene utilizzato per controllare i permessi di accesso all'HSS

Tabella 7.6: AVP presenti nelle richieste di tipo Sh-Update

Nome	AVP	Tipo	Descrizione
Result	`Result-Code`	M	Risultato della richiesta. L'AVP Result-Code deve essere usato per gli errori definiti nel protocollo Diameter di base. L'AVP Experimental-Result deve essere usato per gli errori dell'estensione Diameter per l'interfaccia Sh

Tabella 7.7: AVP presenti nelle risposte di tipo Sh-Update

categoria le operazioni `Subscribe-Notifications-Request` (SNR) e `Subscribe-Notifications-Answer` (SNA). Tali operazioni non verranno tuttavia utilizzate nella trattazione seguente.

- **Notifications** o **Sh-Notif**: procedure invocate dall'HSS e utilizzate per informare l'AS di aggiornamenti avvenuti sui dati per i quali l'AS aveva precedentemente effettuato la sottoscrizione (si veda il punto precedente). Appartengono a questa categoria le operazioni `Push-Notification-Request` (PNR) e `Push-Notification-Answer` (PNA). Anche queste operazioni non verranno utilizzate nel seguito.

Nella Tabella 7.8 è indicato il codice da inserire nell'AVP `Data-Reference` a seconda del tipo di dati richiesti o da aggiornare nell'HSS. Inoltre, ogni tipo di dato può essere interessato da una determinata categoria di procedure: ad esempio, gli *Initial Filter Criteria*, cui corrisponde il valore di `Data-Reference` `13`, potranno essere oggetto di richieste di lettura (Sh-Pull) o di richieste di notifica degli aggiornamenti (Sh-Subs-Notif), ma non potranno essere modificati dagli AS tramite operazioni di Sh-Update, quindi `Profile-Update-Request` (PUR).

Tipo di dato	**Codice**	**Operazioni consentite**
Repository Data	0	Sh-Pull, Sh-Update, Sh-Subs-Notif
IMS Public Identity	10	Sh-Pull, Sh-Subs-Notif
IMS User State	11	Sh-Pull, Sh-Subs-Notif
S-CSCF Name	12	Sh-Pull, Sh-Subs-Notif
Initial Filter Criteria	13	Sh-Pull, Sh-Subs-Notif
Location Information	14	Sh-Pull
User State	15	Sh-Pull
Charging Information	16	Sh-Pull, Sh-Subs-Notif
MSISDN	17	Sh-Pull
PSI Activation	18	Sh-Pull, Sh-Update, Sh-Subs-Notif
DSAI	19	Sh-Pull, Sh-Update, Sh-Subs-Notif

Tabella 7.8: Dati accessibili attraverso l'interfaccia Sh

7.2 Java Diameter Peer

JavaDiameterPeer (JDP) [63] è una implementazione Java open source dello stack Diameter sviluppata dal Fraunhofer Institute for Open Communication Systems (FOKUS), lo stesso ente che ha prodotto OpenIMSCore: su di essa si basa infatti il componente FhoSS, ed è quindi naturale un suo utilizzo per l'interazione tra gli Application Server e lo stesso Home Subscriber Server. Al momento, JDP supporta solamente la connessione tra peer, e quindi non consente meccanismi di autorizzazione, autenticazione e accounting, che comunque non sono necessari per un utilizzo sulle interfacce Cx/Dx/Sh di IMS. Inoltre, manca il supporto per il routing dei messaggi Diameter tra domini differenti, e questo significa che tutti i messaggi devono essere inviati indicando il Fully Qualified Domain Name (FQDN) dell'host di destinazione.

JDP crea un peer Diameter utilizzando un file di configurazione (in formato XML) nel quale vanno specificati il dominio, l'application ID e il FQDN del peer, come pure la porta sulla quale il nodo riceverà le richieste Diameter,

il vendor e application ID ed altre informazioni circa il comportamento del peer. La Figura 74 costituisce un esempio di un tale file di configurazione.

```
<?xml version="1.0" encoding="UTF-8"?>
 <DiameterPeer
  FQDN="localhost.open-ims.test"
  Realm="open-ims.org"
  Vendor_Id="10415"
  Product_Name="JavaDiameterPeer"
  AcceptUnknownPeers="1"
  DropUnknownOnDisconnect="1"
  Tc="10"
  Workers="8"
  QueueLength="32"
 >
  <Acceptor port="3878" bind="127.0.0.1" />
  <Auth id="16777217" vendor="10415"/>
 </DiameterPeer>
```

Le informazioni rilevanti sono :

- **FQDN**: il Fully Qualified Domain Name dell'host che costituirà il Diameter Peer;
- **Realm**: il nome del dominio cui appartiene il Diameter Peer;
- **Vendor_Id**: il valore `10415` corrisponde all'ente 3GPP;
- **Acceptor port**: indica la porta sulla quale il Diameter Peer resterà in ascolto ;
- **Auth id**: il valore `16777217` identifica l'interfaccia Sh di IMS.

Un tale Diameter Peer rappresenta quindi un nodo che implementa il protocollo Diameter, e può operare come Client o come Server. Ogni Peer dispone poi di un componente detto *Communicator* che si occupa di mantenere le connessioni Diameter con gli altri host: alla ricezione di un messaggio, il *Communicator* lo inserisce in una *TaskQueue*, ovvero una coda FIFO che rappresenta la lista di task da eseguire. Un altro componente, il *DiameterWorker*, ha il compito di estrarre il corrispondente messaggio Diameter e sottoporlo ad un *EventListener*, cioè la parte del Diameter Peer che si occupa di elaborare i messaggi ricevuti.

Per quanto riguarda invece le operazioni di invio, il mittente fornisce il messaggio ad un *PeerManager*, che si occupa di individuare il Diameter Peer destinatario ed inoltrare ad esso il messaggio.

7.2.1 Recupero di informazioni dall'HSS (Sh-Pull)

Come descritto nel paragrafo 7.1.2, una delle operazioni consentite dall'applicazione Diameter per l'interfaccia Sh è il recupero delle informazioni salvate nell'HSS relative al profilo di un utente: il messaggio Diameter corrispondente è chiamato `User-Data-Request` (UDR) ed è identificato dal Command Code `306`. Ogni messaggio di questo tipo deve contenere, oltre agli AVP obbligatori prescritti dal protocollo Diameter base, anche quelli indicati nella Tabella 7.4, ovvero:

- `User-Identity`: indica l'identità (Public User Identity) dell'utente del quale vogliamo richiedere informazioni. Si tratta quindi di una SIP URI del tipo `sip:nomeutente@dominio`;
- `Data-Reference`: indica il tipo di dati che si vogliono recuperare. Ogni profilo utente salvato nell'HSS è infatti composto da vari campi (si veda la Tabella 7.8) e questo AVP deve contenere un numero che lo identifichi. Se ad esempio si volessero recuperare le informazioni relative allo stato di registrazione dell'utente, si dovrà indicare il codice 15;
- `Origin-Host`: è il nome del server (così come noto all'HSS) che effettua la richiesta dei dati. L'HSS utilizza questo nome per verificare che il server in questione disponga dei permessi necessari per accedere alle informazioni degli utenti.

A seconda dei dati richiesti (quindi del valore dell'AVP `Data-Reference`) il messaggio Diameter potrà contenere altri AVP, oltre ai precedenti, affinché la richiesta venga accettata dal destinatario (in questo caso l'HSS). L'insieme di questi AVP viene chiamato 'chiave d'accesso' nelle specifiche TS 29.328: la Tabella 7.9 riporta i casi d'uso principali.

La Figura 7.2 evidenzia i canali di comunicazione esistenti tra l'AS, l'HSS e il core della rete IMS: come si vede, oltre all'interfaccia Sh, l'HSS dispone di altre connessioni Diameter con il S-CSCF e l'I-CSCF; l'interfaccia Cx rappresenta questi collegamenti, e le operazioni effettuabili attraverso di essa sono specificate nel documento TS 29.228. Le principali riguardano l'aggiornamento dei dati riguardanti il profilo degli utenti, che possono subire delle variazioni ad ogni registrazione degli stessi (operazione, quest'ultima, notificata all'HSS da parte del S-CSCF).

```
1 <?xml version="1.0" encoding="UTF-8"?>
2  <Sh-Data>
3   <Sh-IMS-Data>
4    <IMSUserState>2</IMSUserState>
5   </Sh-IMS-Data>
6  </Sh-Data>
```

Data Ref.	Significato	Chiave d'accesso
11	IMS User State	IMS Public User Identity, Data-Reference
12	S-CSCF Name	IMS Public User Identity o Public Service Identity, Data-Reference
13	Initial Filter Criteria	IMS Public User Identity o Public Service Identity, Data-Reference, Server-Name
19	DSAI	IMS Public User Identity o Public Service Identity, Data-Reference, DSAI-Tag, Server-Name

Tabella 7.9: Principali chiavi d'accesso per i valori salvati nell'HSS

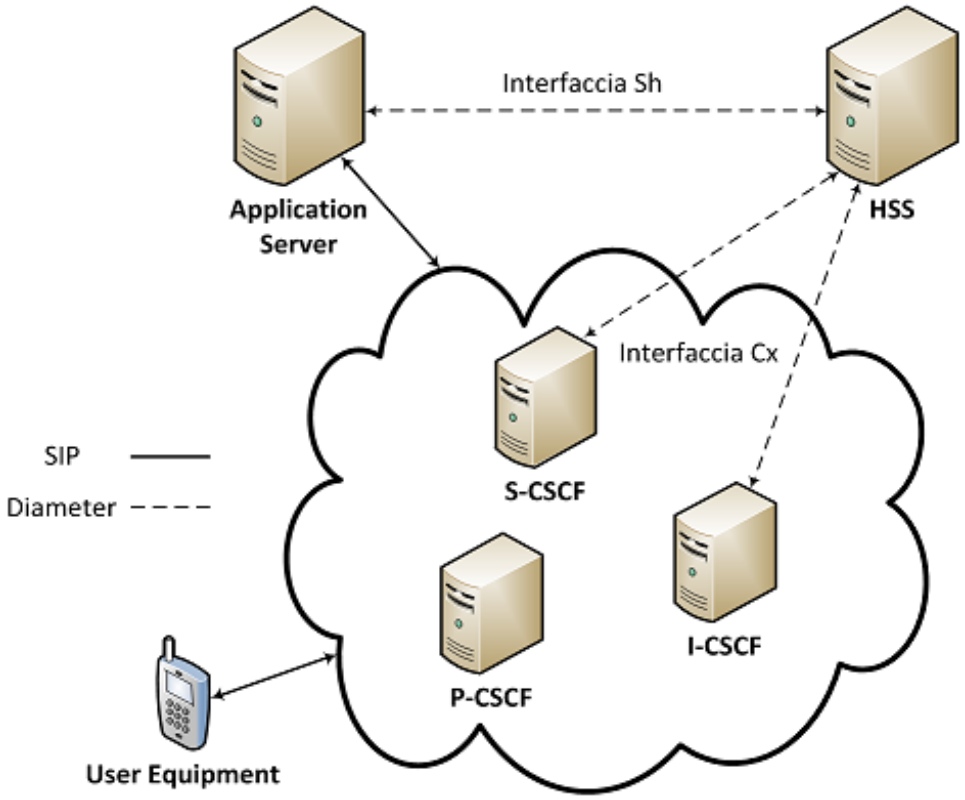


Figura 7.2: Comunicazioni Diameter e SIP tra AS, HSS e CSCF

7.3 Dettagli implementativi

In questa sezione verranno descritte nel dettaglio le caratteristiche del Service Proxy sviluppato per poter interagire con la rete IMS: tre classi Java costituiscono il servizio vero e proprio, mentre altre due sono necessarie per poter utilizzare il framework OSGi [65], ovvero l'ambiente di esecuzione che permette di aggiungere e rimuovere dinamicamente servizi (chiamati *bundle*) nonché interagire con l'orchestratore.

Le prime tre classi sono dunque:

- `SipLayerThread`: è la classe che implementa l'interfaccia `Runnable` e viene dunque eseguita come thread. Essa contiene principalmente la logica che corrisponde alle elaborazioni svolte dal particolare servizio;
- `SipStackImplem`: costituisce l'implementazione dello stack Sip. Il suo compito è la gestione delle richieste Sip provenienti dalla rete IMS;
- `DiameterPeerImpl`: rappresenta il Diameter Peer utilizzato per poter ricevere ed eventualmente modificare le informazioni sul profilo dell'utente salvate nell'HSS.

Per quanto riguarda invece le classi necessarie all'interazione con OSGi, sono state definite:

- `SipSP`: contiene i metodi necessari all'inizializzazione del *bundle* OSGi che rappresenta il Service Proxy Sip, nonché la gestione degli eventi provenienti/diretti da/verso l'orchestratore;
- `Activator`: classe necessaria al framework OSGi per poter gestire l'avvio/interruzione del servizio.

7.3.1 Il framework OSGi

Il framework OSGi[1] [65] si propone di implementare un modello a componenti completo e dinamico per l'ambiente Java attraverso la definizione di alcuni concetti fondamentali:

- la definizione di *bundle*;
- la gestione automatica delle dipendenze;
- la gestione del ciclo di vita del codice.

OSGi è quindi un sistema dinamico che consente l'avvio, l'interruzione e la rimozione di moduli in fase runtime, senza quindi la necessità di riavvii: queste caratteristiche sono desiderabili quando si ha a che fare con

[1] Nel seguito faremo sempre riferimento all'implementazione delle specifiche OSGi sviluppata da Eclipse, che prende il nome di Equinox [66].

un'architettura orientata ai servizi, ed in particolare con quella descritta nel paragrafo 6.3.3, presa come riferimento nello sviluppo del Service Proxy Sip.

Nel contesto di OSGi, il Service Proxy sviluppato equivale ad un *bundle*, cioè il modulo unitario che definisce esplicitamente sia le dipendenze con eventuali altri moduli/servizi, che le API esposte. Un *bundle* non è altro che un file `.jar`, nel quale oltre alla logica del servizio sono contenute informazioni addizionali all'interno del file `MANIFEST.MF`, come ad esempio il nome del *bundle* e le sue dipendenze. Inoltre, ogni modulo ha la possibilità di interagire con gli altri installati mediante l'utilizzo della classe `BundleContext`, resa disponibile dal framework in fase di inizializzazione del servizio.

Volendo fare nuovamente riferimento all'architettura presentata nel paragrafo 6.3.2, sia l'orchestratore che i servizi resi disponibili (installati o attivi) saranno costituiti da *bundle*: per poter funzionare correttamente, essi conterranno una classe `Activator`, necessaria per l'avvio/interruzione del servizio, e una classe contenente la logica vera e propria (che può comunque avere delle dipendenze). Nel caso del Service Proxy Sip, quest'ultima classe prende il nome di `SipSP`.

7.3.2 La classe `SipStackImplem`

Come già accennato, lo scopo della classe `SipStackImplem` è quello di gestire l'arrivo dei messaggi Sip provenienti dalla rete IMS e la loro successiva reimmissione all'interno della rete una volta concluse le operazioni di elaborazione. Questa classe fa largo uso della libreria JAIN Sip, che offre al programmatore molti strumenti utili ad eseguire le procedure più di basso livello prescritte dal protocollo Sip.

Il costruttore della classe riceve come parametri espliciti sia l'indirizzo IP che il numero della porta sulla quale lo stack Sip resterà in ascolto: dal punto di vista della rete IMS questo non è altro che un Application Server. Inoltre al suo interno verrà inizializzata una tabella hash, contenente coppie <attributo,valore>, che corrisponde a quella definita nel paragrafo 6.3.3 per il riconoscimento dei messaggi Sip in arrivo.

Fondamentale è il metodo `processRequest`: il suo scopo è quello di ricevere i messaggi Sip di **richiesta** provenienti dalla rete IMS, estrarre alcune informazioni necessarie al servizio, ed invocare la creazione di un nuovo thread, che si occuperà di effettuare quanto specificato dalla logica del servizio.

```
1 public void processRequest(RequestEvent requestEvent) {
2   Request r = requestEvent.getRequest();
3   CallIdHeader callIdHeader =
4             (CallIdHeader) r.getHeader("Call-ID");
5   String callId = callIdHeader.getCallId();
6   Thread t = new Thread(new SipLayerThread(callId,
7                         this.hash, requestEvent, this));
```

```
8     t.run();
9   }
```

Come si vede, nella creazione di un nuovo thread corrispondente alla richiesta appena ricevuta, viene inserito come parametro esplicito anche l'istanza dell'oggetto `SipStackImplem`, così da consentire al servizio l'accesso alla tabella hash che risulterà dunque essere condivisa tra tutti i thread. Il metodo `processResponse` riceve invece i messaggi Sip di **risposta**: poichè il Service Proxy è trasparente a questo tipo di messaggi, il metodo si limiterà ad inoltrarli secondo quanto specificato nell'header.

Infine, all'interno della classe `SipStackImplem` è presente una classe `protected` chiamata `Prop`: saranno di questo tipo gli oggetti costituenti i campi 'valore' all'interno della tabella hash definita precedentemente, e conterranno informazioni quali la sessione di orchestrazione, il callID, o qualsiasi altro dato di interesse per il Service Proxy.

7.3.3 La classe `SipLayerThread`

La classe `SipLayerThread`, che implementa l'interfaccia `Runnable`, è responsabile delle elaborazioni svolte dal servizio: nel paragrafo precedente s'è visto come una richiesta ricevuta dallo stack Sip venga intercettata dal metodo `processRequest`, che a sua volta si occupa di creare il thread corrispondente; questa classe rappresenta appunto tale thread. Al suo interno è quindi presente il metodo `run()`, che contiene le istruzioni relative a quanto deve essere svolto dal servizio: nel nostro caso, esso si occuperà di inoltrare la richista Sip al destinatario solo se quest'ultimo ha sottoscritto il servizio, ovvero se è stata inserita (a seguito della chiamata `invokeAction`) la entry corrispondente al suo indirizzo Sip nella tabella hash. Possono dunque verificarsi quattro situazioni:

1. viene ricevuto un messaggio `SIP INVITE` e l'indirizzo nel campo `To` è presente nella tabella hash;

2. viene ricevuto un messaggio `SIP INVITE` con un indirizzo nel campo `To` che è già transitato una volta[2];

3. viene ricevuto un messaggio `SIP BYE` con un campo `To` o `From` già presente nella tabella hash;

4. viene ricevuto un messaggio che non rientra in nessuno dei casi elencati precedentemente.

[2]La distinzione tra i punti 1. e 2. è resa possibile dalla presenza, nell'oggetto di tipo `Prop` che costituisce il valore di una coppia presente nella tabella hash, di un campo contenente una flag impostata al valore `true` dopo il primo ricevimento di un messaggio Sip corrispondente all'indirizzo `To` associato.

Ad ogni punto corrisponderanno elaborazioni diverse, quindi considerando ad esempio il caso 1:

```
1  if(method.equals("INVITE")&&!hash.get(toHeader).getFlag()){
2    Response response=messageFactory.createResponse(100,req);
3    sipProvider.sendResponse(response);
4    synchronized(hash){
5      SipStackImplem.Prop prop=sipStack.hash.get(toHeader);
6      prop.setFlag(true);
7      prop.setCallId(callidHeader);
8      hash.put(toHeader, prop);
9      }
10 }
```

Il codice presente nel corpo dell'`if` verrà eseguito solo se il metodo indicato nella richiesta ricevuta è `INVITE` e se la flag presente nella tabella hash in corrispondenza al valore `To` (ovvero `toHeader`) è impostata al valore `false`, ovvero questa è la prima volta che viene ricevuto un messaggio `INVITE` destinato a tale utente.

Nel caso in cui le condizioni vengano soddisfatte, si procede con la creazione e l'invio di una risposta Sip di tipo `100 Trying`, come specificato nel documento RFC 3261. Dopodiché, acquisendo il lock sulla risorsa `hash`, vengono aggiornati i valori della flag corrispondente all'indirizzo del destinatario nonché aggiunta l'informazione relativa al `callID`.

Bisogna comunque sottolineare che, oltre alla logica strettamente collegata al particolare servizio implementato, è necessario inserire in questa classe tutte le operazioni relative alla segnalazione Sip prescritte dal protocollo; in particolare, a seconda del tipo di header, l'indirizzo dello stack Sip va:

- rimosso dall'header `Route`;
- inserito nell'header `Via`;
- inserito nell'header `Record-Route`.

In particolare, l'ultima operazione è necessaria anche per il corretto funzionamento del servizio: il Service Proxy deve infatti poter ricevere le successive richieste relative al dialogo in corso, nella fattispecie la richiesta di `SIP BYE`, alla quale farà seguito la rimozione dell'entry corrispondente nella tabella hash (si considera cioè terminata l'esecuzione del servizio).

7.3.4 La classe `SipSP`

Come si diceva, la classe `SipSP` deve gestire l'interazione tra il Service Proxy e l'orchestratore, quindi principalmente:

1. inizializzare il Service Proxy;

2. estrarre le *input properties* specifiche del Service Proxy (contenute nel parametro `properties`);

3. comunicare con la risorsa che si gestisce, la quale genererà degli input e/o degli eventi;

4. creare le `properties` da inserire nell'invocazione del metodo `notifyEvent`.

Il primo punto viene realizzato attraverso il metodo `initializeBaseService`[3]:

```
boolean initializeBaseService(Orchestrator orchestrator) {
    stack = new SipStackImplem("user", "localhost", 9060);
    orch = orchestrator;
    return true;
}
```

In questo caso l'operazione di inizializzazione consiste nel creare un'istanza dello stack Sip (che comunque sarà l'unica utilizzata durante l'esecuzione del servizio) e inizializzare la variabile d'esemplare `orch` al valore specificato nel parametro esplicito del metodo.

I punti 2, 3 e 4 vengono invece realizzati a seguito dell'invocazione del metodo `invokeAction`: al suo interno viene creato un oggetto di tipo `Sip`, classe privata che estende la classe `Thread` e dispone quindi di un metodo `run()`. Ogni qual volta l'orchestratore richiami il metodo `invokeAction` in un Service Proxy, esso fornisce un vettore di coppie <Attributo,Valore> (appunto l'oggetto `properties`) che hanno il formato:

```
<ServiceProxyName+"."+ServiceProxyInstance+"."+ProperyName, Value>
```

Poichè l'oggetto `properties` è un vettore che conterrà le *input properties* di altri servizi oltre a quello invocato, è compito di quest'ultimo estrarre l'informazione per lui rilevante:

```
for(int i=0; i<props.length; i++){
    if(props[i].getPropertyName().equals("SipSP."+bsInstance
        +".sipTO"))
    toField = props[i].getPropertyValue();
    }
stack.addEntry(toField, csID);
```

Nel codice sopra sono presenti le istruzioni utilizzate dal Service Proxy Sip: il campo di interesse è quello contrassegnato dal nome del Service Proxy Sip, ovvero `SipSP`, dalla particolare istanza del servizio (specificata nell'oggetto `bsInstance`) nonché dal campo *PropertyName* corrispondente a `SipTO` (dal momento che interessa conoscere il destinatario della segnalazione Sip, ovvero il sottoscrittore del servizio). Infine, l'istruzione

[3]Per *Base Service* si intende un modulo che rappresenta un servizio nel modello cui si fa riferimento; un *Service Proxy* è invece l'agente software che contiene la logica del servizio. In ogni caso, i due termini potranno essere adoperati in maniera interscambiabile.

`stack.addEntry(toField, csID)` serve ad inserire il valore dell'indirizzo `To` all'interno della tabella hash corrispondente al servizio, con in aggiunta l'indicazione della sessione di orchestrazione.

A questo punto il Service Proxy Sip dispone di tutte le informazioni necessarie al suo corretto funzionamento nel contesto di una rete IMS: esso può aggiornare il profilo relativo all'utente specificato nel campo `To` e presente nell'HSS attraverso uno scambio di messaggi Diameter. Queste operazioni vengono svolte dalla classe `DiameterPeerImpl`, della quale segue la descrizione.

7.3.5 La classe `DiameterPeerImpl`

La classe `DiameterPeerImpl` viene utilizzata dal Service Proxy Sip per poter comunicare con l'HSS, attraverso lo scambio di messaggi Diameter: essa si basa sull'implementazione dello stack Diameter realizzata da FOKUS e descritta nel paragrafo 7.2.

Come specificato nel paragrafo 7.1.2, relativo al funzionamento dell'applicazione Diameter per l'interfaccia Sh, un Application Server **non** può modificare direttamente il valore degli *initial Filter Criteria* associati ad un utente, bensì può attivare o disattivere i relativi DSAI. Nella realizzazione del Service Proxy abbiamo considerato la presenza nell'HSS di un profilo DSAI attivabile per l'utente specificato nel campo `To`, cui sono collegati degli *initial Filter Criteria* che impongono l'invio dei messaggi `SIP INVITE` direttamente al Service Proxy Sip (ovvero all'indirizzo dello stack Sip).

La creazione del messaggio Diameter `User-Update-Request` (UDR), avviene all'interno del metodo `UDR`, che restituisce il messaggio così creato; il metodo, oltre all'istanza di Peer Diameter, richiede il parametro esplicito `UserIdentity`, ovvero una stringa che identifica l'utente del quale si intendono aggiornare i dati nell'HSS. Nel metodo vengono creati gli AVP prescritti dal tipo di richiesta: a titolo d'esempio riportiamo il codice relativo alla creazione dell'AVP `User-Identity`:

```
1 a = new AVP(700,true,10415);
2 b = new AVP(601,true,10415);
3    b.setData(userIdentity);
4    a.addChildAVP(b);
5 udr.addAVP(a);
```

dove il codice `700` identifica l'AVP `User-Identity`, il codice `601` l'AVP `Public-User-Identity` e infine l'oggetto `udr` rappresenta il messaggio Diameter che verrà inviato all'HSS.

Dopo la creazione del messaggio, questo verrà inviato all'HSS e nel contempo il Peer Diameter resterà in attesa di una risposta: entrambe le operazioni sono presenti all'interno del metodo `start`, che viene invocato all'interno della classe `SipSP` a seguito della ricezione dell'`invokeAction` da parte dell'orchestratore.

```
String hssAddr = "hss.open-ims.test";
DiameterMessage req = PUR(dp1, userIdentity);
DiameterMessage answ=dp1.sendRequestBlocking(hssAddr,req);
```

7.3.6 La classe `Activator`

La classe `Activator` effettua le operazioni necessarie all'avvio/interruzione del servizio all'interno del framework OSGi. In particolare, essa si occupa di creare i collegamenti tra l'orchestratore e il servizio che si sta attivando, mediante:

1. il passaggio del riferimento del servizio Orchestratore al Service Proxy in fase di attivazione;
2. la creazione della *Property* corrispondente al nome del servizio;
3. la registrazione del servizio all'interno del registro OSGi.

Queste operazioni sono svolte dal seguente codice, che opera sull'oggetto di tipo `BundleContext` che rappresenta il contesto di esecuzione per il framework OSGi:

```
1  public void start(BundleContext bundleContext) {
2      Activator.context = bundleContext;
3      ServiceReference reference = bundleContext.
           getServiceReference(Orchestrator.class.getName());
4      Orchestrator orchRef = (Orchestrator) bundleContext.
           getService(reference);
5      SipBS service = new SipBS();
6      service.initializeBaseService(orchRef);
7      Properties props = new Properties() ;
8      props.put("bsNAME","SipSP");
9      bundleContext.registerService(BSInterface.class.getName
           (),service,props);
10 }
```

In particolare, alla riga 3 viene estratto il riferimento al servizio di orchestrazione, che viene poi passato come parametro esplicito al servizio in fase di avvio alla riga 6. Alla riga 8 viene invece aggiunta la Property, di tipo `bsNAME`, corrispondente al nome del Service Proxy attivato, in questo caso `SipSP`. Infine, l'istruzione alla riga 9 effettua la registrazione del servizio nel registro di OSGi.

Capitolo 8

Conclusioni e sviluppi futuri

In questa tesi sono state analizzate le caratteristiche dello standard IMS, a partire dalle quali è stato possibile formulare più soluzioni al problema della composizione di servizi in questo contesto. In particolare, i modelli proposti nel Capitolo 6 hanno fatto uso di tecnologie differenti: in primis, come suggerito da Ericsson Composition Engine, le Servlet SIP, che costituiscono uno standard ampiamente utilizzato e che, grazie all'introduzione del componente Application Router nella versione 1.1, hanno dimostrato di essere adatte alla composizione di servizi telefonici. Tuttavia, la rigidità imposta da tali specifiche non consente di andare molto oltre una semplice composizione **statica** di servizi, sebbene la creazione di un Application Router che svolga tali operazioni è semplice, e non richieda allo sviluppatore di servizi (ovvero Servlet) di conoscerne il funzionamento.

Ma proprio l'impossibilità di utilizzare la tecnologia SIP Servlet per poter creare una composizione di servizi dinamica, ha spinto a considerare altre tecnologie e a discostarsi leggermente dalla soluzione proposta da Ericsson: l'utilizzo di librerie che implementino lo stack SIP richiede una maggiore attenzione nei confronti dei dettagli più di basso livello del protocollo, ma nonostante ciò, la piattaforma proprietaria presentata nel Capitolo 6 consente di sviluppare una volta per tutte la logica necessaria all'interazione con la rete IMS, che può quindi essere utilizzata con limitate modifiche nella creazione di qualsiasi servizio interagente con tale rete. Questo è possibile mediante dall'invocazione dei metodi `notifyEvent` e `invokeAction`, resi disponibili dall'orchestratore e attraverso i quali è attuabile lo scambio di informazioni (le cosiddette *properties*) utili alla realizzazione di una composizione **dinamica** di servizi.

Il *Service Proxy SIP* descritto nel Capitolo 7 costituisce un chiaro esempio di componente software in grado di porsi a metà tra l'orchestratore e il S-CSCF, avente le caratteristiche di modularità richieste dal paradigma *Event-Driven* nonché la possibilità di un riutilizzo del codice secondo quan-

to specificato nella Service Oriented Architecture (SOA). In questo senso può essere considerato come realizzazione di quanto definito da 3GPP nelle specifiche di IMS in merito al componente Service Broker, per il quale nel documento non si è andati oltre la semplice descrizione funzionale ad alto livello.

Questo lavoro lascia invece aperto il problema della modifica dinamica, da parte degli Application Server, dei parametri riguardanti gli *initial Filter Criteria*, salvati nell'HSS: poichè le specifiche dell'applicazione Diameter per l'interfaccia Sh non consentono un aggiornamento di tali informazioni, si è dovuto ricorrere all'attivazione/disattivazione di un apposito DSAI. Questo, tuttavia, deve essere configurato prima dell'attivazione del servizio composito e per ogni utente che ne intenderà sottoscrivere le funzionalità: appare chiaro come questo sistema sia inadatto ad un utilizzo su larga scala, perlomeno senza un adeguato meccanismo automatico per la configurazione. Sviluppi futuri dovrebbero quindi concentrarsi sulla risoluzione di questo problema.

Come detto in precedenza, una delle caratteristiche del *Service proxy Sip* è la possibilità di essere utilizzato per implementare servizi differenti senza dover modificare grandi porzioni di codice, e anzi conservare la base di logica necessaria all'interazione con la rete IMS. Sempre nell'ottica di riutilizzo del codice, uno sviluppo futuro potrebbe consistere nella creazione di una libreria che offra dei metodi standard per poter operare con le componenti interne di IMS, in particolare il S-CSCF e l'HSS: in questo modo lo sviluppatore di servizi potrà, ad esempio, richiedere il profilo utente salvato nell'HSS attraverso la semplice invocazione di un metodo, oppure elaborare direttamente un messaggio `SIP INVITE` destinato allo specifico Application Server.

Infine, ha senso effettuare una serie di test quantitativi destinati a confrontare le performance dei diversi modelli proposti, soprattutto in un contesto di esecuzione il più possibile vicino a quello di uno scenario di utilizzo reale: sulla base di queste considerazioni, sarà possibile formulare il modello di gestione dei thread ottimale, aspetto ancora non esaminato all'interno della tesi.

Bibliografia

[1] K., N. Morita, and T. Towle Knightson, 'NGN architecture: Generic principles, functional architecture, and implementation', *IEEE Communications Magazine*, vol. 43, pp. 49-56, 2005.

[2] International Telecommunication Union. Online: `http://www.itu.int`

[3] A. Cuevas, J.I. Moreno, P. Vidales, and H. Einsiedler, 'The IMS Service Platform: a solution for next-generation network operators to be more than bit pipes', *IEEE Communications Magazine*, pp. 75-81, Agosto 2006.

[4] International Telecommunication Union - NGN definition. Online: `http://www.itu.int/ITU-T/studygroups/com13/ngn2004/working_definition.html`

[5] H. Schulzrinne, E. Schooler, J. Rosenberg M. Handley, 'SIP: Session Initiation Protocol', IETF, RFC 3261, 2002.

[6] Telecoms and Internet converged Services & Protocols for Advanced Networks (TISPAN). Online: `http://www.etsi.org/tispan/`

[7] Internet Engineering Task Force. Online: `http://www.ietf.org/`

[8] TeleManagement Forum - Service Delivery Platform. Online: `http://www.tmforum.org/ServiceDeliveryPlatforms/7284/home.html`

[9] M. Jackson, P. Zave, 'Distributed Feature Composition: A Virtual Architecture for Telecommunications Services', *IEEE Transaction on Software Engineering, Vol. 24, No. 10, Ottobre 1998*

[10] Yahoo! Pipes. Online: `http://pipes.yahoo.com`

[11] JackBe Presto. Online: `http://www.jackbe.com`

[12] IBM Mashup Center. Online: `http://www.ibm.com/software/info/mashup-center/`

[13] Jörg Niemöller et al., 'Composition in Converged Service Networks: Requirements and Solutions', *Ericsson Eurolab, Herzogenrath, Germania, Ericsson Corporate Research, Service Networks Solutions*, 2010.

[14] JAIN SIP. Online: `https://jain-sip.dev.java.net/`

[15] Java Specification Requests (JSR) - JSR 289: SIP Servlet 1.1. Online: `http://jcp.org/en/jsr/detail?id=289`

[16] 3GPP LTE Homepage. Online: `http://www.3gpp.org/article/lte`

[17] OpenEPC - Open Evolved Packet Core. Online: `http://www.openepc.net/en/openepc/index.html`

[18] Gilles Bertrand, 'The IP Multimedia Subsystem in Next Generation Networks', *Network, Multimedia and Security department (RSM) - GET/ENST Bretagne*, 2007.

[19] 3GPP Specification 23.228 - IP Multimedia Subsystem. Online: `http://www.3gpp.org/ftp/Specs/html-info/23228.htm`

[20] 3GPP ETSI, 'ETSI ES 201 915-1 V1.4.1: Open Service Access (OSA), Application Programming Interface (API) - Overview', *ETSI Tech. Rep.*, 2005.

[21] 3GPP, '3GPP TS 29.278: CAMEL Application Part (CAP) specification for IP Multimedia Subsystems (IMS)', *3GPP Tech. Rep.*, 2009.

[22] J. Klensin, 'SMTP: Simple Mail Transfer Protocol', *IETF, RFC 2821*, 2001.

[23] J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Nerners-Lee R. Fielding, 'Hypertext Transfer Protocol - HTTP/1.1', IETF, RFC 2616, 1999.

[24] V. Jacobson M. Handley, 'SDP: Session Description Protocol', IETF, RFC 2327, 1998.

[25] J. Loughney, E. Guttman, G. Zorn, J. Arkko P. Calhoun, 'Diameter Base Protocol', IETF, RFC 3588, 2003.

[26] H. Hakala et al., 'Diameter Credit-Control Application', IETF, RFC 4006, 2005.

[27] IETF ITU-T, 'H.248: Gateway control protocol: version 3', ITU-T Recommendation 2005.

[28] Miguel A. Garcia-Martin Gonzalo Camarillo, 'The 3G IP Multimedia Subsystem (IMS): merging the Internet and the cellular worlds', *John Wiley & Sons Ltd.*, 2004.

[29] 3GPP Specification TS 23.218, 'IP Multimedia (IM) session handling; IM call model; stage 2'. Online: `http://www.3gpp.org/ftp/Specs/html-info/23218.htm`

[30] Ericsson IMS Common System. Online: `http://www.ericsson.com`

[31] IMS Advanced Research Cluster for Services. Online: `http://www.ims-arcs.org`

[32] The UCT IMS client. Online: `http://uctimsclient.berlios.de`

[33] The IMS communicator. Online: `http://imscommunicator.berlios.de`

[34] K. Knuettel, T. Magedanz N. Blum, 'Convergence in Services, Media and Services - Basic Requirements for Virtual Network Operators', *International Conference on Intelligence in Networks (ICIN)*, Bordeaux, France, 2006, pp. 265-270.

[35] H. Lundgren et al., 'Experiences from the design, deployment and usage of the UCSB MeshNet testbed', *IEEE Wireless Communications, pp. 18-29*, Aprile 2006.

[36] SIP Express Router (SER). Online: `http://www.iptel.org/ser`

[37] myMONSTER - Telco Communicator Suite. Online: `[40http://www.monster-the-client.org`

[38] Jean-Charles Grégoire Hechmi Khlifi, 'IMS Application Servers: roles, requirements, and implementation technologies', *IEEE Internet Computing, pp. 40-51*, Maggio/Giugno 2008.

[39] ETSI Open Service Access (OSA). Online: `http://etsi.org/WebSite/Technologies/OSA.aspx`

[40] E. Henrikson, D. Mills M. Garcia Martin, 'Private Header (P-Header) Extensions to the Session InitiationProtocol (SIP) for the 3rd-Generation Partnership Project (3GPP)', IETF, RFC 3455, 2003.

[41] SailFin Project. Online: `https://sailfin.dev.java.net/`

[42] GlassFish Open Source Application Server. Online: `https://glassfish.dev.java.net/`

[43] NetBeans IDE. Online: `http://netbeans.org`

[44] Manifesto SOA. Online: `http://www.soa-manifesto.org/`

[45] OASIS, Web Services Business Process Execution Language (WSBPEL), 2007. Online: `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel`

[46] Web Service Business Process Execution Language - versione 2.0. Online: `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf`

[47] OASIS, UDDI Specification TC, 2005. Online: `http://www.oasis-open.org/committees/tc_home`

[48] W3C, Web Services Description Language (WSDL) 1.1, 2001. Online: `http://www.w3.org/TR/wsdl`

[49] W3C, SOAP Version 1.2, 2007. Online: `http://www.w3.org/TR/soap/`

[50] G. Bond, E. Cheung, I. Fikouras, R. Levenshteyn, 'Unified Telecom and Web Services Composition: Problem Definition and Future Directions', *IPTComm*, 2009.

[51] Kristin F. Kocan et al., 'A novel software approach for service brokering in advanced service architectures', *Bell Labs Technical Journal, no. 11, pp. 5-20*, 2006.

[52] M. e E. H. Magill Kolberg, 'Handling incompatibilities between services deployed on IP-based networks', *IEEE Intelligent Networks, pp. 360-370*, 2001.

[53] D., H. Fujiwara, e T. Ohta Harada, 'Avoidance of feature interactions at run-time', *IEEE International Conference on Software Engineering Advances, pp. 6-12*, 2006.

[54] K. Y., e G. V. Bochmann. Chan, 'Methods for designing SIP services in SDL with fewer feature interactions', *Feature interaction in telecommunications, pp. 59-77*, 2003.

[55] Z., S. Cherkaoui, e A. Khoumsi Chentouf, 'Implementing online feature interaction detection in SIP environment', *IEEE International, pp. 515-521*, 2003.

[56] A. Khoumsi, 'Detection and resolution of interactions between services of telephone networks', *Feature interaction in telecommunications networks, pp. 78-92*, 1997.

[57] R. G., M. Carvalho, e L. Logrippo Crespo, 'Distributed resolution of feature interactions for internet applications', *Elsevier Computer Networks*, 2006.

[58] 3GPP Technical Report 23.810, 'Study on architecture impacts of service brokering'.

[59] M. Stecca, M. Maresca, 'A comparison between Web Service and JAVA Message Service technologies for Event-Driven Mashup execution', *Int. J. Web and Grid Services, vol. 6, no. 3, pp. 269-288*, 2010.

[60] 3GPP Technical Specification 29.328, 'IP Multimedia Subsystem Sh interface: signalling flows and message contents (Release 10)', 2011. Online: `http://www.3gpp.org/ftp/Specs/html-info/29328.htm`

[61] 3GPP Technical Specification 29.329, 'Sh interface based on Diameter protocol: protocol details', 2010. Online: `www.3gpp.org/ftp/Specs/html-info/29329.htm`

[62] IANA - Internet Assigned Numbers Authority. Online: `http://www.iana.org`

[63] JavaDiameterPeer (FOKUS). Online: `http://www.openimscore.org/project/jdp`

[64] 3GPP Specifications 24.229, 'IP multimedia call control protocol based on SIP and SDP'. Online: `http://www.3gpp.org/ftp/Specs/html-info/24229.htm`

[65] Open Service Gateway initiative (OSGi). Online: `http://www.osgi.org/`

[66] Eclipse Equinox. Online: `http://www.eclipse.org/equinox/`