UNIVERSITÀ DEGLI STUDI DI MILANO
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica

Sviluppo di un sistema per la gestione di identificatori universali per la comunicazione

Relatore: Prof.ssa Elisa Bertino
Correlatore: Dott. Marco Mesiti

Tesi di Laurea di
Galli Valeria
Matricola 536467

Anno Accademico 2000-2001

Indice

Elenco delle figure

Capitolo 1

Introduzione

Le tecnologie che si sono maggiormente sviluppate negli ultimi anni, a seguito di un enorme ampliamento del mercato dell'utenza, sono state sicuramente quelle dei sistemi informatici e delle reti legate ad Internet, da una parte, e della telefonia mobile (o *wireless*) dall'altra. Questo sviluppo ha spostato l'attenzione da *desktop Internet* a *mobile Internet* supportato da apparecchiature portatili. Pertanto i telefoni portatili stanno diventando dei mini-computer, somigliando sempre più ai palmari. Il rapido sviluppo delle tecnologie di telecomunicazione sta portando alla convergenza tra telefonia e Internet. A testimoniare le possibilità di questa nuova frontiera tecnologica è l'UMTS (*Universal Mobile Telephone Service* – servizio di telefonia mobile universale), un sistema di telefonia mobile per proporre Internet sul telefono portatile.

L'elevato sviluppo tecnologico nel campo delle telecomunicazioni ha portato varie case produttrici alla produzione di diversi modelli di dispositivi, come telefoni, cellulari, PC e palmari, con caratteristiche tecnologiche mol-

to diversificate. Grazie a questa espansione tecnologica e alla forte richiesta da parte dei consumatori, i costi dei dispositivi sono diminuiti notevolmente, dando la possibilità ad ognuno di noi di possederne diversi, più o meno tecnologicamente avanzati, utilizzati sia per la comunicazione verbale che per il trasferimento di dati opportunamente formattati.

Se da una parte questo sviluppo ha portato gli utenti ad avere molti dispositivi con cui comunicare, dall'altra ha introdotto alcuni problemi. Principalmente il problema di doversi ricordare svariati identificatori di dispositivi e quali formati dei messaggi sono supportati da ogni singolo dispositivo. Inoltre, ogni utente può voler acquistare o cambiare dispositivi senza dover informare del cambiamento effettuato le persone che potrebbero inviargli un messaggio.

Al fine di risolvere i problemi delineati viene introdotto il concetto di Identificatore Universale per la Comunicazione (*Universal Communication Identifier, UCI*). Questo è un numero unico associato ad ogni utente indipendentemente dai dispositivi posseduti o utilizzati. Un utente può aggiungere, rimuovere o modificare i dispositivi, ma tale identificativo rimane invariato. In questo modo è possibile spedire un generico messaggio ad un utente senza preoccuparsi di indirizzarlo al dispositivo opportuno per la sua ricezione.

La spedizione di un messaggio richiede che allo stesso siano associate diverse informazioni, quali:

- L'UCI del mittente.
- L'UCI del destinatario, che può anche essere un gruppo. I gruppi

definiti dagli utenti devono contenere UCI appartenenti al sistema. Se un messaggio viene inviato ad un gruppo, una copia del messaggio sarà creata per essere spedita ad ogni destinatario.

- Informazioni relative al messaggio stesso, come l'identificatore, la dimensione, il formato del messaggio, la sensitività, l'eventuale stato di crittazione, l'eventuale stato di compressione, la firma digitale del mittente, la data di spedizione, il contenuto.

- Le credenziali che l'utente vuole condividere. Le credenziali sono un insieme di attributi che caratterizzano l'utente in un determinato ambito. Una credenziale è la versione elettronica dei documenti che in genere ciascuno porta nel proprio portafoglio. Le credenziali sono utili al momento della spedizione di un messaggio, quando si vogliono comunicare al ricevente le proprie caratteristiche. Diverse credenziali possono essere associate in un unico insieme per dare al ricevente del messaggio maggiori informazioni personali.

Gli utenti, attraverso i dispositivi che posseggono, possono scambiarsi messaggi di diversi formati, come immagini, email, video, fax. Può succedere che un determinato dispositivo non sia in grado di gestire il messaggio direttamente. Per questo motivo, sono stati previsti metodi per tradurre il messaggio in un nuovo formato, rendendolo compatibile con il dispositivo stesso. Poichè ogni messaggio è legato alla sensitività dell'informazione che trasporta, può essere consegnato ad un dispositivo solamente se il livello di sicurezza di quest'ultimo lo permette, indipendentemente dalla compatibilità del formato.

Il sistema, proposto in questa tesi, è basato sull'uso di regole attive per instradare un messaggio, in arrivo ad un certo UCI, su dispositivi adatti a riceverlo. Il sistema permette all'utente di definire regole di instradamento per gestire i messaggi in arrivo. Attraverso le regole è possibile definire differenti tipi di condizioni, raggruppati nelle seguenti categorie:

1. *Condizione sul messaggio.* È possibile definire una condizione sui diversi parametri del messaggio. Ad esempio, sul suo tipo (audio, stream, voice, phone call, email...), sul livello di sicurezza dell'informazione contenuta nel messaggio, se il messaggio è stato compresso o crittato, sul tipo di dispositivi in grado di gestirlo e così via.

2. *Condizioni sull'utente.* È possibile definire filtri sull'UCI, sul gruppo di appartenenza del mittente e sulle proprietà specificate nelle credenziali del mittente.

3. *Condizioni temporali.* È possibile definire una condizione sul momento di arrivo del messaggio.

Le regole di instradamento sono definite da una grammatica e hanno al loro interno diversi tipi di condizione da valutare. Ogni utente specifica un certo numero di regole a sua discrezione, per gestire i messaggi in arrivo e consegnarli a particolari dispositivi. Le regole permettono di rifiutare un messaggio oppure di specificare i dispositivi su cui instradarlo. In particolare, un messaggio può essere instradato su:

- *Un singolo dispositivo.* La regola specifica l'identificatore del dispositivo dove il messaggio deve essere spedito. In questo caso, è possibile

definire un metodo di traduzione per ricevere il messaggio in un altro formato.

- *Una lista di dispositivi.* La regola specifica gli identificatori dei dispositivi, dove il messaggio deve essere spedito. Il messaggio può essere spedito a tutti o ad almeno uno dei dispositivi della lista.

- *Un insieme di dispositivi specificati da una condizione.* La regola specifica una condizione sulla capacità o sullo stato del dispositivo.

Il sistema, che riceve un messaggio per un particolare UCI, stabilisce le regole che possono essere utilizzate, in base alle caratteristiche del messaggio. Valutando tali regole, il sistema identifica l'insieme di dispositivi adatti a ricevere il messaggio. Attraverso servizi esterni, offerti da un *provider*, è possibile controllare lo stato dei dispositivi e verificare se la consegna avviene correttamente.

Quando un messaggio arriva all'UCI di un utente, può capitare che il sistema non trovi dispositivi a cui consegnarlo. Questa situazione può essere causata da una scorretta definizione delle regole, o dall'impossibilità di individuare i dispositivi, dovuta alla possibilità di esprimere differenti tipi di condizioni. Il messaggio, inoltre, non potrà essere consegnato se non esistono dispositivi compatibili. La compatibilà di un messaggio con un dispositivo è legata sia al suo formato sia alla sua sensitività. Il formato, che un dispositivo è in grado di gestire direttamente, deve essere lo stesso del messaggio, oppure, deve esistere un metodo di traduzione che permette al dispositivo di gestire il messaggio tradotto. Il livello di sicurezza del dispositivo deve essere superiore al livello di sensitività del messaggio. Infatti, per

problemi di sicurezza, se un dispositivo è in grado di garantire un certo livello di riservatezza delle informazioni ricevute, non può gestire messaggi che richiedono un livello maggiore. Nonostante esistano dispositivi compatibili con il messaggio può succedere che la consegna non avvenga correttamente. Le cause possono essere diverse, come ad esempio, la non disponibilità del dispositivo. Il messaggio viene, quindi, rispedito un certo numero di volte[1] per tentare la consegna del messaggio. Qualora il sistema non riuscisse a consegnare il messaggio, lo memorizza nella base di dati, per permettere al ricevente, in un secondo momento, di recuperarlo. Una notifica viene spedita all'utente per informarlo della situazione di errore verificatasi.

Messaggi di notifica possono essere anche spediti su richiesta dell'utente che desidera conoscere su quali dispositivi un dato messaggio è stato consegnato. Nella base di dati vengono memorizzati, oltre ai messaggi non consegnati, tutte le informazioni degli utenti.

Ciascun utente viene registrato nel sistema dall'amministratore inserendo l'UCI e la parola chiave assegnati a tale utente. Il sistema, sulla base di tali informazioni, crea un profilo che inizialmente conterrà solo l'UCI dell'utente stesso. In seguito, gli utenti potranno dichiarare i propri dispositivi, stabilire le proprie credenziali, creare gruppi e definire regole per la gestione dei messaggi in arrivo. Le informazioni sui dispositivi e le regole sono contenute nel profilo che, al momento dell'identificazione, viene caricato in memoria e poi aggiornato ad ogni modifica apportata. In particolare nel profilo vengono memorizzati:

- UCI (cioè l'identificatore dell'utente).

[1] Questo è un parametro definito dal sistema.

- La lista dei dispositivi utilizzati effettivamente dall'utente, vale a dire i dispositivi di cui è proprietario e che non ha prestato ad altri utenti, e quelli che ha ricevuto in prestito.
- La lista dei dispositivi di notifica, dove l'utente desidera ricevere i messaggi di notifica.
- La lista delle regole di instradamento specificate dall'utente per gestire i messaggi in arrivo al proprio UCI.

Per rappresentare la situazione sopra descritta, è stato sviluppato un applicativo in Java, che interagisce con una base di dati Oracle su cui vengono memorizzati i dati degli utenti, dei dispositivi e dei messaggi. Il progetto è stato realizzato in Microsoft Windows NT 4.0 utilizzando Oracle 8i Personal Edition e Borland Jbuilder 4.

Il presente lavoro di tesi si è focalizzato sulla progettazione della base di dati, sull'architettura del sistema e sulle interfacce per l'utente e per l'amministratore. La tesi è organizzata in 5 capitoli. Nel Capitolo 1 vengono illustrate alcune architetture di sistemi telefonici esistenti. Nel Capitolo 2 viene presentato il modello dei dati del motore a regole. Nel Capitolo 3 viene spiegata l'architettura del sistema progettato. Nel Capitolo 4 viene mostrata la struttura delle interfacce per gli utente e per l'amministratore. Nell'ultimo capitolo vengono affrontati i possibili sviluppi futuri.

Capitolo 2

Modello dei dati del motore a regole

Per conservare la grande quantità di dati di utenti, dispositivi e messaggi, è stato necessario creare una struttura in grado di acquisire, elaborare, trasmettere ed archiviare informazioni. Questa struttura è un sistema informativo, la cui realizzazione ha come componente fondamentale il DBMS (*Data Base Management System* - sistema per il trattamento di basi di dati). Una base di dati è una collezione di dati correlati, condivisa tra più applicazioni, che deve soddisfare il fabbisogno informativo di tutte le applicazioni. Per poter usare i servizi di un DBMS, in una qualsiasi applicazione, è necessario poter rappresentare le entità, del modello reale di interesse all'applicazione, mediante un insieme di concetti che il DBMS è in grado di trattare. Tale insieme di concetti è detto *modello dei dati*, ossia un insieme di strumenti concettuali, o formalismo, che consiste di tre componenti fondamentali:

1. un insieme di strutture dati,
2. una notazione per specificare i dati,
3. un insieme di operazioni per manipolare i dati.

Per tanto un modello dei dati è un formalismo matematico per descrivere: dati, che rappresentano entità di un dato ambito aplicativo; interrelazioni tra dati, che rappresentano associazioni semantiche tra le entità descritte dai dati; vincoli semantici sui dati, che rappresentano le proprietà semanticamente significative per le entità rappresentate dai dati. Dato che i modelli dei dati servono per rappresentare una certa realtà di interesse per un certo insieme di applicazioni è utile identificare i concetti alla base di una tale rappresentazione, ovvero:

- *Entità*, oggetto della realtà applicativa di interesse, esistente e distinguibile da tutti gli altri.
- *Attributo*, una proprietà significativa ai fini della descrizione della realtà applicativa di interesse di una data enitità; ogni entità è pertanto caratterizzata da uno o più attributi.
- *Insieme di entità,* raggruppa un insieme di oggetti con le stesse caratteristiche, ossia entità aventi gli stessi attributi, anche se non necessariamente gli stessi valori per gli attributi.
- *Associazione*, una corrispondenza tra gli elementi di due (o più) insiemi di entità.

Un esempio di modello dei dati è rappresentato dal modello relazionale, basato su una singola struttura dati: la relazione[1] [1]. La descrizione dei dati memorizzati in una base di dati, specificata tramite il modello dei dati, è lo schema della base di dati.

Progettare una base di dati significa definire in modo preciso il suo contenuto informativo e la struttura che tale contenuto dovrà possedere. Il modello, che comunemente viene utilizzato per la fase di definizione dei concetti che si vogliono modellare e delle relazioni che intercorrono tra essi, è il modello entità-relazione (in seguito denotato con **ER**). I componenti principali del modello ER sono:

- *Entità*, rappresenta un insieme di oggetti della realtà, che possiedono caratteristiche comuni. Gli oggetti, appartenenti ad una certa entità, rappresentano le istanze dell'entità.

- *Relazione* rappresenta un legame logico tra due entità. Le istanze di una relazione denotano combinazioni di istanze delle entità che prendono parte alla relazione.

- *Attributo* rappresenta una proprietà elementare posseduta da entità o da relazioni. Un insieme di attributi che identificano univocamnete le istanze dell'entità è chiamato *identificatore* o *chiave*.

- *gerarchia di generalizzazione* una entità E è una generalizzazione delle entità E_1, E_2,...,E_n se ogni istanza delle entità E_1, E_2,...,E_n è anche istanza di E. L'entità E viene chiamata *entità padre*, mentre le entità

[1] Una tabella con righe e colonne contente dati di tipo specificato.

E_1, E_2,...,E_n sono dette *entità figlie*. Tutte le proprietà delle entità padre sono anche proprietà delle entità figlie.

Nel modello ER viene data una rappresentazione grafica delle componenti descritte, riportata in Tabella 2.1. In questo capitolo verrà decritto lo schema ER della base di dati utilizzata dal motore a regole, analizzando separatamente tre parti: lo schema dei dispositivi, lo schema degli utenti e lo schema dei messaggi.

Il capitolo è oragnizzato nel seguente modo. Nella Sezione 2.1 sono state descritte tutte le informazioni riguardanti i dispositivi, come questi sono stati modellati all'interno dello schema ER, e come è stato affrontato il problema del prestito durante la fase di progettazione. Nella Sezione 2.2 sono stati introdotti gli utenti e ciò che il sistema permette loro di definire. Sono, quindi, stati specificati i concetti di credenziali, gruppi e regole di instradamento. Il sistema crea per ogni utente un profilo che riassume le informazioni principali sui dispositivi utilizzati e sulle regole impostate. Nella Sezione 2.3 è stata analizzata la stuttura dei messaggi, spiegando i concetti di formato e metodo di traduzione.

2.1 Dispositivi

Sul mercato delle telecomunicazioni esistono diverse categorie di dispositivi con caratteristiche tecnologiche differenti, come, ad esempio, cellulari, telefoni, PC. Per ogni tipo di dispositivo, le varie case produttrici sviluppano molti modelli, con caratteristiche tecniche sempre più sofisticate, che cercano di migliorare in ogni momento la qualità del prodotto. Un dispositivo

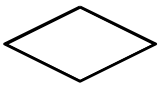
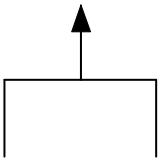
Componente	Simbolo
Entità	
Relazione	
Attributo	
Gerarchia di generalizzazione	
Identificatore	
Vincolo di cardinalità	(c_{min},c_{max})

Tabella 2.1: Principali simboli grafici utilizzati nello schema ER

rappresenta l'istanza di un dato modello di dispositivi, univocamente identificato da un numero. Ad esempio, il Motorola Timeport 3200 è un modello di cellulare realizzato dalla Motorola.

In questa sezione vengono presentati i concetti di tipo, modello e dispositivo, mostrando le relazioni che intercorrono fra queste entità.

Nella Sezione 2.1.1 viene spiegato il concetto di tipo di dispositivo. Nella Sezione 2.1.2 viene presentato il modello di un dispositivo. Nella Sezione 2.1.3 vengono analizzate tutte le caratteristiche di un dispositivo. Nella Sezione 2.1.4 viene definito lo stato del dispositivo.

2.1.1 Tipi di dispositivi

Ogni dispositivo appartiene ad un tipo indipendentemente dalla compagnia che lo produce. Ciascun tipo è caratterizzato dalle proprietà che descrivono

le sue capacità e dai servizi che è in grado di offrire. Esempi di tipi possono essere il telefono, il cellulare, il PC, il PDA (*Personal Digital Assistant*. Per esempio, un PC può essere *desktop*, *laptop* o *potable*, mentre un telefono può essere *PSTN* o *IP*, ciascuno con le caratteristiche che ne conseguono.
Diversi modelli hanno caratteristiche comuni a seconda del tipo di dispositivo a cui appartengono, per cui si è creata una gerarchia di generalizzazione per conoscere per ogni tipo le sue capacità e proprietà (Figura 2.1).

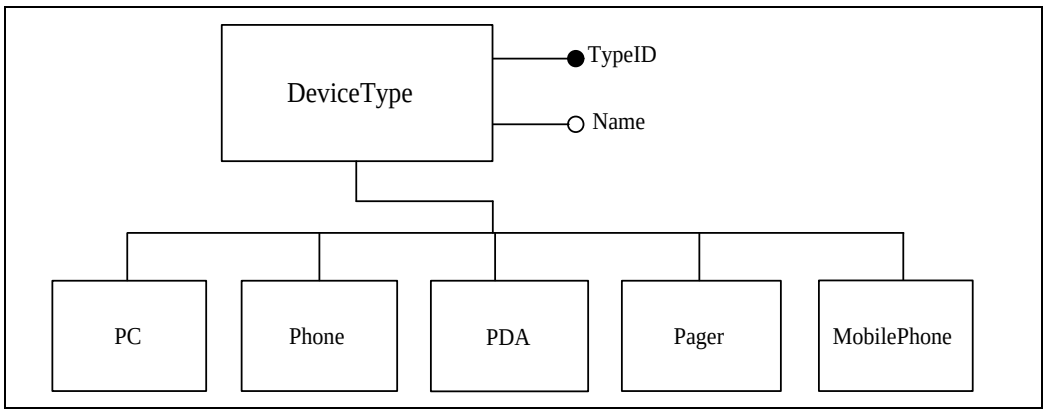


Figura 2.1: Gerarchia dei tipi

2.1.2 Modelli di dispositivi

Per ciascun tipo di dispositivo esistono tanti modelli differenti, come, ad esempio, tra i cellulari esiste il Panasonic GD90 e il Siemems S25.
Ogni modello di dispositivo ha delle caratteristiche intrinseche quali, ad esempio, la capacità di vibrare in ricezione di una chiamata, la dimensione del display, la possibilità di personalizzare le melodie. Queste funzionalità cambiano secondo il tipo e/o modello di dispositivo.
L'entità **DeviceModel** (Figura 2.2) rappresenta quindi l'insieme di modelli di dispositivi ed è caratterizzata dall'identificatore univoco *DevModelID* e

dal nome *Name*. L'associazione uno a molti *DevModelHasType* (Figura 2.3)

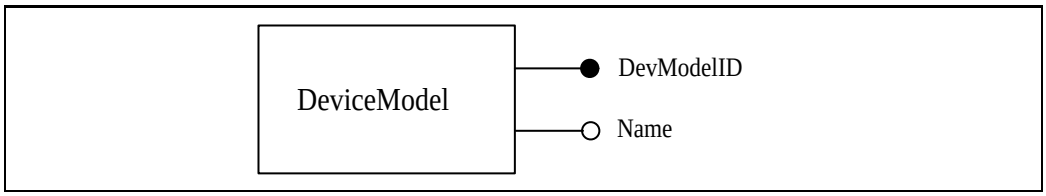


Figura 2.2: Modelli di dispositivi

mostra la relazione esistente tra i tipi e i modelli di dispositivi. Per ogni tipo esistono più modelli, mentre ciascun modello può appartenere ad un solo tipo. In questo modo, un dispositivo assume le stesse capacità del tipo e quelle tipiche del modello.

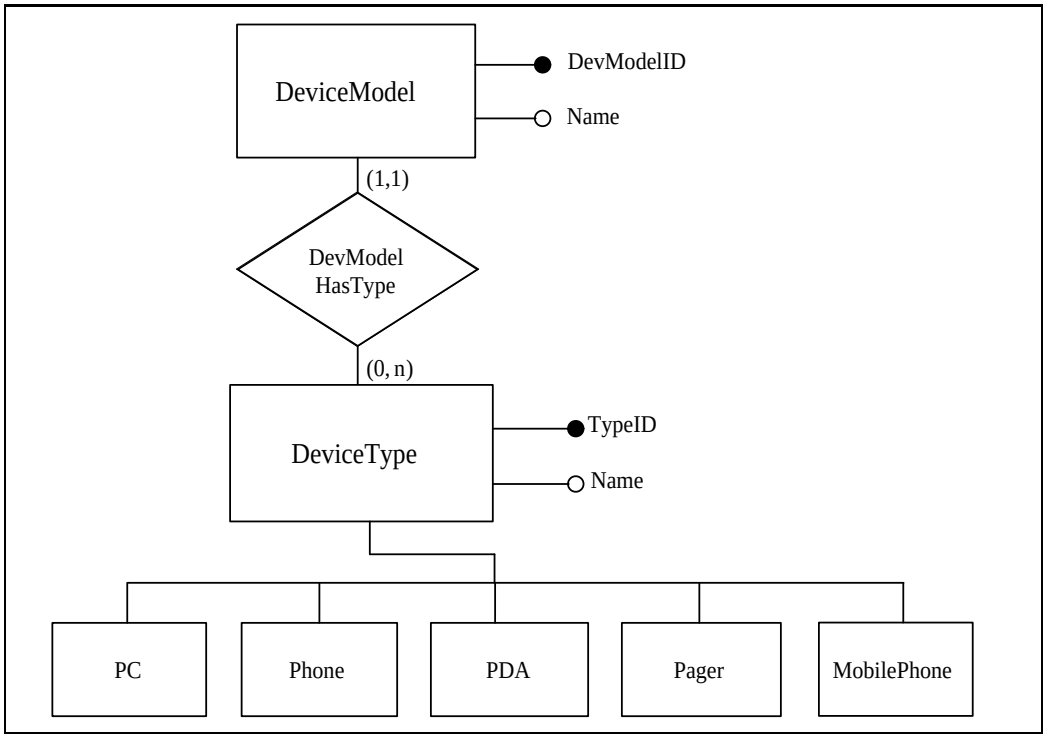


Figura 2.3: Relazione tra tipi e modelli

2.1.3 Dispositivi

Un dispositivo, univocamnete identificato da un numero, è un particolare modello associato ad un utente. Per esempio, dato un modello "Siemens S21 cellulare", Bob ha il dispositivo identificato dal numero 100.
Ogni dispositivo, a seconda del modello di appartenenza, può gestire la ricezione di determinati formati di messaggio. Tuttavia, se un dispositivo è in grado di garantire un certo livello di riservatezza delle informazioni ricevute, non può gestire messaggi che richiedono un livello maggiore, nonostante la compatibilità di formato.

Se un utente deve spedire un messaggio con formato e sensitività non supportati dai dispositivi che possiede, può decidere di chidere in prestito un dispositivo. Per questo motivo, occorre distinguere per ogni dispositivo il proprietario da chi lo utilizza effettivamente. Per esempio, se Bob ha comprato un Siemens S21 e lo ha prestato ad Alice, Bob è il proprietario e Alice la persona che lo utilizza per tutta la durata del prestito. È stato previsto un unico livello di prestito (Alice non può prestare il Siemens S21 di Bob ad altri) senza condivisione, quindi non è possibile che più utenti utilizzino lo stesso dispositivo contemporaneamente. Questa limitazione è stata posta per ragioni di sicurezza. Infatti, si dovrebbe controllare chi utilizza il dispositivo condiviso, e che il dispositivo sia utilizzato nei limiti delle capacità concesse dal proprietario del dispositivo. Il problema di gestire la condivisione di dispositivi tra più utenti rimane, comunque, una delle possibili espansioni future al progetto UCI.
Le due entità, che descrivono la situazione sopra riportata, sono **Device** e

TemporaryDevice(Figura 2.4), che rappresentano i diversi tipi di informazioni che si vogliono memorizzare.

L'entità **Device** è caratterizzata dai seguenti attributi:

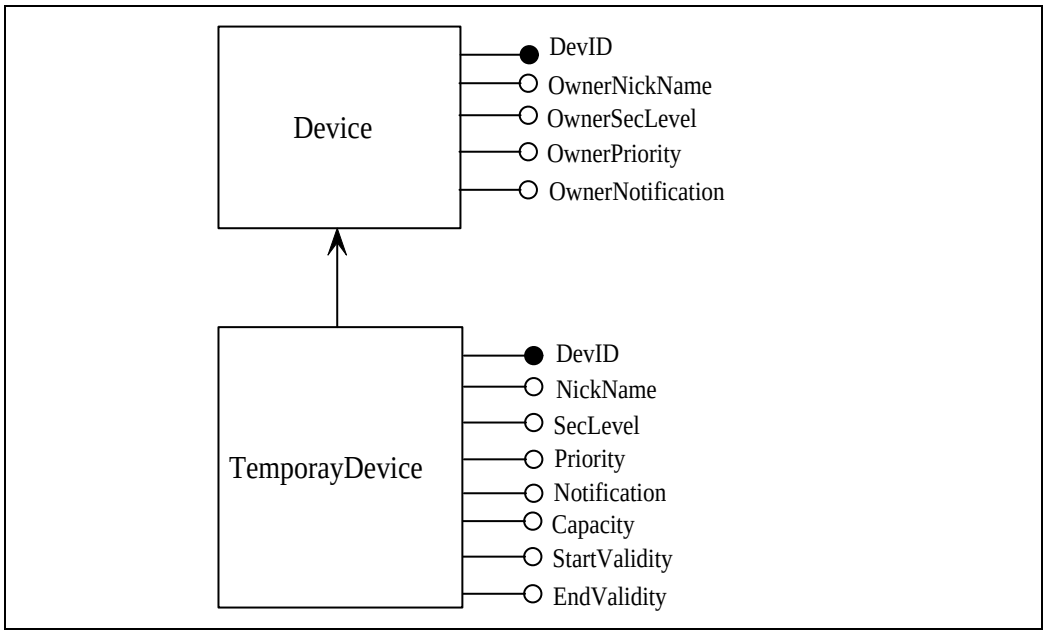


Figura 2.4: Entità dispositivo

- *DevID* è l'identificatore del dispositivo.
- *OwnerNickName* è il soprannome dato al dispositivo dal proprietario.
- *OwnerSecLevel* indica il livello di sicurezza dato al dispositivo dal proprietario. Esistono tre livelli HIGH, NORMAL, LOW che indicano se un device può o meno ricevere un messaggio. Per esempio, un dispositivo con basso livello di sicurezza non può ricevere un messaggio con un alto livello di sensitività.
- *OwnerPriority* permette la scelta tra più dispositivi. La scelta tra i

tre livelli possibili HIGH, NORMAL, LOW è effettuata da parte del proprietario del dispositivo.

- *OwnerNotification* indica che il proprietario aveva selezionato il dispositivo per ricevere messaggi di notifica.

In caso di prestito, le informazioni sul dispositivo del proprietario verranno mantenute nei campi *OwnerNickName*, *OwnerSecLevel*, *OwnerPriority*, *OwnerNotification*, mentre l'utilizzatore potrà stabilire le caratteristiche del dispositivo attraverso l'entità **TemporaryDevice**, definita dai seguenti attributi:

- *NickName* è il soprannome dato al dispositivo da chi lo utilizza effettivamente.
- *SecLevel* indica il livello di sicurezza assegnato al dispositivo da parte di chi lo usa.
- *Priority* facilita la scelta tra più dispositivi. Il valore viene impostato dall'utilizzatore del dispositivo.
- *Notification* distingue tra i dispositivi quelli indicati per ricevere le notifiche.
- *Capacity* permette, durante il prestito, di limitare l'utilizzo del dispositivo, definendo una disponibilità fisica opportuna.
- *StartValidity* indica la data di inizio del prestito.
- *EndValidity* indica la data di scadenza del prestito.

Dato che un utente può essere sia possessore che utilizzatore di dispositivi, si vengono a creare le due associazioni *UserOwnsDev* e *UserUsesDev* per rappresentare il legame di "proprietario di" o "utilizzatore di" con i dispositivi. L'associazione uno a molti *UserOwnsDev* che lega **Device** e **User** implica che un utente può possedere più dispositivi, sebbene possa esistere un utente che non ne possiede (Figura 2.5).

L'associazione *UserUsesDev* tra le entità **User** e **Device** permette ad un utente di utilizzare anche dispositivi prestati (Figura 2.5). Questa associazione è uno a molti in quanto non esiste condivisione.

Poichè esistono più dispositivi corrispondenti ad un solo modello, le entità

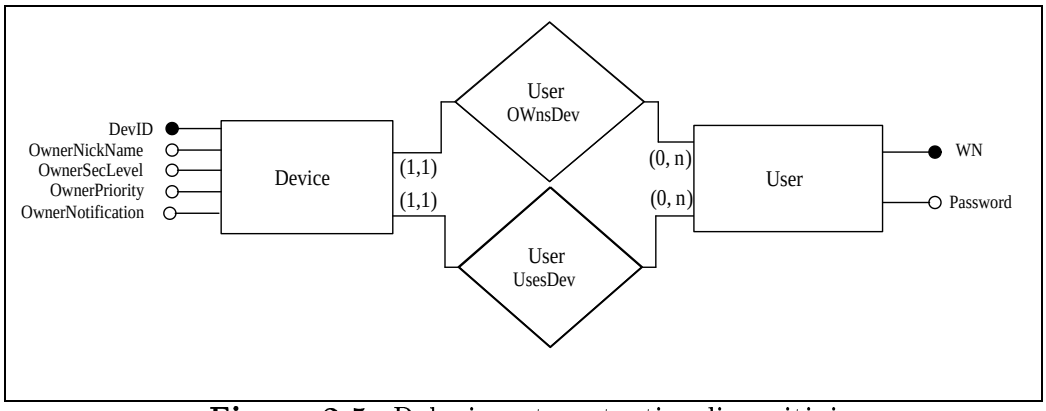


Figura 2.5: Relazione tra utenti e dispositivi

DeviceModel e **Device** sono messe in relazione dall'associazione molti a uno *DevIsModelOf*, con partecipazione obbligatoria della seconda entità e opzionale della prima (Figura 2.6).

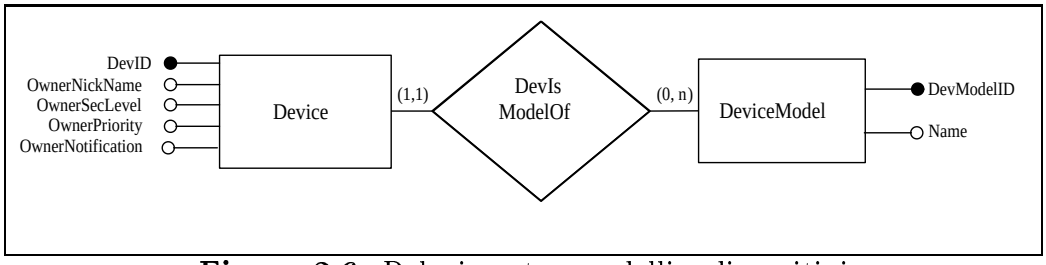


Figura 2.6: Relazione tra modelli e dispositivi

2.1.4 Stato di un dispositivo

Per stato si intende lo stato fisico assunto in un determinato momento dal dispositivo. Il tipo di stato, che un dispositivo può assumere, dipende dal modello di appartenenza e dalle proprietà specifiche possedute. Per esempio, lo stato "busy" è tipico dei dispositivi telefonici, mentre "low disc space" è assumibile dai dispositivi caratterizzati da un supporto di memorizzazione. Ogni dispositivo può, quindi, assumere uno o più stati, rappresentati attraverso l'entità **Status** (Figura 2.7) caratterizzata dagli attributi nome *Name* e identificatore *StatusID*. Ad ogni modello di dispositivo vengono assegnati

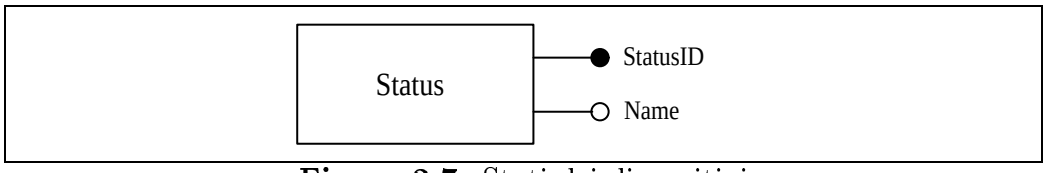


Figura 2.7: Stati dei dispositivi

determinati stati caratteristici, tramite l'associazione *DevModelHasStatus* tra **DeviceModel** e **Status**, che è di tipo molti a molti con partecipazione opzionale di entrambe le entità (Figura 2.8).

Il controllo dello stato dei dispositivi avviene attraverso un servizio esterno supportato da un *provider*, che valuta gli stati che un dispositivo assume in un determinato istante. Ad esempio, un dispositivo può trovarsi contem-

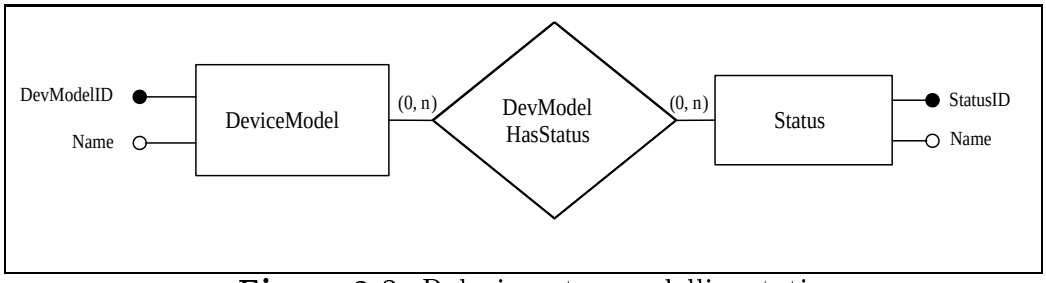


Figura 2.8: Relazione tra modelli e stati

poraneamente sia nello stato "available" che in quello "low battery". Nella fase di progettazione, si è quindi assunto che, tra tutti gli stati assumibili dai vari modelli, un dispositivo può assumerne diversi. A modellare questa associazione molti a molti, che lega **Device** a **Status**, è *DevIsInStatus*, obbligatoria per il Device e opzionale per lo Status (Figura 2.9).

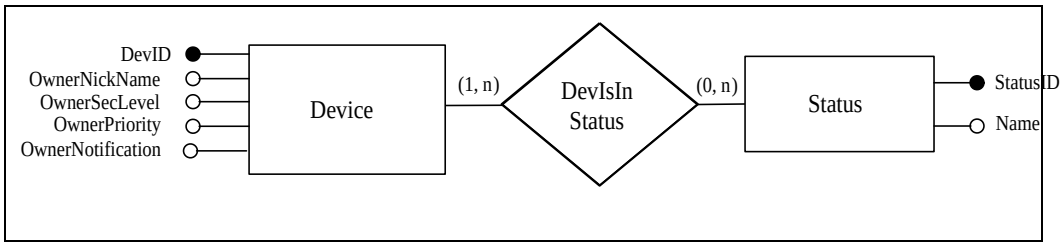


Figura 2.9: Relazione tra dispositivi e stati

2.1.5 Schema ER sui dispositivi

In questa sezione sono state analizzate tutte le componenti dello schema ER sui dispositivi, illustrato in Figura 2.10.

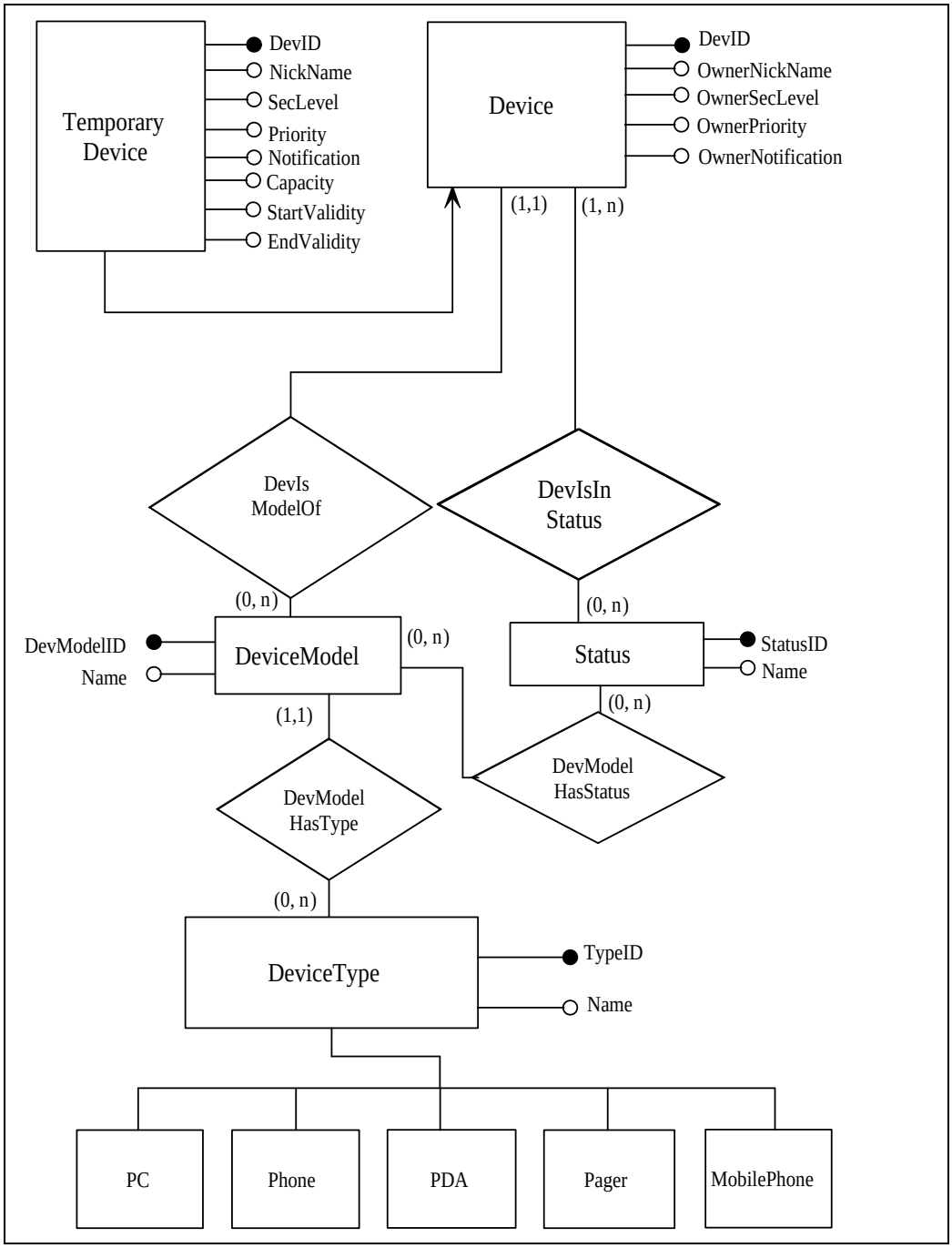


Figura 2.10: Schema dei dispositivi

2.2 Utenti

Ogni utente viene univocamnete identificato all'interno del sistema grazie al suo UCI. Attraverso opportune interfacce, un utente ha la possibilità di dichiarare i dispositivi di cui è in possesso, e di definire le proprie caratteristiche personali. Al fine di gestire al meglio l'instradamento dei messaggi in arrivo al proprio UCI, l'utente può stabilire delle regole, che verranno poi valutate dal sistema, per la corretta redirezione del messaggio, sui dispositivi degli utenti che verificano le regole valide. Al fine di spedire lo stesso messaggio a più persone, per un utente può essere utile definire dei gruppi. La creazione di gruppi permette inoltre di filtrare i messaggi in arrivo al proprio UCI.

In questa sezione vengono, quindi, introdotti i concetti di utente, credenziali, gruppi, regole di instradamento e profilo, mostrando le relazioni che intercorrono fra queste entità.

Nella Sezione 2.2.1 vengono introdotti gli utenti del sistema. Nella Sezione 2.2.2 viene definite le credenziali degli utenti. Nella Sezione 2.2.3 vengono analizzate le regole di instradamento. Nella Sezione 2.2.4 viene definita la funzionalità dei gruppi. Nella Sezione 2.2.5 vine presentato il concetto di profilo.

2.2.1 Utenti

Essendo gli utenti univocamente identificati, nel sistema progettato, tramite i loro UCI, è possibile trovare all'interno della base di dati tutte le loro informazioni personali, del tipo:

- *Dispositivi*, i dispositivi che un utente ha definito nel sistema, con i rispettivi attributi;
- *Credenziali*, caratteristiche riguardanti la vita dell'utente, come ad esempio la ditta per cui lavora o l'indirizzo;
- *Gruppi*, insiemi di UCI definiti dall'utente (ad esempio *Amici*);
- *Regole di instradamento*, politiche per la redirezione del messaggio verso i dispositivi definiti dall'utente;

Tali informazioni possono essere inserite nella base di dati dopo che è avvenuta la registrazione al sistema. Ciascun utente viene registrato nel sistema dall'amministratore attraverso la memorizzazione dell'UCI, di una parola chiave, e la creazione di un profilo che inizialmente conterrà solo l'UCI dell'utente stesso. In seguito, gli utenti potranno dichiarare i propri dispositivi e definire regole per la gestione dei messaggi ricevuti. Queste informazioni sono contenute nel profilo, aggiornato ogni volta che l'utente effettua delle modifiche.

Gli utilizzatori e i possessori dei dispositivi sono modellati dall'entità **User** (Figura. 2.11). Ogni utente è caratterizzato da un identificatore univoco, *UCI*, dalla sua parola di identificazione *Password*, per l'accesso al sistema, e dal profilo *Profile*.

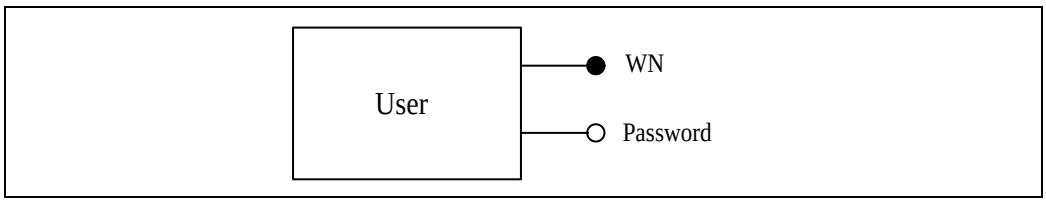


Figura 2.11: Utenti

2.2.2 Credenziali

Ogni utente ha la possibilità di definire le proprie credenziali, ossia un insieme di attributi che lo caratterizzano in un determinato ambito. Una credenziale è la versione elettronica dei documenti che in genere ciascuno porta nel proprio portafoglio. Le credenziali sono utili al momento della spedizione di un messaggio, quando si vogliono comunicare al ricevente le proprie caratteristiche. Esempi di informazioni che possono essere spedite al ricevente del messaggio sono il nome, il cognome, il tipo di impiego oppure la società per cui si lavora. Per questo, sono state definite più credenziali utilizzabili dagli utenti, come, ad esempio, la tessera della biblioteca, le credenziali personali, quelle pubbliche e quelle legate al lavoro. Per credenziali pubbliche si intendono le informazioni utili a qualsiasi ricevente, per questo motivo comprendono solamente il nome e il cognome dell'utente. Al contrario, le credenziali private contengono informazioni personali che non sempre l'utente è intenzionato a comunicare, come la propria data di nascita e il proprio indirizzo di casa. Si possono raggruppare insieme diverse credenziali per dare al ricevente del messaggio maggiori informazioni. Ad esempio, una società sarà interessata a ricevere sia le credenziali pubbliche che quelle legate al lavoro.

L'entità **Credential**, con attributo identificatore *CredID*, è al primo livello della gerarchia di generalizzazione che ne permette la definizione (Figura 2.12).

L'insieme delle credenziali associate ad un utente definiscono le sue **UserCard**, caratterizzate da un identificatore *UserCardID* e da un nome *Name*

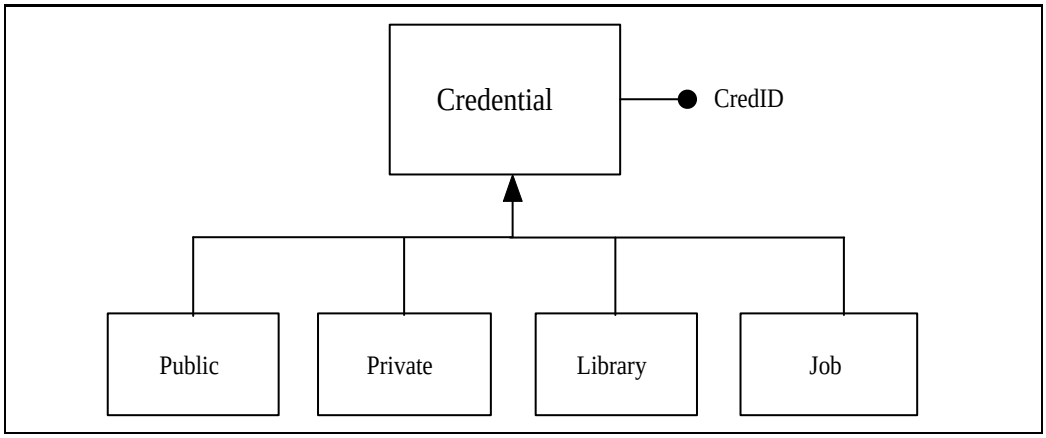


Figura 2.12: Gerarchia delle credenziali

(Figura 2.13).

L'entità **Credential** è legata all'entità **UserCard** dall'associazione uno a

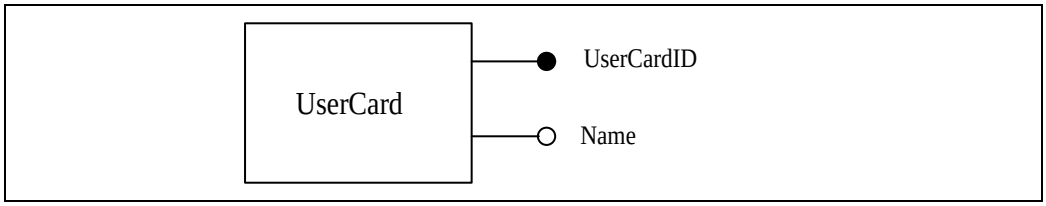


Figura 2.13: UserCard

molti *CardHasCredential*. Una UserCard è composta da una o più credenziali, una sola per tipo (Figura 2.14).

A sua volta l'entità **User** è legata all'entità **UserCard** con l'associazione uno a molti *UserHasCard*, in quanto ogni utente possiede più UserCard, a seconda delle sue esigenze[2], e una UserCard non ha motivo di esistere se non è assegnata ad un utente (Figura 2.15).

[2]Per esigenze si intendono i diversi insiemi di credenziali che si vogliono condividere con i destinatari dei messaggi.

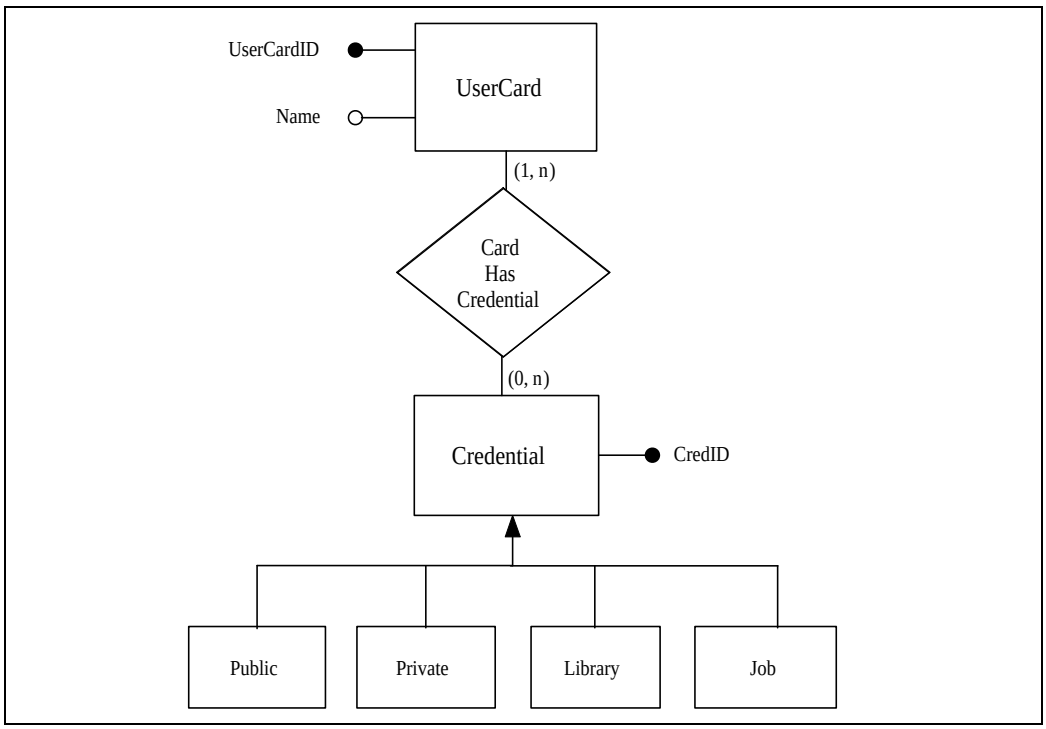


Figura 2.14: Credenziali dell'utente

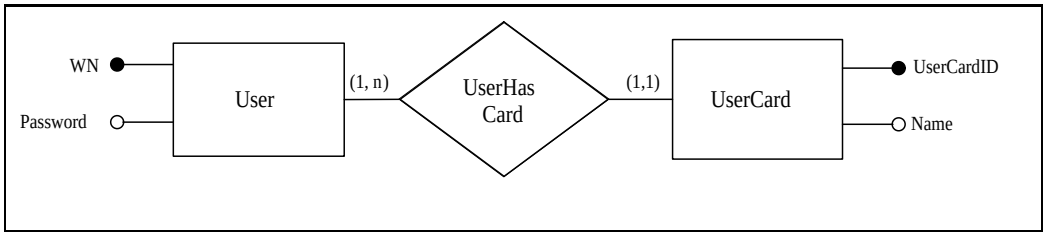


Figura 2.15: Scheda caratteristica dell'utente

2.2.3 Gruppi

I gruppi che l'utente può creare, sono insiemi di UCI, ossia di utenti tutti sottoscritti al sistema, conosciuti dall'utente stesso o che in qualche modo sono a lui legati. La definizione di gruppi può, infatti, essere molto utile all'utente sia in fase di spedizione che di ricezione di un messaggio.
Se un utente vuole spedire un messaggio a diversi amici, troverà comodo inviarlo ad un gruppo da lui definito, piuttosto che a ciascuno di loro singolarmente. Il sistema si preoccuperà di creare e inviare una copia del messaggio ad ogni componente del gruppo.
Al momento della ricezione di messaggi al proprio UCI, la definizione dei gruppi può essere, invece, utilizzata per filtrare i messaggi in arrivo, stabilendo condizioni sui gruppi stessi. Quando il messaggio in arrivo è stato spedito da un membro del gruppo, la regola viene eseguita. Per esempio, supponiamo che Bob definisca il gruppo "amici" contenente gli UCI di Alice e Jack; se Bob stabilisce che ogni email proveniente da un utente in "amici" deve essere instradata sul suo PC e sul suo PDA, ossia un palmare), allora una eventuale email di Jack sarà ricevuta da Bob sia sul PC che sul PDA.
I gruppi sono rappresentati dall'entità **Group** (Figura 2.16) che ha come attributi un nome *Name* e un identificatore *GroupID*.
Le due entità **User** e **Group** sono legate da due diverse associazioni. Se

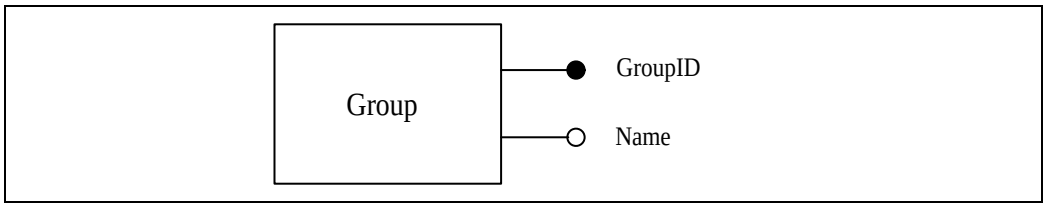


Figura 2.16: Gruppi

l'utente è colui che definisce il gruppo, l'associazione uno a molti che si viene a creare è *UserGroup* con partecipazione opzionale della prima entità e obbligatoria dell'altra (Figura 2.17). Un utente può definire diversi gruppi, ma lo stesso gruppo può essere creato da un solo utente.

Nel caso l'utente sia membro di un gruppo, l'associazione *UserIsInGroup*

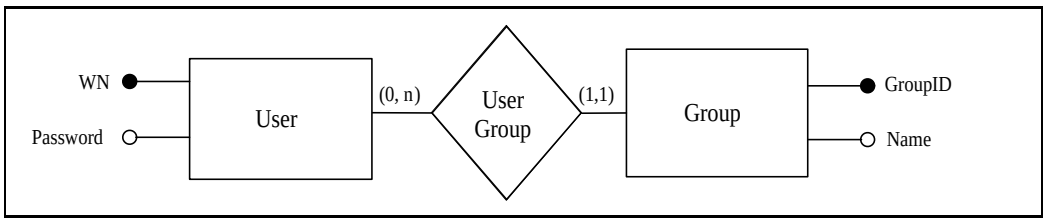


Figura 2.17: Gruppi definiti da utenti

risulta molti a molti con partecipazione opzionale di entrambe le entità. Un utente può appartenere a più gruppi così come un gruppo può contenere più UCI (Figura 2.18).

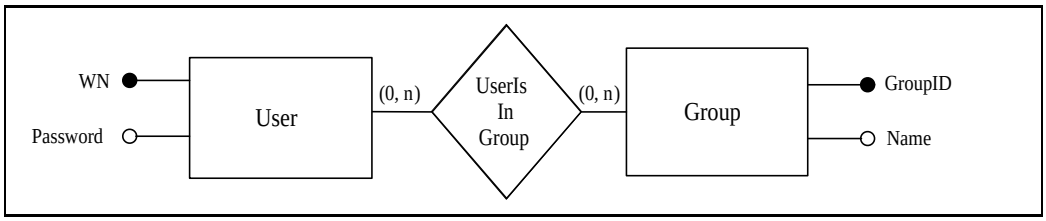


Figura 2.18: Utenti appartenenti ad un gruppo

2.2.4 Regole di instradamento

La possibilità di definire regole nasce dall'esigenza degli utenti di poter decidere quali tipi di messaggi ricevere, in un dato momento, su un determinato dispositivo.

Le regole (o politiche) di instradamento sono definite da una grammatica attraverso cui è possibile definire diversi tipi di condizione. Ogni utente

specifica un certo numero di regole a sua discrezione per gestire i messaggi in arrivo e consegnarli a particolari dispositivi. Queste regole permettono di instradare, sui dispositivi adatti, i messaggi ricevuti.
I dettagli delle regole sono riportati in [2]. Di seguito riportiamo solo le caratteristiche principali. Attraverso le regole è possibile definire condizioni semplici, raggruppate nelle seguenti categorie:

1. *Condizione sul messaggio.* É possibile definire una condizione sui diversi parametri del messaggio. Ad esempio, sul suo tipo (audio, stream, voice, phone call, emai...), sul livello di sicurezza dell'informazione contenuta nel messaggio, se il messaggio è stato compresso o crittato, sul tipo di dispositivi in grado di gestirlo. Per esempio, l'utente Bob può specificare regole del tipo: "rifiuta un messaggio se è un video".

2. *Condizioni sull'utente.* Filtri sull'UCI sono della forma "rifiuta il messaggio se viene da Alice". Altri filtri sono stabiliti a seconda del gruppo di appartenenza: se Bob ha definito il gruppo "indesiderati" al quale Alice appartiene, se Alice chiama allora la telefonata viene rifiutata.

3. *Condizioni temporali.* È possibile definire una condizione sul momento di arrivo di un messaggio. Se, in un determinato momento della giornata, si vogliono rifutare i messaggi in arrivo ad un dato dispositivo, la regola potrebbe essere del tipo: "rifiuta ogni messaggio che arriva di domenica" oppure "rifiuta ogni messaggio che arriva tra le ore 10.00 e le ore 11.30".

Le regole permettono di rifiutare un messaggio, oppure di specificare i dispositivi su cui instradarlo. In particolare un messaggio può essere instradato su:

- *Un dispositivo.* La regola specifica l'identificatore del dispositivo, dove il messaggio deve essere spedito. In questo caso, è possibile definire un metodo di traduzione per ricevere il messaggio in un altro formato.
- *Una lista di dispositivi.* La regola specifica gli identificatori dei dispositivi, dove il messaggio deve essere spedito. Il messaggio può essere spedito a tutti o ad almeno uno dei dispositivi della lista.
- *Un insieme di dispositivi specificati da una condizione.* La regola specifica condizione sulla capacità o sullo stato del dispositivo.

Le regole permettono anche il reinstradamento di un messaggio; per esempio Bob può definire la regola che se un messaggio arriva sul cellulare, deve essere spedito al PC. La possibilità di reinstradare i messaggi, così come la possibilità di esprimere differenti tipi di condizioni, possono portare all'assenza di dispositivi adatti alla ricezione del messaggio. In questo caso, il sistema si preoccupa di spedire un notifica all'utente per informarlo della situazione di errore verificatasi. Tale avvertimento può essere ricevuto solo sui dispositivi che l'utente ha impostato come adatti a ricevere notifiche.
Le regole sono modellate dall'entità **RoutingRule** (Figura 2.19), definita attraverso gli attributi *Name*, unico per ogni utente, e *Rule* che rappresenta l'espressione della regola vera e propria. L'identificatore di tale entità è rappresentato dalla chiave esterna *WN* dell'entità **User**. Dato che ogni utente

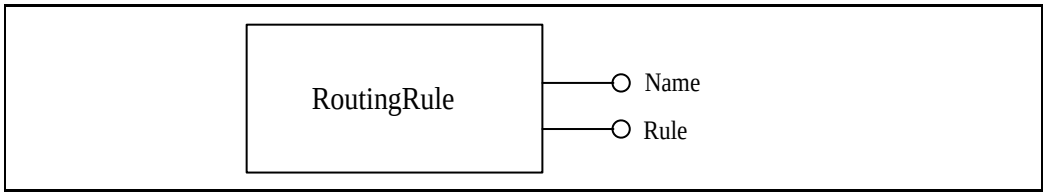


Figura 2.19: Regole di instradamento

può definire più regole, l'associazione uno a molti che si viene a creare tra le entità **User** e **RoutingRule** è *UserDefRules* (Figura 2.20).

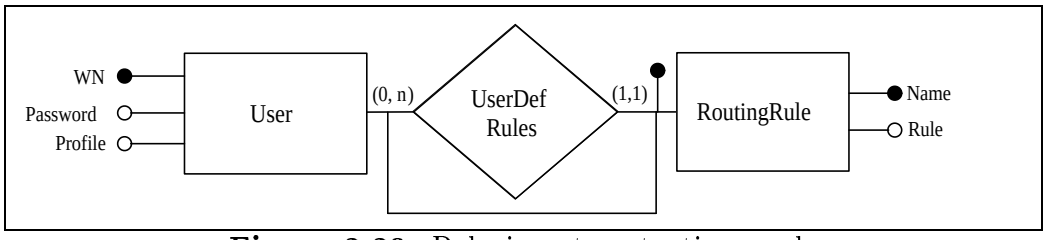


Figura 2.20: Relazione tra utenti e regole

2.2.5 Profilo

Il profilo contiene diverse informazioni legate all'utente. Al momento dell'identificazione, il profilo dell'utente viene caricato in memoria e, succesivamente, aggiornato ad ogni modifica apportata.
In particolare nel profilo vengono memorizzati:

- UCI (cioè l'identificatore dell'utente);
- la lista dei dispositivi effetivamente utilizzati dall'utente, vale a dire i dispositivi di cui è proprietario, e che non ha prestato ad altri, e quelli che ha preso in prestito;
- la lista dei dispositivi di notifica, dove l'utente desidera ricevere informazioni indicanti l'avvenuta consegna o meno dei messaggi in arrivo;

- la lista delle regole di instradamento specificate dall'utente per gestire i messaggi in arrivo al proprio UCI.

Ogni volta che il sistema deve gestire un messaggio arrivato ad un certo utente, viene caricato il profilo in memoria. Infatti, tutte le informazioni utili al sistema per instradare il messagio sui dispositivi dell'utente, sono contenute nel profilo.

2.2.6 Schema ER sugli utenti

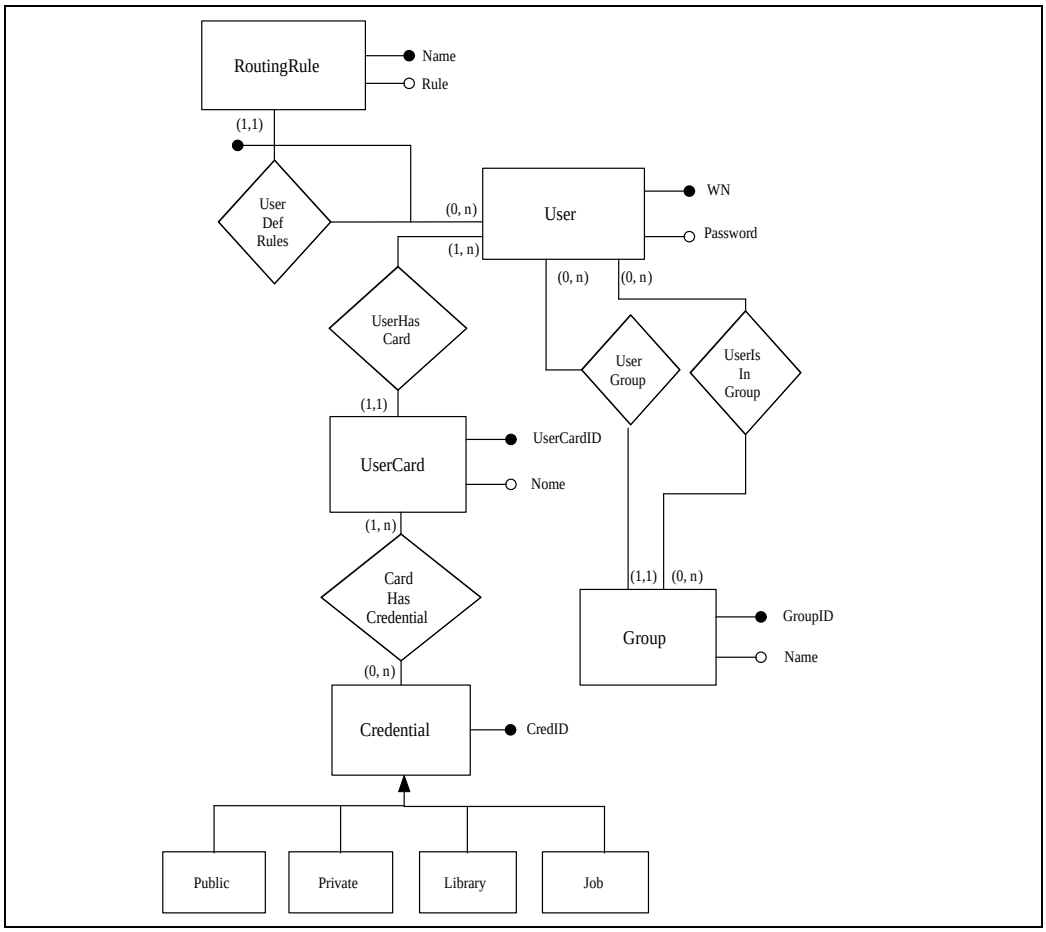


Figura 2.21: Schema degli utenti

2.3 Messaggi

Gli utenti, attraverso i dispositivi che posseggono, possono scambiarsi messaggi di diversi formati. Può succedere che un determinato dispositivo non sia in grado di gestire il messaggio direttamente. Per questo motivo, sono stati previsti metodi per tradurre il messaggio in un nuovo formato, rendendolo compatibile con il dispositivo stesso.
In questa sezione vengono introdotti i concetti di messaggio, formato e metodi di traduzione, mostrando le relazioni che intercorrono fra queste entità. Nella Sezione 2.3.1 vengono spiegati i formati di un messaggio e metodi di traduzione. Nella Sezione 2.3.2 viene descritta la struttura del messaggio.

2.3.1 Formati e metodi di traduzione

Gli utenti comunicano tra loro attraverso messaggi di diversi tipi, come, ad esempio, immagini, email, video, fax. I formati, che un dispositivo può supportare, sono legati al modello del dispositivo stesso; in questo modo, un utente, a seconda dei dispositivi utilizzati, può trasferire o ricevere solo alcuni tipi di messaggi. Ad esempio, se Bob possiede un cellulare, può ricevere telefonate e SMS (*Short Message Service*-Servizio Messaggi Brevi), ma non fax o immagini.
L'entità **MessageFormat** (Figura 2.22) rappresenta i formati possibili per i messaggi. Gli attributi, che compongono questa entità, sono un identificatore *FormatID* e un nome *Name*.
Dato che ciascun modello di dispositivo può ricevere messaggi di un certo formato, le entità **DeviceModel** e **MessageFormat** sono messe in rela-

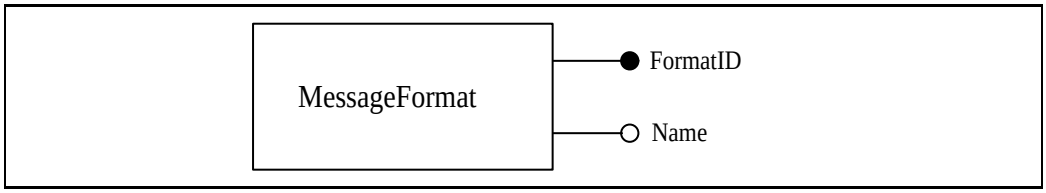


Figura 2.22: Formati del messaggio

zione dall'associazione molti a molti *DevModelHasFormatMsg* con partecipazione obbligatoria della prima e opzionale per la seconda (Figura 2.23). Può succedere che un determinato dispositivo non sia in grado di gestire il

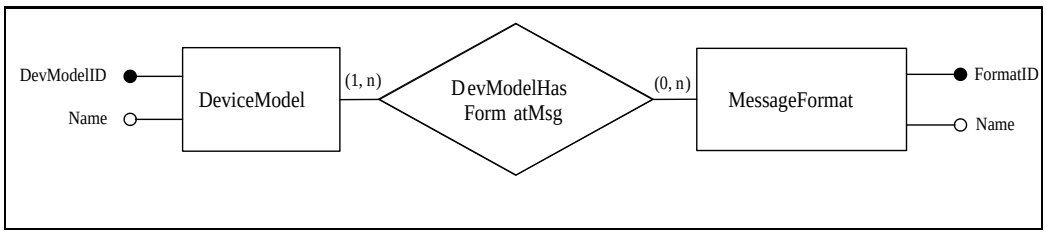


Figura 2.23: Formati per modello

messaggio direttamente, ma occorre applicare un metodo di traduzione per rendere il messaggio gestibile dal dispositivo stesso. Per esempio, un cellulare non è in grado di gestire una email, ma il metodo *email2sms* permette ad un telefono portatile di ricevere i primi 160 caratteri dell'email. L'associazione molti a molti *TranslationMethod* lega l'entità **FormatMessage** (Figura 2.24) a se stessa con partecipazione opzionale di entrambi i ruoli assunti (sorgente e destinatario).

2.3.2 Struttura di un messaggio

Grazie all'UCI, è possibile spedire un generico messaggio ad un utente senza preoccuparsi di indirizzarlo al dispositivo opportuno per la sua ricezione. Al

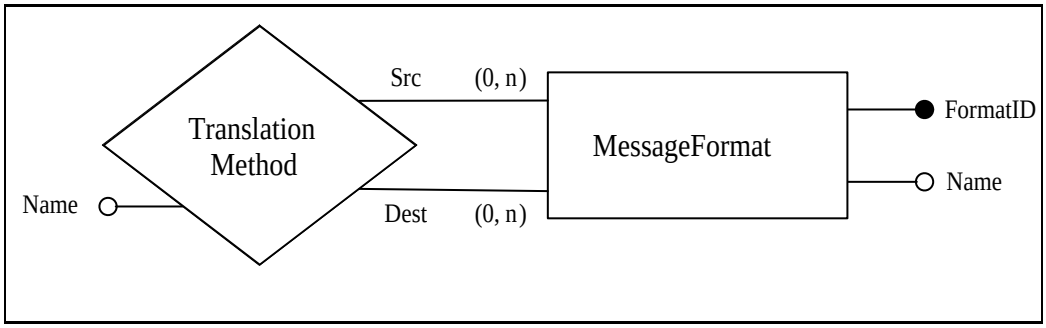


Figura 2.24: Metodi di traduzione

momento della gestione del messaggio arrivato ad un certo UCI, il sistema deve poter riconoscere la sua struttura, per instradarlo correttamente in base alle informazioni che contiene.

La struttura del messaggio comprende prima di tutto un identificatore, che permette di distinguere ogni messaggio in modo univoco.

Per assicurare l'unicità di questa stringa di numeri, l'identificatore è stato generarto nella forma seguente: $A.B.C.D$, dove:

- A è il numero di millisecondi dal 01/01/1970 al momento di creazione del messaggio,
- B è il numero di millisecondi dal momento della creazione a quello di arrivo,
- C è l'UCI di chi spedisce il messaggio,
- D è il formato del messaggio.

Per esempio, supponiamo che Bob in data x decida di mandare un messaggio di posta elettronica ad Alice e che il sistema lo riceva al tempo y. L'identificatore avrà, quindi, la seguente forma 986898902955.30.Bob.2, dove A assume il valore 986898902955, cioè il numero di millisecondi tra

l'01/01/1970 e x, B = A-y diventa ad esempio 30, per il parametro C possimamo supporre che l'UCI di Bob sia "Bob", ed infine D rappresenta la costante identificativa del dipo di informazione trasportata, che nel caso dell'*email* è 2.

L'entità **Message** (Figura. 2.25), che modella la struttura dei messaggi, è caratterizzata dai seguenti attributi:

- *MessageID* l'identificatore del messaggio.
- *Size* la dimensione del messaggio.
- *Type* il formato del messaggio (text, email, image...).
- *Sensitivity* la sensitività dell'informazione trasportata (HIGH, LOW, NORMAL). Affinché un dato messaggio possa essere consegnato su un dato dispositivo, è necessario che la sua sensitività sia maggiore del livello di sicurezza del dispositivo.
- *Encrypted* indica se il messaggio è stato crittato o meno.
- *Compressed* indica se il messaggio è stato compresso o meno.
- *Sign* la firma digitale del mittente.
- *TimeStamp* la data di spedizione.
- *Content* il contenuto del messaggio.

Oltre a queste informazioni, ogni spedizione di un messaggio ha bisogno di un mittente, di un destinatario, che può essere anche un gruppo e delle credenziali che l'utente vuole condividere.

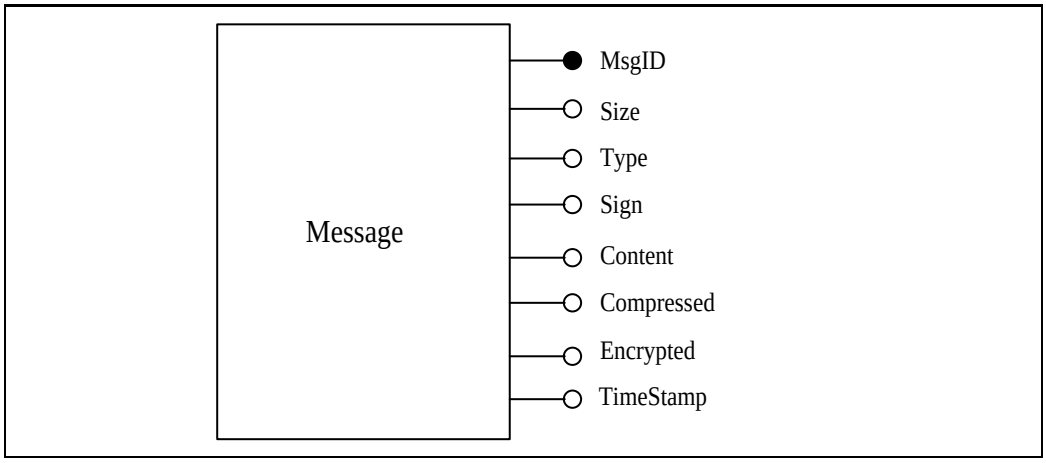


Figura 2.25: Messaggio

Le entità **Message**, **User** e **Group** e **UserCard** sono, quindi, legate dalle associazioni uno a molti *UserSendsMsg*, *UserReceivesMsg*, *MsgHasUserCard*, *GroupReceivesMsg* (Figura 2.26).

Anche il sistema può mandare dei messaggi per comunicare con l'utente. I messagggi di sistema sono notifiche, in formato testo, che vengono spedite all'utente per avvisarlo di una situazione di errore verificatasi, o dell'avvenuta consegna del messaggio.

2.3.3 Schema ER sui messaggi

In questa sezione è stato descritto, analizzando ogni sua componente, lo schema ER sui messaggi, illustrato in Figura 2.26.

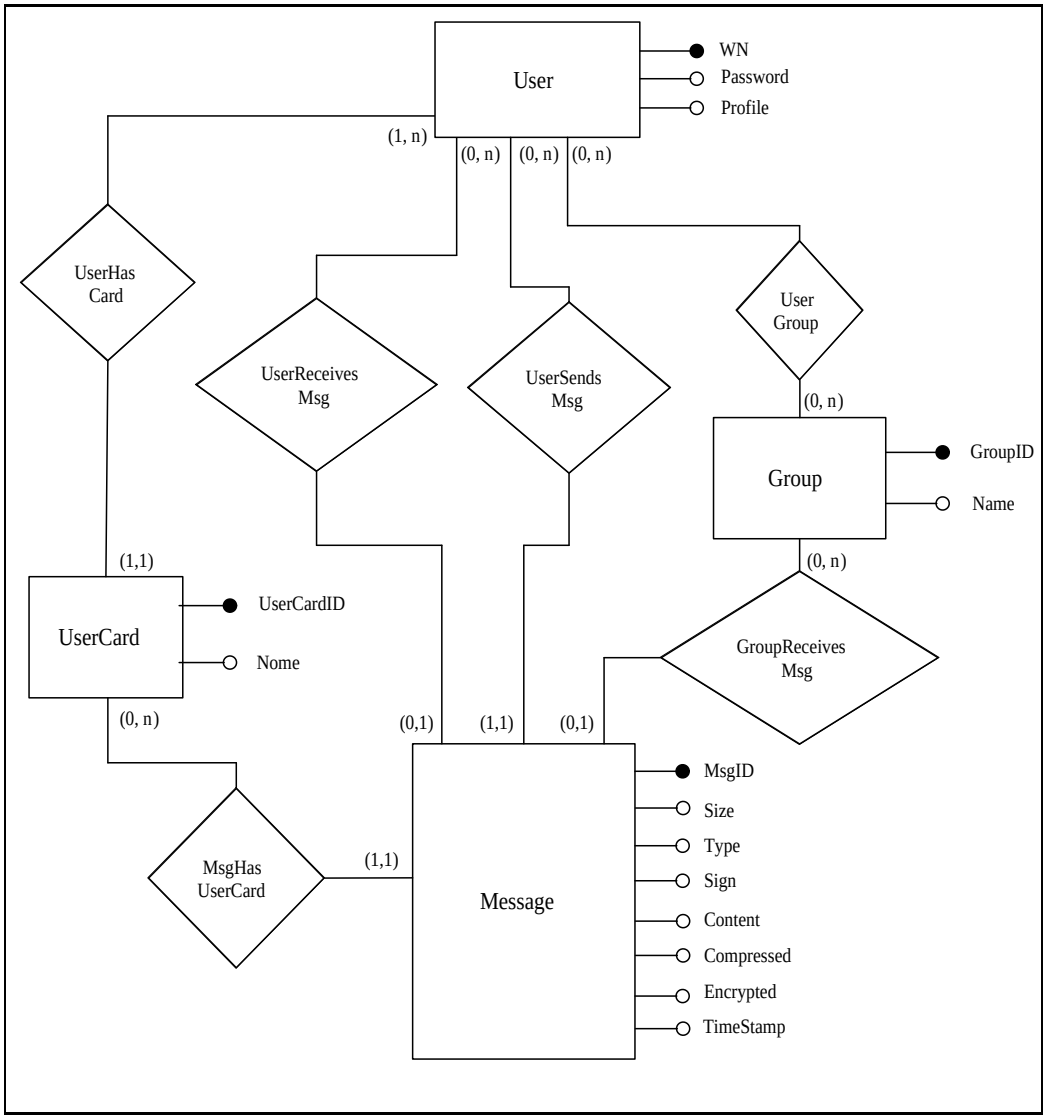


Figura 2.26: Schema dei messaggi

Capitolo 3

Architettura

A partire dal modello dei dati presentato nel precedente capitolo, si è creata una base di dati per la memorizzazione di tutte le informazioni degli utenti, utili al sistema per la gestione degli identificatori universali per la comunicazione. Infatti, la redirezione dei messaggi, arrivati ad un utente, sui dispositivi opportuni, dipende dal profilo che l'utente stesso si è delineato e che è mantenuto nella base di dati.

Lo scopo di questo capitolo è spiegare il comportamento interno del motore, basato su regole attive, sviluppato per instradare un messaggio, in arrivo ad un determinato UCI, sui dispositivi adatti a riceverlo. Prima di entrare nel dettaglio della descrizione della gestione dei messaggi, è opportuno illustrare l'interazione del sistema sviluppato con l'ambiente esterno. Gli utenti, utilizzando le interfacce implementate, possono accedere alla base di dati per definire il loro profilo e, in seguito, spedire messaggi agli altri utenti usando i dispositivi dichiarati al sistema. Quando arriva un messaggio per un determinato UCI, il sistema accede alla base di dati per prelevare il pro-

filo del destinatario del messaggio e, servendosi di servizi esterni offerti dal *provider*, tenta di effettuare la consegna. Nella Sezione 3.1 sono descritte le operazioni che utenti e amministratori possono effettuare. Nella Sezione 3.2 viene analizzato il comportamento del motore all'arrivo di un messaggio ad un particolare UCI. Nella Sezione 3.3 è stata presentata l'analisi dei parametri di sistema introdotti nella sezione precedente. Nella Sezione 3.4 sono stati descritti alcuni esempi di instradamento dei messaggi in arrivo all'UCI di un utente in base alle regole che ha definito.

3.1 Architettura del sistema

L'interazione del sistema con l'ambiente esterno presuppone che gli utenti, attraverso apposite interfacce, possano definire il loro profilo, utilizzato poi dal sistema per instradare i messaggi rievuti (Figura 3.1).
Affinché il sistema possa gestire i messaggi in arrivo ad un utente, è necessario che quest'ultimo sia stato registrato e che tutti i sui dati siano stati definiti. La registrazione di un nuovo utente avviene tramite l'amministratore di sistema, che ha il compito di aggiungere o rimuovere utenti, inserire nuovi modelli o nuovi tipi di dispositivi che sono usciti in commercio o cancellarne altri non più in uso, e configurare variabili di sistema per un migliore funzionamento del motore.
Quando l'amministratore del sistema introduce un nuovo utente, registra il suo UCI nella base di dati, insieme alla sua parola chiave e al profilo appositamente creato. A questo punto, attraverso una apposita interfaccia, in cui l'utente inserisce il suo UCI e la sua parola chiave, avviene l'dentificazione

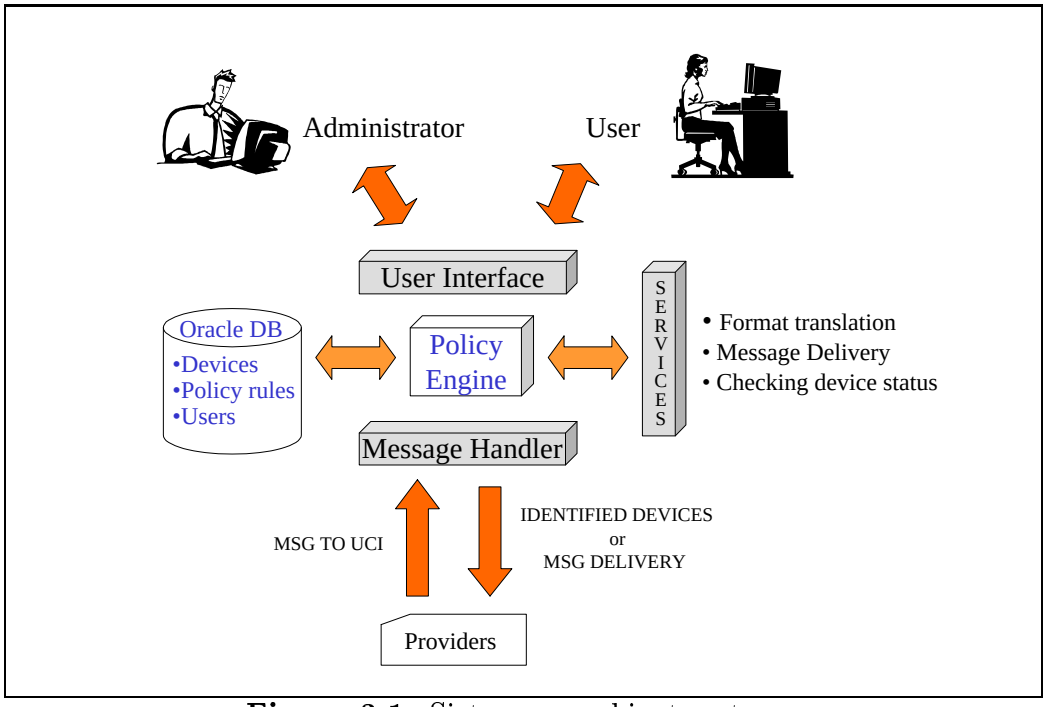


Figura 3.1: Sistema e ambiente esterno

dell'utente da parte del sistema, che mostra all'utente il menù delle operazioni possibili. L'utente può, ora, dichiarare i suoi dispositivi, definire le proprie credenziali, creare dei gruppi e stabilire le regole che il sistema deve applicare al momento dell'arrivo di un messaggio all'UCI, per indirizzare i messaggi sui dispositivi dell'utente (Figura 3.2). Tutte queste informazioni vengono memorizzate nella base di dati e consultate dal sistema ogni volta che l'utente o la gestione di un messaggio lo richiedono.

Per migliorare l'efficienza del sistema, si è pensato di mantenere nel profilo dell'utente le informazioni che direttamente sono coinvolte nell'instradamento e nella consegna del messaggio. I dispositivi utilizzati dall'utente, quelli indicati per la ricezione delle notifiche e le regole definite, sono tutte informazioni contenute nel profilo. In questo modo, quando un messaggio

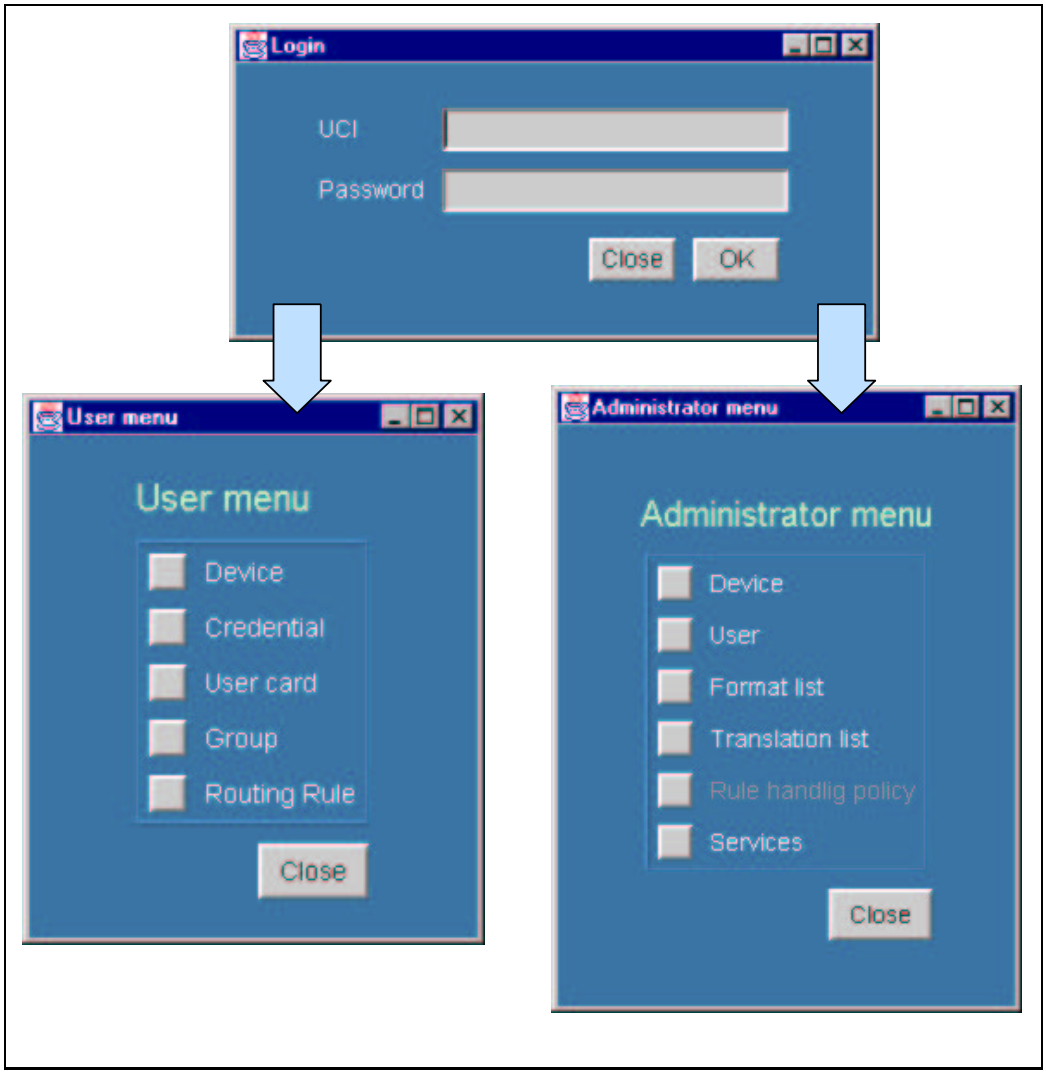


Figura 3.2: Menu utente e amministratore

arriva all'UCI di un utente, il sistema, con una sola interrogazione alla base di dati, può conoscere tutte le informazioni di cui ha bisogno.

Quando un utente vuole spedire un messaggio ad un'altro utente, è sufficiente che indichi il suo UCI, disinteressandosi completamente dei dispositivi che quest'ultimo possiede. Prima che il messaggio venga effettivamente inoltrato, il sistema crea un nuovo messaggio caratterizzato da una struttura predefinita, che il sistema è in grado di riconoscere all'arrivo del messaggio stesso all'UCI destinatario.
Il sistema memorizza tutti i messaggi ricevuti in un *buffer*, li indirizza uno alla volta ai dispositivi dell'utente destinatario, in base alle regole da lui definite, e spedisce poi una notifica che descrive la consegna avvenuta. Se si verificano errori durante la valutazione delle regole, nella selezione di dispositivi compatibili o nella consegna effettiva del messaggio, il sistema avvisa l'utente della situazione verificatasi, gestendola a seconda della causa.
Questo comportamento del sistema permette all'utente di essere sempre a conoscenza di tutti i messaggi che gli sono stati inviati e, eventulmente, a quali dispositivi sono stati consegnati.
Se non si verificano situazioni di errore, la consegna del messaggio ai dispositivi avviene grazie all'interazione del sistema con i fornitori di servizi esterni per le telecomunicazioni, attraverso un'interfaccia che permette di applicare le politiche di instradamento definite dagli utenti stessi.

Per una corretta e migliore gestione del messaggio, il sistema sviluppato potrebbe essere esteso definendo servizi esterni che il *provider* sarebbe poi in grado di offrire. Poiché, le regole, che un utente può definire per la gestione dei messagi in arrivo al prorpio UCI, prevedono anche una condi-

zione sullo stato in cui si trova un dispositivo, il *provider* potrebbe offrire un servizio che permette questa verifica. Ad esempio, se l'utente definisce una condizione del tipo "se il telefono è occupato, inoltra i messaggi sul telefonino", al sistema serve un servizio esterno per controllare lo stato del dispositivo in un determinato istante. Tale servizio potrebbe, inoltre, essere utilizzato al momento della consegna effettiva del messaggio per verificare se il dispositivo è disponibile a riceverlo. Un servizio esterno, utilizzato per controllare se la consegna è avvenuta correttamente, è utile al sistema per spedire all'utente una notifica che indica su quali dispositivi il messaggio è stato realmente ricevuto. Nel caso non esistano dispositivi dell'utente adatti a ricevere un messaggio di un dato formato, il sistema può decidere di modificare il formato di quest'ultimo in base ai metodi di traduzione supportati dal *provider* stesso, sempre rispettando le regole definite dall'utente. Potrebbero essere, quindi, implementate delle funzioni del tipo *fax2image* oppure *sms2email*, per dare maggiori possibilità di riuscita della consegna di un messaggio di un dato formato. Infatti, grazie a questi metodi di traduzione, è possibile consegnare un messaggio anche su dispositivi che non gestiscono direttamente il suo formato.

In questa sezione è stata presentata l'architettura del sistema analizzando le varie parti che interagiscono con il sistema stesso. Ogni gestione di un messaggio richiede, quindi, che il sistema acceda alla base di dati per prelevare il profilo che l'utente si è delineato attraverso le interfacce. Utilizzando i servizi supportati dal *provider*, il sistema può tentare di consegnare il messaggio ai dispositivi dell'utente, valutando le regole impostate, e spedire una notifica all'utente stesso per descrivere quanto avvenuto.

3.2 Architettura del motore a regole

Il motore del sistema progettato può essere spiegato analizzando le sue componenti principali nel dettaglio. L'architettura presentata in questa sezione è illustrata in Figura 3.3, dove sono state messe in evidenza le componenti che ricoprono un ruolo attivo e significativo all'interno del motore. Una breve descrizione di queste componenti, che interagiscono al suo interno, può essere utile per spiegare il funzionamento della gestione dei messaggi. Tali componenti sono, quindi, indicate di seguito, delineando le loro caratteristiche basilari.

- *MainBuffer*. Area di memoria utilizzata per memorizzare temporaneamente i messaggi in arrivo al sistema.
- *MessageDaemon*. Demone che preleva, ad intervalli di tempo di pochi millisecondi, un messaggio dal *MainBuffer* affinché possa essere gestito.
- *UCIRouter*. (Gestore dei messaggi). Valuta le regole valide, se ce ne sono, e, servendosi di servizi esterni supportati da un *provider*, controlla lo stato della consegna. Se la consegna avviene correttamente, spedisce una notifica all'utente per informarlo.
- *ErrorRouter*. (Gestore degli errori). A seconda dell'errore verificatosi, decide se spedire una notifica, se rispedire il messaggio, o memorizzarlo nella base di dati.
- *DelayedMessageBuffer*. Area di memoria per memorizzare temporaneamente i messaggi inoltrati dal gestore degli errori.

- *DelayDaemon.* Demone che preleva, ad intervalli di tempo di pochi millisecondi, un messaggio dal *DelayedMessageBuffer* per poi memorizzarlo nel *MainBuffer* per essere gestito.

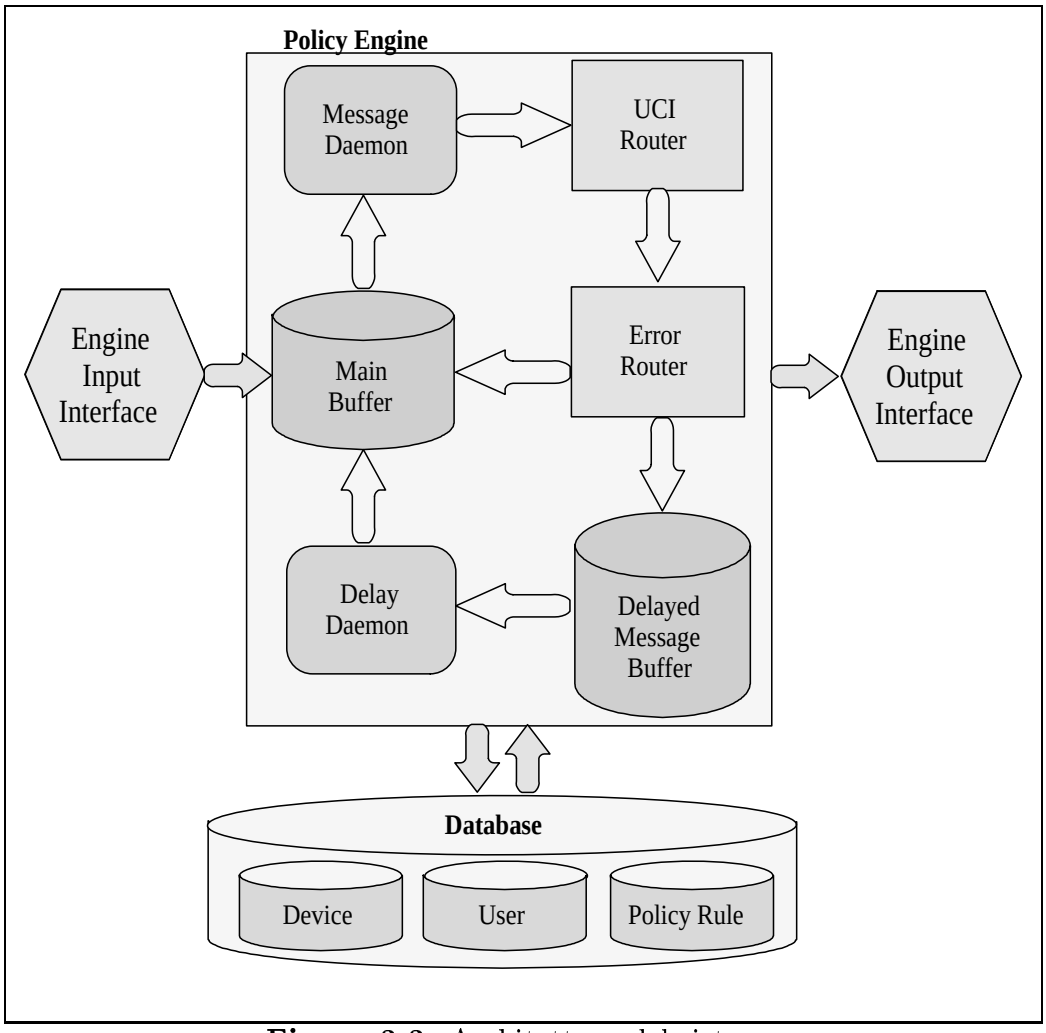


Figura 3.3: Architettura del sistema

In Figura 3.3 sono state raffigurate anche le due intefacce che permetto al sistema di interagire con la rete che supporta i dispositivi utilizzati dagli utenti per la comunicazione. Quando viene spedito un messaggio, attraverso la rete, viene attivato il servizio di instradamento e l'*EnginInputInterface*

```
Function MessageDaemon()
    while (true)
        if(MainBuffer is not empty)
          Get a msg from the MainBuffer;
          if(the msg is new)
            Set the identifier of the msg;
          Get the receiver's profile retrieved from the database;
          Start the UCIRouter with the receiver's profile and the msg;
        Wait for PICK_MESSAGE_DELAY milliseconds;
```

Figura 3.4: Algoritmo di attivazione dell'UCIRouter

rende la struttura del messaggio interpretabile dal sistema. A questo punto, il motore seleziona tra i dispositivi dell'utente quelli adatti alla ricezione del messaggio, applicando le regole definite dal destinatario.
L'*EngineOutputInterface* utilizza i servizi esterni forniti dal *provider* per effettuare la consegna del messaggio ai dispositivi.

Il sistema sviluppato si occupa della ricezione e dell' instradamento dei messaggi, e della gestione degli errori. Una descrizione approfondita dell'architettura del motore può essere data analizzando il funzionamento delle singole componenti che lo costituiscono e spiegando come queste cooperano.

Il sistema progettato raccoglie e memorizza tuttti i messaggi in arrivo ai diversi UCI in un *buffer* (*MainBuffer*), dove rimangono in attesa di essere gestiti. Dopo un certo numero di millisecondi, parametro di sistema stabilito dall'amministratore, un messaggio viene prelevato dal *MainBuffer* e viene attivato il gestore effettivo del messaggio (*UCIRouter*), come descritto dall'algoritmo in Figura 3.4.

L'*UCIRouter*, grazie alla particolare struttura del messaggio, riconosce l'UCI del mittente e quello del destinatario. In questo modo, è in grado di

caricare in memoria il profilo del ricevente, con tutte le informazioni utili per l'indirizzamento del messaggio ai dispositivi indicati dall'utente. Sempre attraverso la struttura del messaggio, l' *WNRouter* controlla il suo formato e la sensitività dell'informazione che trasporta. Servendosi di queste informazioni, trova la lista degli identificatori dei dispositivi compatibili con il messaggio. La compatibilità di un messaggio con un dispositivo è legata sia al suo formato sia alla sua sensitività. Infatti, un messaggio è compatibile con un dispositivo, quando:

- Il formato, che un dispositivo è in grado di ricevere direttamente, è lo stesso del messaggio, oppure, esiste un metodo di traduzione che permette al dispositivo di gestire il messaggio tradotto.

- Il livello di sicurezza del dispositivo è più grande del livello di sensitività del messaggio. Questa restrizione è stata introdotta per problemi di sicurezza; se un dispositivo è in grado di garantire un certo livello di riservatezza delle informazioni ricevute, non può gestire messaggi che richiedono un livello maggiore.

Se l'utente non ha impostato regole, il messaggio viene consegnato al primo dispositivo della lista in grado di ricevere tale tipo di messaggio. Se l'utente ha definito delle regole, l' *UCIRouter* le valuta fino a trovare la prima regola valida e, in base a quest'ultima, stabilisce quali tra i dispositivi indicati nella lista precedentemente selezionata, sono indicati per la consegna del messaggio. Attraverso le regole è possibile decidere se inoltrare il messsaggio a tutti i dispositivi ottenuti dalla seconda selezione, o, semplicemente, di bloccare la spedizione del messaggio al primo dispositivo per cui la consegna

è andata a buon fine.

Una volta che la consegna è avvenuta, se l'utente ha predisposto dei dispositivi per ricevere notifiche, l' *UCIRouter* ne spedisce una all'UCI dell'utente per avvisarlo su quali dispositivi il messaggio è stato consegnato. Se la notifica non può essere consegnata, per esempio perchè lo stato dei dispositivi indicati è di non disponibilità, il messaggio di sistema viene inoltrato un certo numero di volte (anche questo è un parametro impostato dall'amministratore). Se la notifica risulta inconsegnabile, viene persa. Infatti, la sua ricezione non risulta indispensabile, dato che la consegna del messaggio vero e proprio è avvenuta correttamente.

Il fatto che la consegna non avvenga a causa della mancanza di disponibilità dei dispositivi, selezionati per la ricezione del messaggio, ha portato alla definizione di un attributo che indica il tempo di vita del messaggio stesso (*Time To Live*, *TTL*), ossia il numero di volte che questo può essere inoltrato per tentare la consegna. Infatti, potrebbe succedere che durante il nuovo inoltro del messaggio, almeno uno dei dispositivi scelti assuma uno stato di disponibilità per la ricezione del messaggio. Per esempio, se si tenta di consegnare un fax ma il telefono in quell'istante è occupato, instradando nuovamente il messaggio può succedere che poco dopo il telefono risulti libero e la consegna del fax può avvenire. Il parametro *TTL*, definito sia per i messaggi normali che per quelli di notifica, viene impostato dall'amministratore in base alle esigenze del sistema e gestito dall'*ErrorRouter*. Finché il valore del *TTL* è diverso da zero, l'*ErrorRouter* lo decrementa e memorizza il messaggio nel *DeleyedMessageBuffer* per essere inoltrato nuovamente. Quando il *TTL* diventa nullo, il messaggio, se non è una notifica, viene me-

morizzato nella base di dati per dare la possibilità all'utente di recuperarlo successivamente. Se il messaggio è una notifica, come già detto, viene persa.

Gli errori che l'*ErrorRuoter* si trova a gestire, spesso sono causati da una scorretta definizione delle regole, o dall'impossibilità di individuare i dispositivi, dovuta alla possibilità di esprimere diversi tipi di condizioni. Infatti:

- La possibilità di definire regole per reinstradare i messaggi può causare la formazione di cicli. Per esempio, se Bob stabilisce che tutti i messaggi, in arrivo al suo fax di casa, devono esser ricevuti su quello dell'ufficio, e viceversa, si crea un ciclo che non permette al sistema di identificare dispositivi disponibili.

- La possibilità di esprimere differenti tipi di condizioni può portare all'impossibilità di gestire determinati formati di messaggi. Per esempio, se Bob definisce la condizione:

  1. Inoltra tutti i fax sul PC,

  Se il *provider* non fornisce dei metodi opportuni per tradurre il formato fax in un formato supportabile dal PC, come ad esempio *fax2email* o *fax2image*, i fax non potranno mai essere ricevuti dall'utente.

Il motore è in grado di accorgersi della presenza degli errori sopra descritti. In questo modo, il sistema ha la possibilità di di avvisare l'utente affinché modifichi le regole che ha impostato. Di seguito viene spiegato come questi errori sono stati gestiti.

Al fine di verificare la presenza di cicli, l' *UCIRouter*, per ogni messaggio

che sta gestendo, mantiene una lista dei dispositivi specificati nell'azione della regola. Se si verifica un ciclo, l' *UCIRouter* se ne accorge in quanto, trovandosi a gestire nuovamente la regola, uno stesso dispositivo è già presente nella lista. Quando il sistema si accorge della presenza di un ciclo, la lista viene svuotata e viene inserito nuovamente solo il dispositivo che ha permesso di rilevare la presenza del ciclo. Ogni volta che il sistema incontra il dispositivo che rappresenta il punto di partenza del ciclo stesso, questo procedimento viene ripetuto e un contatore, che indica il numero di cicli incontrati, viene incrementato fino ad un valore massimo espresso dalla costante *MAX_CYCLE_RULE*, il cui valore viene impostato dall'amministratore di sistema. Quando il contatore supera questo limite, allora non si procede ulteriormente nell'esecuzione delle regole, e viene attivato l'*ErrorRouter*, che ha il compito di memorizzare il messaggio nella base di dati e di spedire una notifica all'utente. In questo modo, l'utente, avvisato dell'errore verificatosi, può ridefinire le regole in modo corretto.
Se anche per la notifica si verifica un ciclo, l'utente non potrà ricevere nessun tipo di avvertimento, ma controllando i messaggi memorizzati nella base di dati si può rendere conto della presenza di un errore.
Se le condizioni definite dall'utente non permettono la selezione di alcun dispositivo, tra quelli compatibili, allora il messaggio viene memorizzato dall'*ErrorRouter* nel *DeleyedMessageBuffer*, finché il valore dell'attributo TTL non risulta nullo. Questo in quanto può succedere che una data condizione non sia verificata in un determinato istante, ma che risulti valida poco dopo. Ad esempio, se la condizione temporale è del tipo "se tra le ore 12.00 e le ore 15.00 arriva un messaggio sul telefono di casa, inoltralo sul

```
Function DelayDaemon()
    while (true)
        while (DelayedMessageBuffer is not empty)
            Get a msg from the DelayedMessageBuffer;
            Put this Msg into the MainBuffer;
        Wait for PICK_MESSAGE_DELAY milliseconds;
```

Figura 3.5: Algoritmo del DelayDaemon

cellulare", alle 12.00 la regola diventa vera.
Se l'*UCIRouter* non trova dispositivi compatibili con il messaggio, l'*ErrorRouter* lo memorizza nella base di dati, spedendo una notifica opportuna.

Ogni volta che l'*ErrorRouter* memorizza un messaggio nel *DelayedMessageBuffer*, affinché venga reinstradato, il tempo di vita del messaggio viene decrementato. Il *DelayDaemon*, ad intervalli di pochi millisecondi, definiti attraverso la costante *PICK_MESSAGE_DELAY*, preleva un messaggio dal *DelayedMessageBuffer* e lo memorizza nel *MainBuffer* per cominciare la nuova gestione. Il funzionamento del *DelayDaemon* è spigato attraverso l'algoritmo in Figura 3.5.

In questo capitolo, si è analizzata l'architettura del sistema; gli algoritmi utilizzati, per la gestione dei messaggi e per l'implementazione del motore a regole, sono stati descritti in [2].

3.3 Analisi dei parametri di sistema

Nel sistema sono presenti dei parametri che l'amministratore può modificare tramite interfaccia per migliorare il funzionamento del motore e avere una

maggiore efficienza del sistema.

Nella descrizione dell'architettura del sistema e della gestione del messaggio, si è spiegato che quando arriva un messaggio al sistema, questo viene memorizzato nel *MainBuffer* dove rimane per *PICK_MESSAGE_DELAY* millisecondi. Trascorso questo tempo, il MessageDaemon prende un messaggio dal MainBuffer e attiva l'*UCIRouter* per la sua gestione. Se la consegna non può avvenire, per esempio perché non ci sono dispositivi disponibili alla ricezione del messaggio fra quelli valutati compatibili dal sistema, l'*ErrorRouter* memorizza il messaggio nel *DelayBuffer*, dove rimane per *MESSAGE_DELAY* millisecondi prima di essere memorizzato nel *MainBuffer* per tentare una nuova gestione. Il numero di volte che uno stesso messaggio può essere gestito è indicato dal parametro *TTL*. Se la causa dell'impossibilità di consegnare un messaggio è la presenza di un ciclo causato dalle regole definite dall'utente, l'*UCIRouter* si trova a valutare per lo stesso messaggio le stesse regole ogni volta che il ciclo si ripresenta. Il numero massimo di cicli che si possono verificare per un messaggio è stabilito dal parametro *MAX_CYCLE_RULE*.

Il valore del parametro *PICK_MESSAGE_DELAY* deve essere impostato tenendo presente che un valore troppo alto, rispetto al numero di messaggi in arrivo al sistema, causerebbe una saturazione del *MainBuffer*. Infatti, se supponiamo che arrivino tre messaggi al secondo e che il tempo di permanenza nel *MainBuffer* sia per ogni messaggio di due secondi, dopo dieci secondi ci sarebbero nel *MainBuffer* venticinque messaggi ancora da gestire. Infatti, il *MessageDaemon*, prelevando un messaggio alla volta ogni due secondi, causa la formazione di una coda sempre crescente di messaggi in attesa di essere gestiti.

Poiché, ogni gestione di un messaggio comporta l'accesso alla base di dati per il caricamento in memoria del profilo del ricevente, il gestore della base di dati potrebbe trovarsi a gestire più connessioni di quante ne supporta effettivamente. Per ridurre il tempo di giacenza dei messaggi da gestire nel *MainBuffer*, si potrebbe pensare ad una soluzione di accesso centralizzato alla base di dati. In questo modo, l'intervallo di tempo, che passa prima che il *MessageDaemon* prelevi un messaggio dal *MainBuffer*, non dipende più dal numero di messaggi in arrivo al sistema e non può succedere che una sbagliata impostazione di tale variabile porti il gestore della base di dati in una situazione di errore a causa delle troppe connessioni.

Al contrario, un valore elevato per il parametro *MESSAGE_DELAY* è utile in quanto potrebbe verificarsi che un dispositivo, trascorso questo intervallo di tempo, diventi disponibile per la ricezione del messaggio che prima aveva rifiutato. Per esempio, se si tenta di consegnare una email ma la mailbox è piena, instradando nuovamente il messaggio dopo un tempo ragionevole, può succedere che l'utente abbia eliminato alcuni messaggi e che, quindi, la consegna possa avvenire correttamente. In questo caso non si può verificare la saturazione del *DeleyedMessageBuffer*, in quanto il *DelayDaemon*, quando accede al *DeleyedMessageBuffer*, preleva tutti i messaggi contenuti per metterli nel *MainBuffer*.

Il parametro *TTL*, indicando il numero di volte che un messaggio può essere gestito, incide sulle risorse del sistema. Infatti, se i messaggi in arrivo al sistema sono molti, un valore elevato di tale costante porterebbe il motore a dover gestire i nuovi messaggi arrivati e diverse volte i messaggi che richiedono una nuova gestione. Questa situazione potrebbe portare ad una

saturazione dei buffer a causa dei troppi messaggi da gestire. Tuttavia, un valore troppo basso non sarebbe sufficiente affinché una data regola diventi valida permettendo la consegna del messaggio.

Dato che le regole definite dall'utente potrebbero provocare la formazione di cicli, permettere che questi si ripetano molte volte, impostando un valore troppo alto per il parametro *MAX_CYCLE_RULE*, posticipa la gestione dell'errore, ritardando la spedizione di una notifica all'utente per avvisarlo di modificare le regole impostate.

I valori di questi parametri devono essere, quindi, impostati dall'amministratore tenendo conto del numero di utenti e della quantità di messaggi spediti.

3.4 Esempi di instradamento dei messaggi

Analizzando l'architettura del motore, è stato spiegato come viene gestito un messaggio. Di seguito vengono mostrati degli esempi di regole che un utente può definire e come un messaggio in arrivo ad un UCI viene gestito in base alle regole stabilite dal destinatario. Per dare una definizione formale delle regole è stato usato il linguaggio di definizione delle politiche di instradamento definito in [2].
Supponiamo di avere i due utenti Bob e Alice, univocamente identificati dai loro UCI, che in questo esempio coincidono con i loro nomi. Bob possiede 4 dispositivi indicati di seguito:

- Il telefono di casa, identificato tramite il numero 100

- Il telefono dell'ufficio, identificato tramite il numero 101
- Il telefono cellulare, identificato tramite il numero 102
- Il PC, identificato tramite il numero 103
- Il palmare, identificato tramite il numero 104
- La stampante, identificata tramite il numero 105

I dispositivi che Alice possiede sono, invece, i seguenti:

- Il telefono di casa, identificato tramite il numero 200
- Il telefono dell'ufficio, identificato tramite il numero 201
- Il telefono cellulare, identificato tramite il numero 202
- Il PC, identificato tramite il numero 203
- Il FAX, identificato tramite il numero 204

Ciacuno di questi dispositivi può supportare determinati formati in base ai modelli di appartenenza. In generale, un telefono fisso può ricevere solamente *phoncall*, mentre un telefono cellulare può ricevere anche *SMS*. Un PC è in grado di gestire direttamente, ossia senza l'applicazione di metodi di traduzione, *email*, *video*, *immagini* e *audio*, *testo*, *HTML* e *XML*. Il palmare è in grado di ricevere messaggi con gli stessi formati supportati dal PC, mentre il FAX può ricevere solo fax. Ad una stampante è possibile consegnare messaggi di formato *testo* o *immagini*. Applicando opportuni metodi per tradurre il formato di un messaggio in un altro, i dispositivi possono

ricevere anche messaggi con un formato differente da quelli normalmente gestiti. Per esempio, un PC attraverso il metodo di traduzione *fax2image* è abilitato a ricevere anche i fax. Questi servizi devono essere offerti dal *provider* e in questo esempio supponiamo che quest'ultimo metodo di traduzione sia effettivamente supportato.
Supponiamo, inoltre, che sia Bob che Alice abbiano definito delle regole, in questo modo possiamo descrivere cosa succede quando i due utenti si spediscono dei messaggi.

Esempio 1 *Supponiamo che Alice desideri inoltrare tutte le chiamate provenienti da Bob sul suo telefono cellulare.*

Regola1

ON MESSAGE ARRIVAL AT UCI(*Alice*)
WHENEVER SENDER HAS UCI(*Bob*) AND
MSG CONTENT TYPE IS *phonecall*
ROUTE TO *202*;

Se Bob chiama Alice, Alice riceverà la chiamata di Bob sul suo telefono portatile. Se il telefono cellulare di Alice risulta occupato, la telefonata non può avvenire. Se un altro utente telefona ad Alice, le regola non viene eseguita e la telefonata sarà inoltrata al primo dispositivo compatibile tra quelli di Alice. In questo caso, l'inoltro del messagio verrà bloccato al primo dispositivo per cui la consegna è avvenuta correttamente.

Esempio 2 *Supponiamo, ora, che Alice desideri inoltrare tutte le chiamate provenienti da Bob sul suo telefono cellulare e, se questo è occupato, la telefonata debba essere inoltrata al telefono dell'ufficio, se questa avviene tra le 9.00 e le 18.00, altrimenti sul telefono di casa.*

Regola1

ON MESSAGE ARRIVAL AT UCI(*Alice*)
WHENEVER SENDER HAS UCI(*Bob*) AND
MSG CONTENT TYPE IS *phonecall*
ROUTE TO *202*;

Regola2

ON MESSAGE ARRIVAL AT DEVICE(*202*)
WHENEVER SENDER HAS UCI(*Bob*) AND
MSG CONTENT TYPE IS *phonecall*
ROUTE TO *201*;
TIME BETWEEN *9.00* TO *18.00*;

Regola3

ON MESSAGE ARRIVAL AT DEVICE(*202*)
WHENEVER SENDER HAS UCI(*Bob*) AND
MSG CONTENT TYPE IS *phonecall*
ROUTE TO *20O*;
TIME BETWEEN *18.00* TO *9.00*;

In base alle tre regole sopra definite, quando Bob chiama, se il telefono cellulare di Alice è occupato, Alice può ricevere la telefonata in ufficio durante le ore di lavoro, oppure a casa durante il resto della giornata. Se anche il telefono su cui avviene l'inoltro risulta non disponibile, la comunicazione tra Bob e Alice non potrà avvenire.

Esempio 3 *Supponiamo, inoltre, che Alice desideri ricevere tutti i fax, in arrivo al suo UCI, sul FAX oppure sul PC servendosi del metodo di traduzione fax2image. Nel caso i messaggi consegnati al FAX siano caratterizzati*

da un alto livello di sensitività, Alice desidera ricevere una notifica sul suo telefono cellulare.

Regola1

ON MESSAGE ARRIVAL AT UCI(*Alice*)
WHENEVER MSG CONTENT TYPE IS *fax*
ROUTE TO ANY *203* USING *fax2image*;*204*;

Regola2

ON MESSAGE ARRIVAL AT DEVICE(*204*)
WHENEVER MSG SENSITIVITY LEVEL IS *HIGH*
NOTIFY TO *202*;

Se Alice riceve un fax, questo verrà consegnato al primo dispositivo disponibile tra quelli indicati nella regola. Nel caso si tratti di un messaggio di una certa importanza, se la consegna avviene sul FAX, una notifica viene spedita ad Alice per avvisarla dell'arrivo del messaggio. Se il Fax, in un certo istante, non è disponibile a ricevere il messaggio arrivato, questo sarà convertito in un'immagine, assumendo quindi un formato compatibile con il PC. Se anche il PC non risulta disponibile per la ricezione del fax, questo, dopo un certo intervallo di tempo, verrà gestito uovamente per tentare la consegna. Infatti, potrebbe succedere che, durante il reinoltro, almeno uno dei dispositivi indicati nella regola diventi disponibile per la ricezione. Se ciò non avviene, dopo essere stato gestito un numero di volte pari al valore massimo consentito dal sistema, il fax sarà memorizzato nella base di dati e verrà spedita una notifica ad Alice per avvisarla della mancata consegna del messaggio. Alice può, in seguito, prelevare il fax dalla base di dati.
Supponiamo ora che Bob abbia definito le sue regole e che Alice ed altri

utenti vogliono comunicare con lui. Nell'esempio seguente viene presentata un'altra situazione che si può verificare.

Esempio 4 *Supponiamo che Bob desideri ricevere tutte le email in arrivo al suo UCI sia sul PC che sul palmare. Tuttavia, se il mittente del messaggio appartiene al gruppo degli "indesiderati", l'email non viene consegnata al palmere, e una notifica viene spedita a Bob sul suo telefono cellulare per avvisarlo.*

Regola1

ON MESSAGE ARRIVAL AT UCI(*Bob*)
WHENEVER MSG CONTENT TYPE IS *email*
ROUTE TO ALL *103*; *104*;

Regola2

ON MESSAGE ARRIVAL AT DEVICE(*104*)
WHENEVER SENDER IS IN GROUP *indesiderati*
REJECT;
NOTIFY TO *102*;

Bob potrebbe aver creato il gruppo "indesiderati" per evitare di ricevere posta elettronica da persone che normalmente spediscono email di grandi dimensioni. Se un utente è stato inserito da Bob in questo gruppo, tutte le sue email saranno consegnate al PC, ma rifiutate dal palmare. Su richiesta di Bob, gli viene comunque spedita una notifica sul telefono cellulare, in modo da sapere che un messaggio di posta elettronica è stato rifiutato dal suo palmare. Anche se la regola è stata definita in modo che una email debba essere consegnata sia al palmare che al PC, se uno dei due dispositivi non è disponibile a ricevere il messaggio, ma la consegna è avvenuta ad

almeno uno dei due, il nuovo inoltro non viene gestito e non viene spedita alcuna notifica.

Vediamo ora cosa succede se Bob definisce delle regole che portano alla formazione di cicli.

Esempio 5 *Supponiamo che Bob desideri ricevere tutte le immagini sul palmare. Se il palmare non ha abbastanza memoria, queste vengono inoltrate alla stampante. Nel caso in cui la stampante non risulti disponibile, Bob vuole che tutti messaggi in arrivo alla stampante vengano inoltarti al PC. Inoltre, se arrivano messaggi al PC da parte di Alice durante il fine settimana, questi devono essere inoltrati al palmare.*

Regola1

ON MESSAGE ARRIVAL AT UCI(*Bob*)
WHENEVER MSG CONTENT TYPE IS *image*
ROUTE TO *104*;

Regola2

ON MESSAGE ARRIVAL AT DEVICE(*104*)
WHENEVER STATUS IS *low_memory* AND
MSG CONTENT TYPE IS *image*
ROUTE TO *105*;

Regola3

ON MESSAGE ARRIVAL AT DEVICE(*105*)
WHENEVER STATUS IS *not_ avaliable*
ROUTE TO *103*;

Regola4

ON MESSAGE ARRIVAL AT DEVICE(*103*)

WHENEVER SENDER HAS UCI(*Alice*)
ROUTE TO *104*; DURING WEEKEEND;

Se tutte le regole definite da Bob dovessero risultare valide contemporaneamente, un'immagine spedita da Alice non verrebbe consegnata a nessun dispositivo di Bob a causa della formazione di un ciclo. Infatti, le immagini spedite da Alice a Bob durante il fine settimana, non potendo essere ricevute sul palmare per mancanza di memoria, ne dalla stampante, momentaneamente non disponibile, finirebbero per essere inoltrate nuovamente al palmare. Se il numero di volte che un messaggio può essere gestito e il tempo intercorso tra due differenti gestioni del messaggio, sono stati scelti in modo opportuno, potrebbe succedere che il ciclo si risolve e la consegna del messaggio avviene correttamente. Affinché ciò avvenga, dovrebbe verificarsi che durante il nuovo inoltro del messaggio, Bob liberi la memoria del suo palmare. In questo modo, l'immagine spedita da Alice nel fine settimana può essere consegnata. Un'altra situazione che potrebbe accadere in attesa della nuova gestione del messaggio, corrisponde al ripristino del funzionamento della stampante. In questo caso, Bob non riceverebbe l'immagine spedita da Alice direttamente sul suo palmare, ma la troverebbe, comunque, stampata. Finché le regole definite da Bob non diventano valide nello stesso istante, Bob non si accorge dell'errore. Se si verifica un ciclo e tutte le regole continuano a rimanere valide, il sistema, dopo aver tentato la gestione del messaggio un certo numero di volte, lo memorizza nella base di dati e spedisce una notifica a Bob per avvisarlo della situazione di errore che si è verificata.

Capitolo 4

Struttura delle interfacce e dell'applicativo

Attraverso la descrizione dell'architettura del sistema, si è potuto vedere che l'applicazione consta di tre parti principali: la base di dati, il motore e gli utenti. Le prime due parti sono state ampliamente analizzate nei due capitoli precedenti, mentre l'interazione degli utenti con il sistema avviene attraverso opportune interfacce illustrate in questo capitolo. La struttura delle interfacce è stata organizzata in modo da separare le operazioni possibili per gli utenti, quelle necessarie all'amministratore e alcune interfacce di sistema per poter controllare il funzionamento del motore.

Per agevolare l'uso e le estensioni future dell'implementazione, viene fornito un manuale utente che spiega come poter utilizzare le interfacce per controllare come viene gestita la consegna di un messaggio, sui dispositivi definiti da un utente in base alle regole che ha stabilito. Infatti, il mittente ed il destinatario di un messaggio devono essere utenti appartenenti al siste-

ma, caratterizzati da un certo profilo. Solo partendo da questa situazione, è possibile servirsi delle interfacce di sistema per verificare lo stato della consegna di un messaggio. È necessario, quindi, spiegare come un amministratore può inizializzare il sistema e come un utente può definire il proprio profilo.

Tutte le operazioni, che l'amministratore e gli utenti possono compiere, e la gestione del messaggio sono state implementate organizando il progetto in diversi *package*, definiti in base alle componenti che hanno assunto un ruolo significativo e attivo all'interno del progetto UCI. Per esempio, considerando l'interazione del sistema con l'ambiente esterno, sono stai creati *package* differenti per le interfacce, la gestione della base di dati e l'instradamento dei messaggi. Ciascuna di queste parti è stata poi ulteriormente divisa definendo nuovi *package*, contenenti classi per la gestione di oggetti specifici.

Il capitolo è organizzato come segue. Nella Sezione 5.1 viene mostrata la maschera per l'autenticazione dell'utente. Nella Sezione 5.2 viene spiegato come l'amministratore può registrare e cancellare nuovi utenti, controllare i tipi e i modelli dei dispositivi e definire variabili di sistema. Nella Sezione 5.3 vengono descritte le operazioni che un utente può compiere sui dispositivi e come si possono definire credenziali, gruppi e regole. Nella Sezione 5.4 viene illustrato il funzionamento delle interfacce di sistema per controllare la gestione e la consegna di un messaggio. Nella Sezione 5.5 viene spiegato come il progetto è stato strutturato.

4.1 Maschera di *Login*

Una volta avvenuta la compilazione del progetto, l'applicazione viene lanciata e compare una prima interfaccia per permettere all'utente di autenticarsi. Non essendoci, inizialmente, utenti registrati nel sistema, l'unico utente presente di default nella base di dati è l'amministratore. Il suo profilo viene creato automaticamente dal sistema e contiene solamente il suo UCI, *admin*, utilizzato anche per la parola chiave. Usando queste informazioni, l'amministratore può accedere al sistema e, attraverso opportune interfacce, registrare nuovi utenti.

Affinché avvenga l'identificazione, un utente deve specificare il suo UCI e la sua *password*, come illustrato in Figura 4.1. Il sistema confronta le infor-

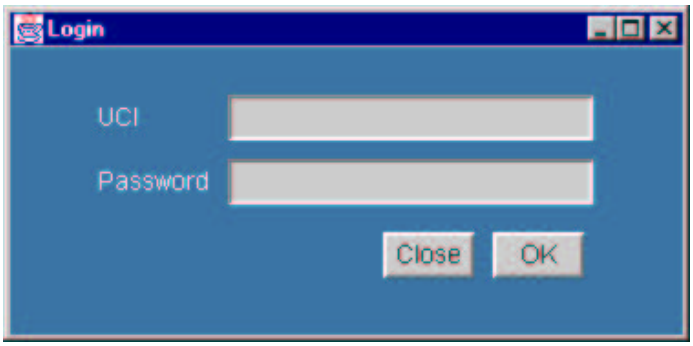


Figura 4.1: Maschera di identificazione

mazioni date dall'utente tramite interfaccia con quelle presenti nella base di dati, se corrispondono compare la maschera di menù. Nel caso l'utente non inserisca i dati corretti, viene mostrato un opportuno messaggio di errore. Attraverso il bottone *close* è possibile chiudere l'applicazione.

4.2 Struttura delle interfacce dell'amministratore

La struttura delle interfacce implementate per l'amministratore è stata creata in modo da poter selezionare l'operazione desiderata utilizzando i vari menù. Attraverso l'interfaccia pricipale è possibile vedere i menù sui dispositivi e sugli utenti. Questa interfaccia permette, inoltre, di accedere a maschere di servizio per facilitare l'utilizzo del sistema e migliorarne l'efficienza. Le funzionalità e lo scopo di queste ultime operazioni sono descritte nella Sezione 5.4. La struttura delle interfacce per l'amministratore è illustrata in Figura 4.2 e analizzata di seguito mostrando le maschere principali.

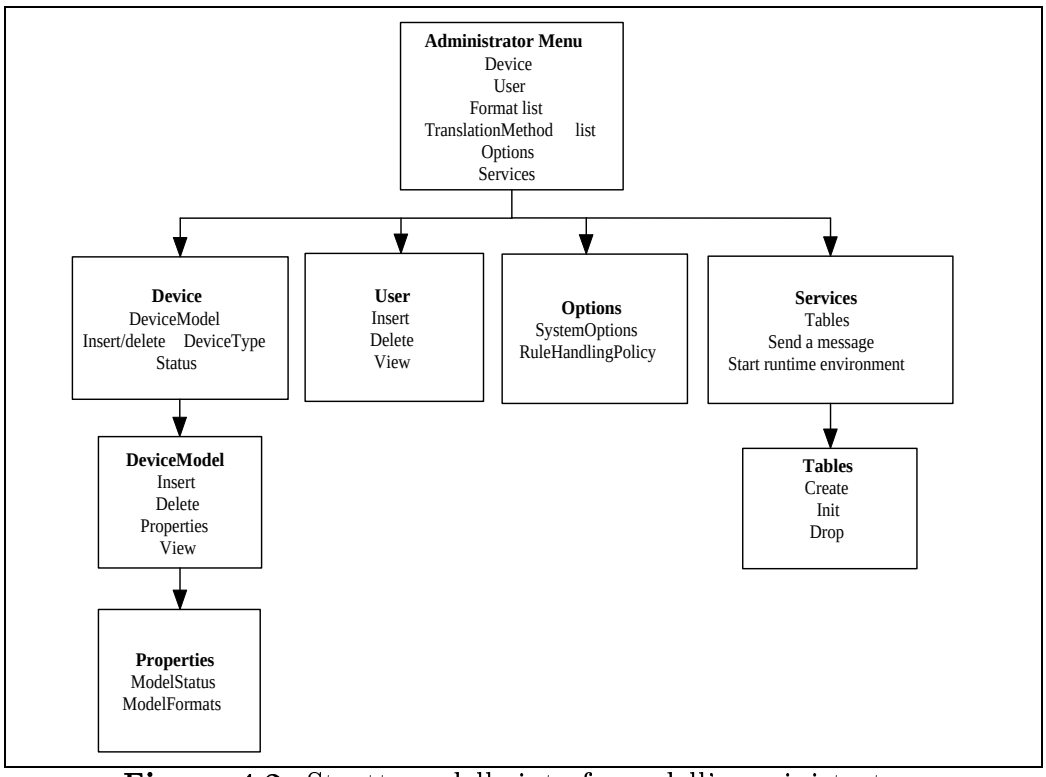


Figura 4.2: Struttura delle interfacce dell'amministratore

Una volta avvenuta l'identificazione al sistema, viene aperta una maschera di menù con l'elenco dei tipi di operazioni che è possibile effettuare (Figura 4.3).

L'amministratore può inserire nuovi tipi di dispositivi e i corrispondenti

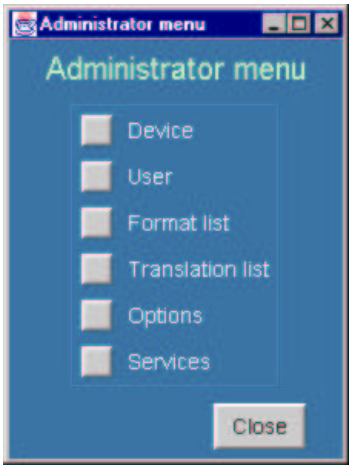


Figura 4.3: Menu dell'amministratore

modelli che si intende considerare. Ad ogni modello è necessario associare i formati dei messaggi che è in grado di gestire e gli stati in cui un dato dispositivo, appartenente al modello, si può trovare. Per fare questo, l'amministratore ha la possibilità di controllare l'elenco degli stati, dei formati, e dei metodi di traduzione supportati dal sistema (Figura 4.4). Se l'amministratore decide di cancellare un tipo di dispositivo, vengono automaticamente eliminati anche tutti i modelli di quel tipo e i dispositivi che vi appartengono. La cancellazione dei dispositivi di un dato modello può essere provocata anche dalla scelta dell'amministratore di eliminare quel particolare modello.

Dopo aver inizializzato la base di dati, l'amministratore può inserire i nuovi utenti. Durante la registrazione, viene creato il profilo dell'utente, inizial-

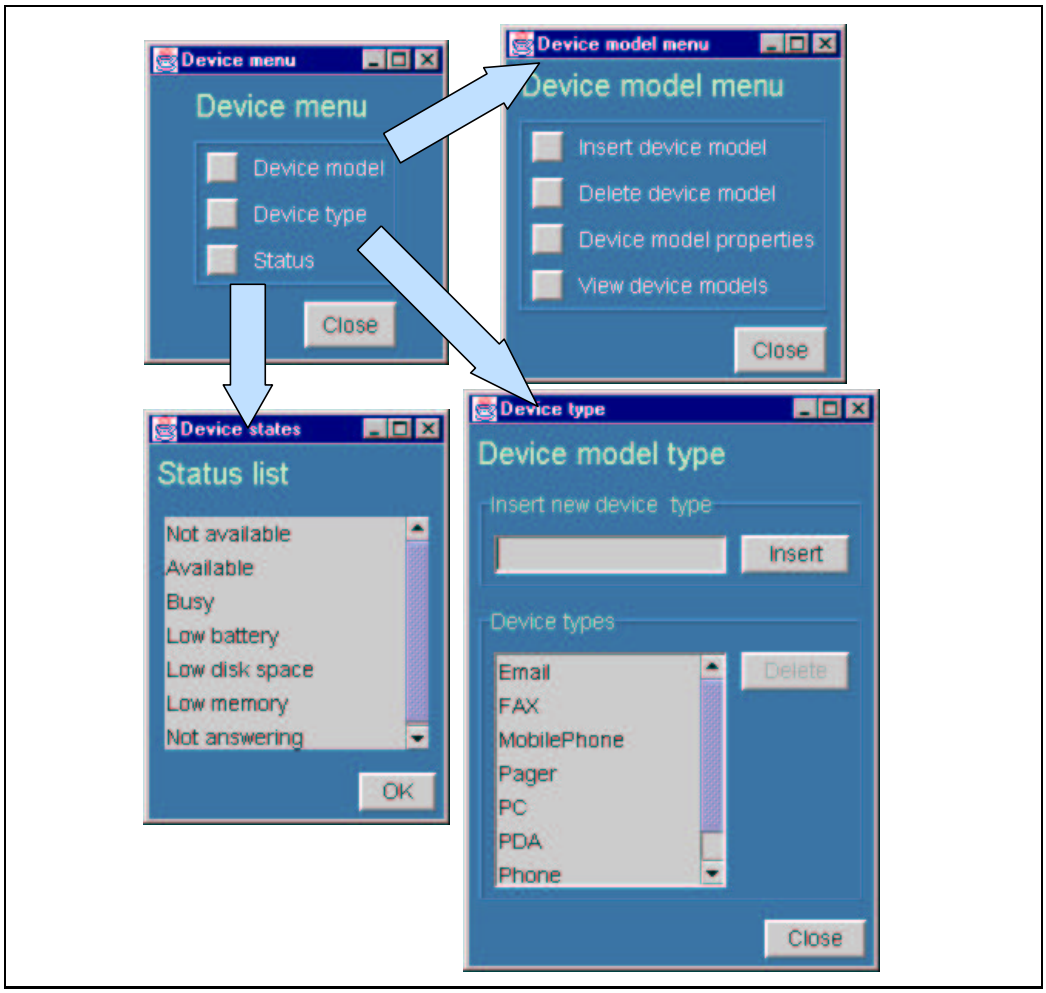


Figura 4.4: Tipi, modelli e stati dei dispositivi

mente formato solo dal suo UCI, e la parola chiave, che viene criptata e memorizzata nella base di dati insieme al profilo e all'UCI stesso. Per criptare la parola chiave si sono utilizzati i metodi supportati da Java. Una futura estensione potrebbe riguardare l'utilizzo di un sistema per assicurare un livello superiore di protezione dei dati.

In qualunque momento l'amministratore può controllare gli utenti appartenenti al sistema e le credenziali che hanno definito (Figura 4.5). In realtà, per assicurare la veridicità delle informazioni personali spedite dagli utenti con i messaggi, potrebbe essere l'amministratore a richiedere la definizione delle credenziali al momento della registrazione nel sistema. Ogni modifica delle credenziali comporterebbe però il controllo da parte dell'amministratore, compito che potrebbe risultare troppo dispendioso in tempo. Ad esempio, le credenziali riguardo al lavoro potrebbero essere cambiate spesso dalla maggior parte degli utenti. Momentaneamente si è, quindi, deciso di lasciare la definizione delle credenziali agli utenti stessi, ma una possibile soluzione potrebbe essere quella di definire le credenziali con il controllo di enti autorizzati.

Se l'amministratore cancella un utente dal sistema, viene eliminato anche il suo profilo, ossia tutti i suoi dispositivi, le sue credenziali, le regole che ha impostato, eventuali suoi messaggi memorizzati nella base di dati poiché la consegna non era avvenuta correttamente, i gruppi che ha creato e viene eliminato dai gruppi in cui è presente.

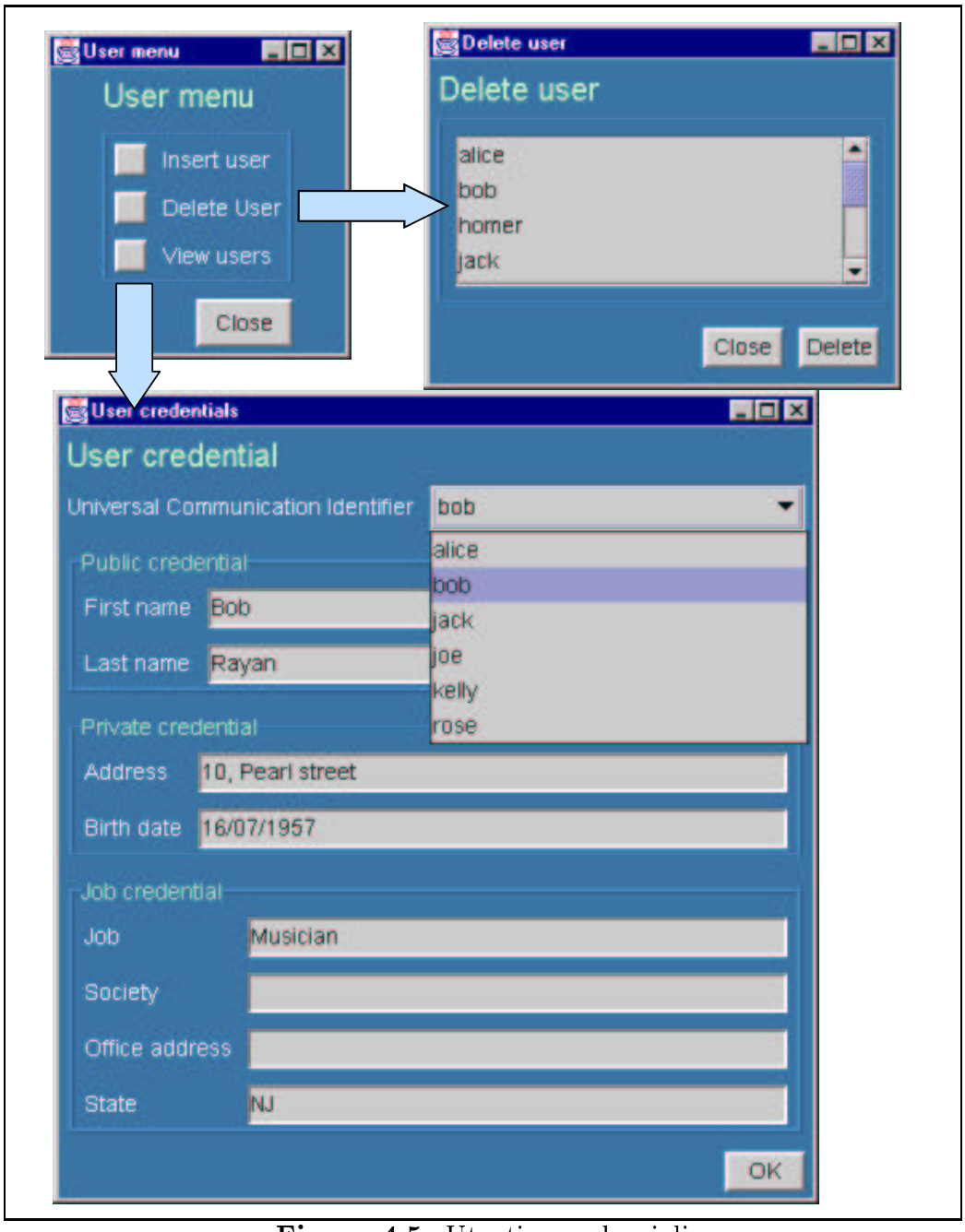


Figura 4.5: Utenti e credenziali

4.3 Struttura delle interfacce dell'utente

La struttura delle interfacce dell'utente è organizzata in modo da raggruppare tutte le operazioni in base alle entità soggette a tali modifiche. Partendo dal menù pricipale, l'utente ha, quindi, la possibilità di sceglire tra i menù sottostanti, fino alla maschera per l'esecuzione vera e propria dell'operazione. In Figura 4.6 viene rappresentata questa struttura e, di seguito, viene descritta mostrando alcune delle interfacce implementate.

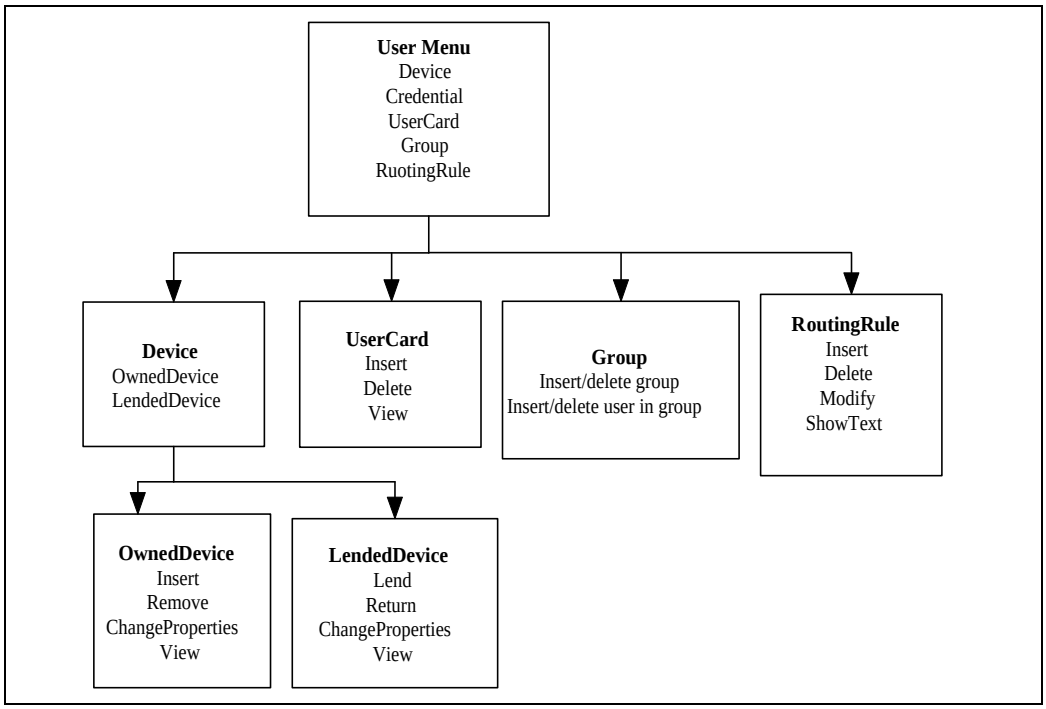


Figura 4.6: Struttura delle interfacce dell'utente

Dopo l'autenticazione dell'utente al sistema, compare il menù principale che permette all'utente stesso di dichiarare i propri dispositivi, definire le proprie credenziali, creare gruppi e stabilire le regole di instradamento dei messaggi in arrivo al proprio UCI (Figura 4.7).

Le operazioni che un utente può compiere sui propri dispositivi sono diffe-

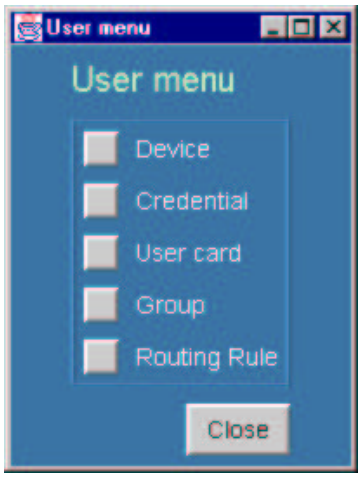


Figura 4.7: Menu dell'utente

renti a seconda che l'utente sia il proprietario o semplicemente l'utilizzatore di dispositivi. I dispositivi utilizzati effettivamente dall'utente sono quelli di cui è proprietario, e che non ha prestato ad altri utenti, e quelli che ha ricevuto in prestito. Se l'utente è l'utilizzatore dei dispositivi, può inserine di nuovi, cancellarli, cambiare le propiretà o semplicemente vederli. Le proprietà dei dispositivi che possono essere modificate sono le seguenti: il *nickname*, dato dall'utente ad un suo dispositivo per distinguerlo dagli altri; il livello di sicurezza, che il dispositivo può offrire per assicurare una certa riservatezza dei messaggi ricevuti; il livello di priorità, utilizzato dal sistema per sapere a quale dispositivo tra quelli dell'utente dare la precedenza nella consegna del messaggio; ed infine l'utilizzo del dispositivo in caso di notifica. Se l'utente è in possesso di dispositivi ricevuti in prestito, può controllare da chi gli sono stati prestati, per quanto tempo e con che capacità. Se l'utente è il proprietrio, può decidere di prestare i dispositivi (se non li ha già prestati ad altri), porre termine ad un prestito, modificare

la capacità con cui il dispositivo era stato prestato, cambiare l'intervallo di validità del prestito, o controllare a chi era stato prestato un dispositivo e con quali modalità (Figura 4.8).

Attraverso i dispositivi che può utilizzare, un utente spedisce messaggi di diversi formati, in genere uniti alle proprie credenziali. Infatti, le credenziali che l'utente definisce sono utili al momento della spedizione di un messaggio, quando l'utente vuole comunicare le proprie informazioni personali al ricevente. Le credenziali, per essere spedite insieme al messaggio, devono essere raggruppate in *usercards*, in modo da poter spedire più tipi di informazione contemporaneamente. Attraverso l'interfaccia per l'eliminazione delle *usercards*, è possibile vedere quali credenziali appartengono ad una determinata *usercard* e quali dati contengono (Figura 4.9).

La possibilità di creare diversi gruppi di utenti dà l'opportunità di stabilire filtri sui mittenti dei messaggi in arrivo all'UCI e di spedire lo stesso messaggio a più persone contemporaneamente. Le condizioni che si possono ora definire sui gruppi prevedono semplicemente il controllo sull'appartenenza del mittente al gruppo specificato nella regola. In futuro, nella definizione delle condizioni sui gruppi, si potrebbero considerare gruppi dinamici, per esempio stabilire un filtro su tutte le persone che lavorano per una data società. È possibile creare e eliminare gruppi, oppure inserire o togliere utenti da un gruppo, come mostrato in Figura 4.10. Attraverso l'interfaccia di eliminazione degli utenti da un gruppo è possibile controllare, per ciascun gruppo definito, quali utenti vi appartengono.

L'utente può stabilire, eliminare o modificare le regole di instradamento dei messaggi. Per effettuare le operazioni relative alle regole è stato creato un

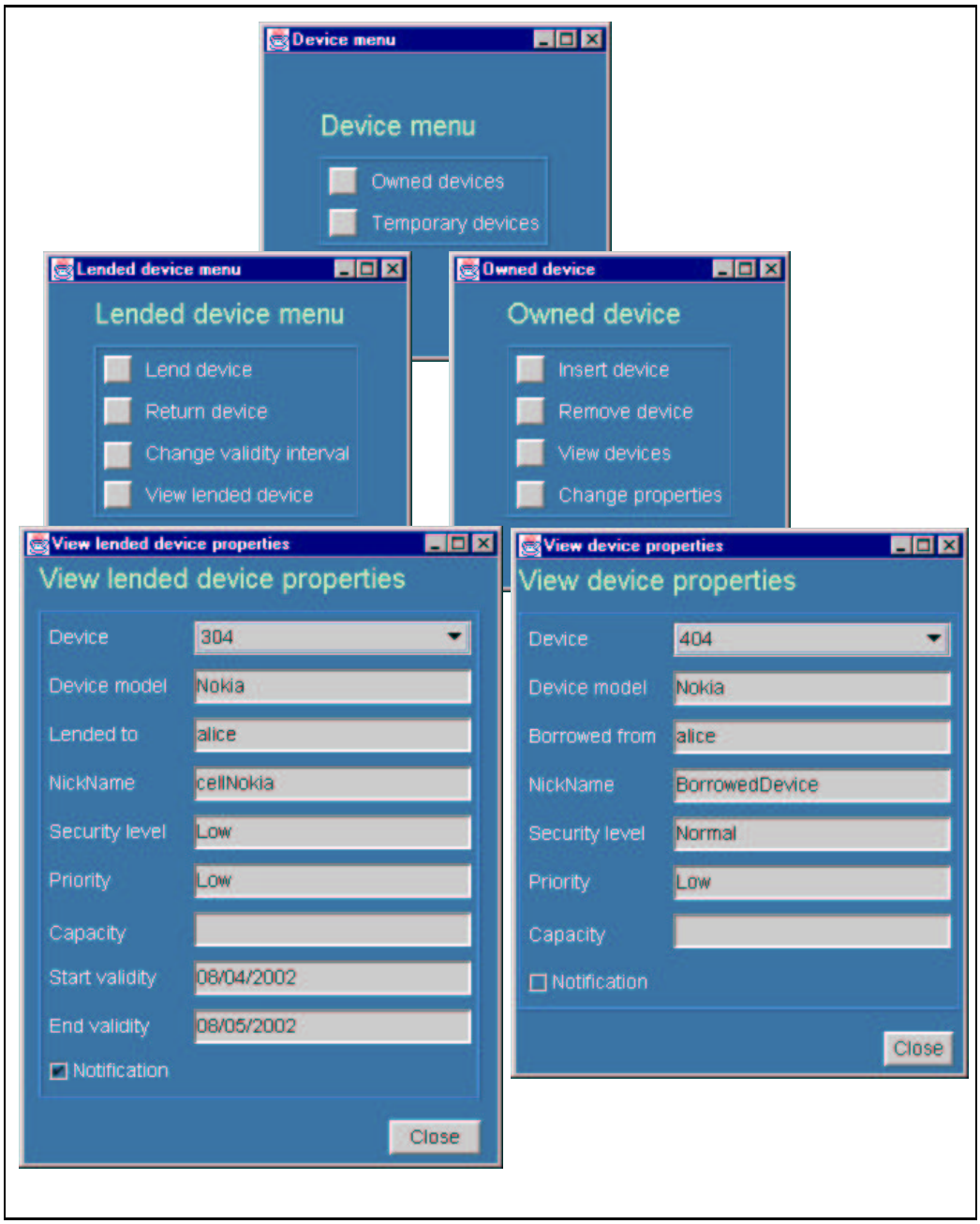


Figura 4.8: Operazioni possibili sui dispositivi

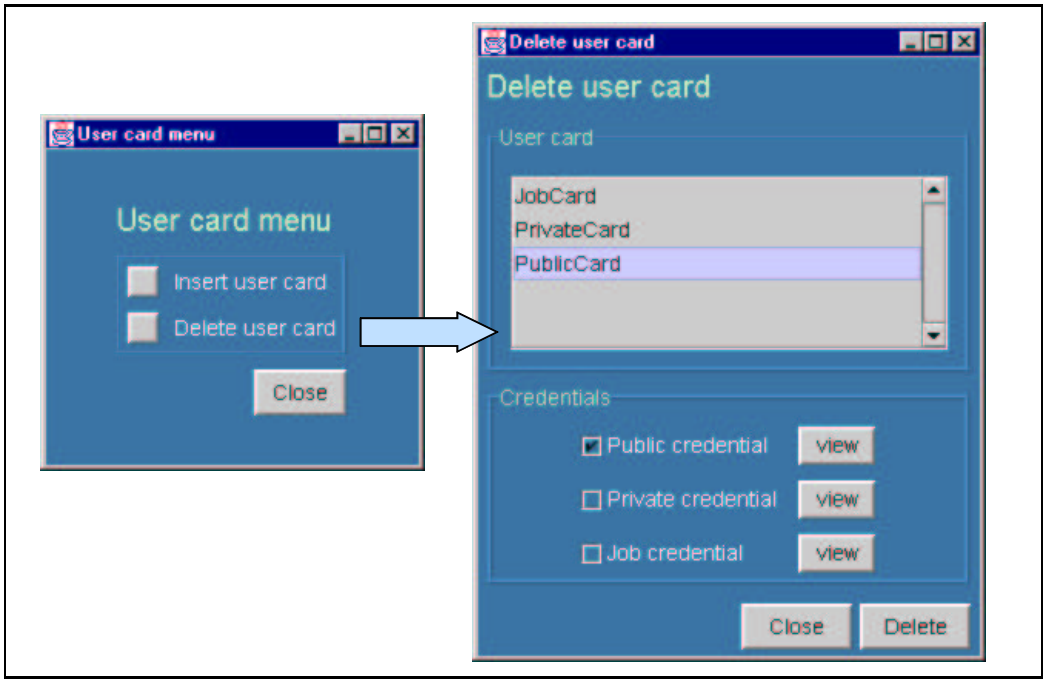


Figura 4.9: Definizione delle UserCards

menù dal quale si può accedere all'interfaccia di definizione (Figura 4.11). Le regole possono essere definite, in base all'arrivo del messaggio sull'UCI o su un determinato dispositivo. Se la regola è stata impostata sull'UCI dell'utente, le condizioni che si possono definire sono sul mittente, sul tempo o sul contenuto del messaggio. Se la regola è stata stabilita per un particolare dispositivo, è possibile definire una condizione anche sul suo stato o sul suo livello di sicurezza. Se la condizione definita si è verificata, la consegna del messaggio avverrà in base all'azione specificata e, se è stato indicato su quali dispositivi deve essere spedita la notifica, questa verrà inoltrata. La validità della regola dipende, inoltre, dall'intervallo di validità specificato (Figura 4.12). Ogni condizione impostata all'interno di una regola può essere anche negata. Per controllare la correttezza di una regola è possibile vederne anche la formulazione in formato testo.

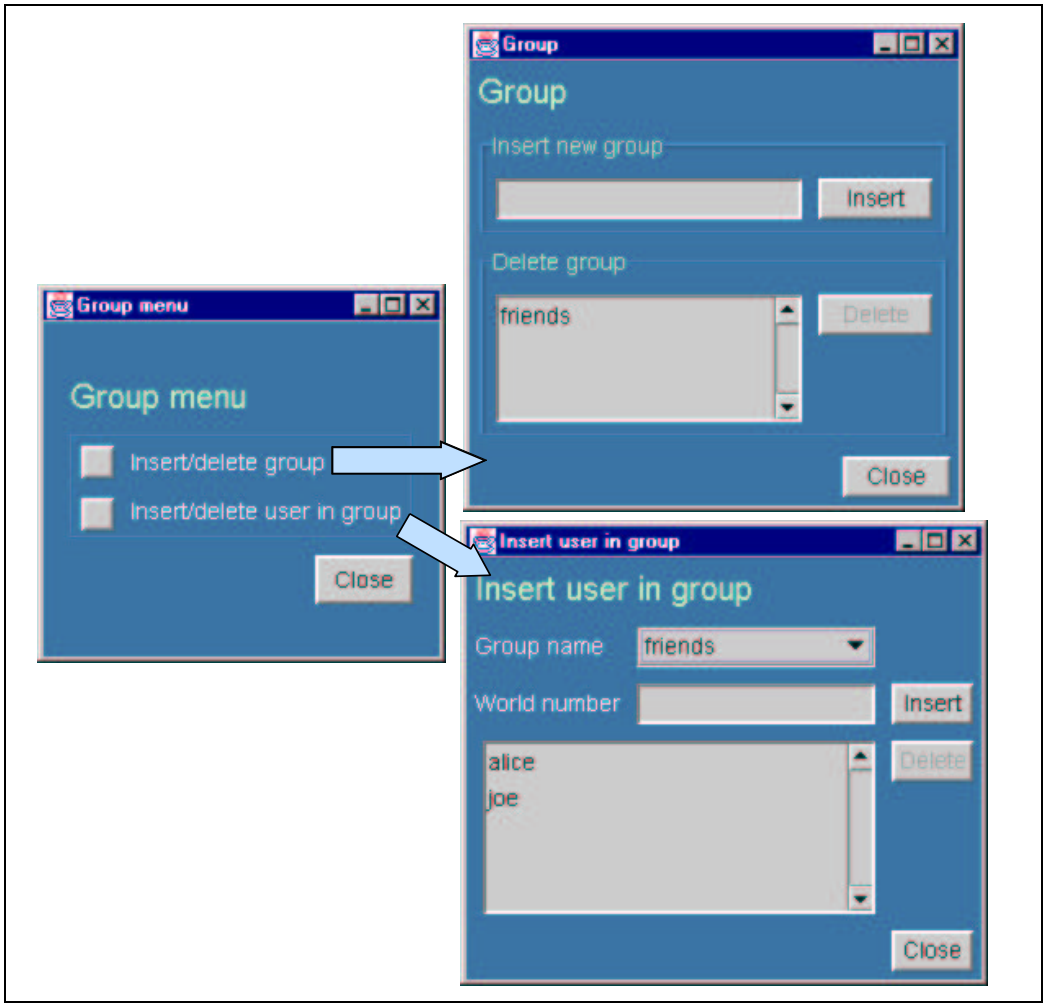


Figura 4.10: Definizione dei gruppi

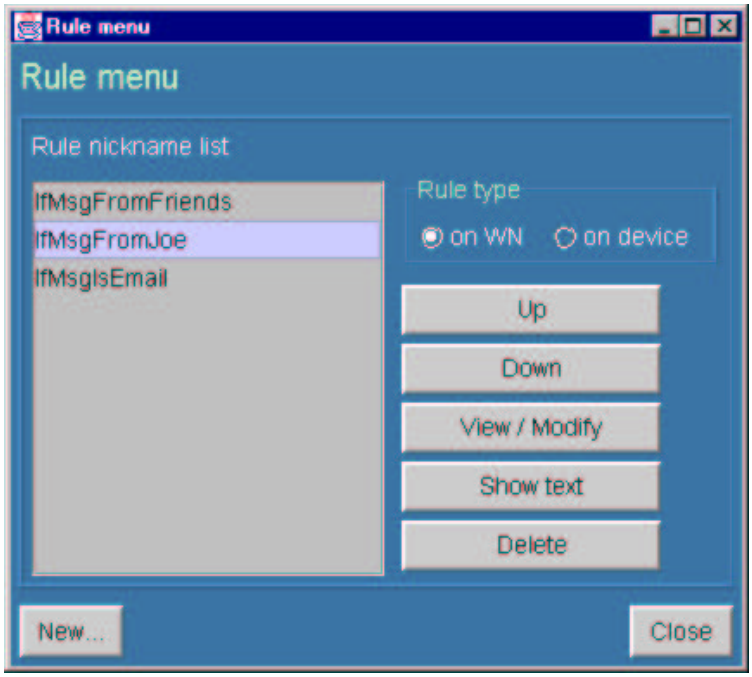


Figura 4.11: Menu delle regole

Quando l'utente finisce di modificare il suo profilo e chiude il suo menù principale, viene riaperta la maschera di *login* per permettere ad un nuovo utente di accedere al sistema. Se l'utente non chiude tutti i menù prima di quello principale, non può lasciare la sessione e un messaggio di errore lo avvisa del problema verificatosi.

4.4 Maschere di sistema

Le operazioni, che riguardano direttamente la gestione e la funzionalità del sistema implementato, sono accessibili attraverso il menù dell'amministratore. Sono stati creati due menù separati per distinguere le operazioni di impostazione dei parametri di sistema, che realmente interessano l'amministratore per avere un migliore funzionamento del motore, e le opearzioni

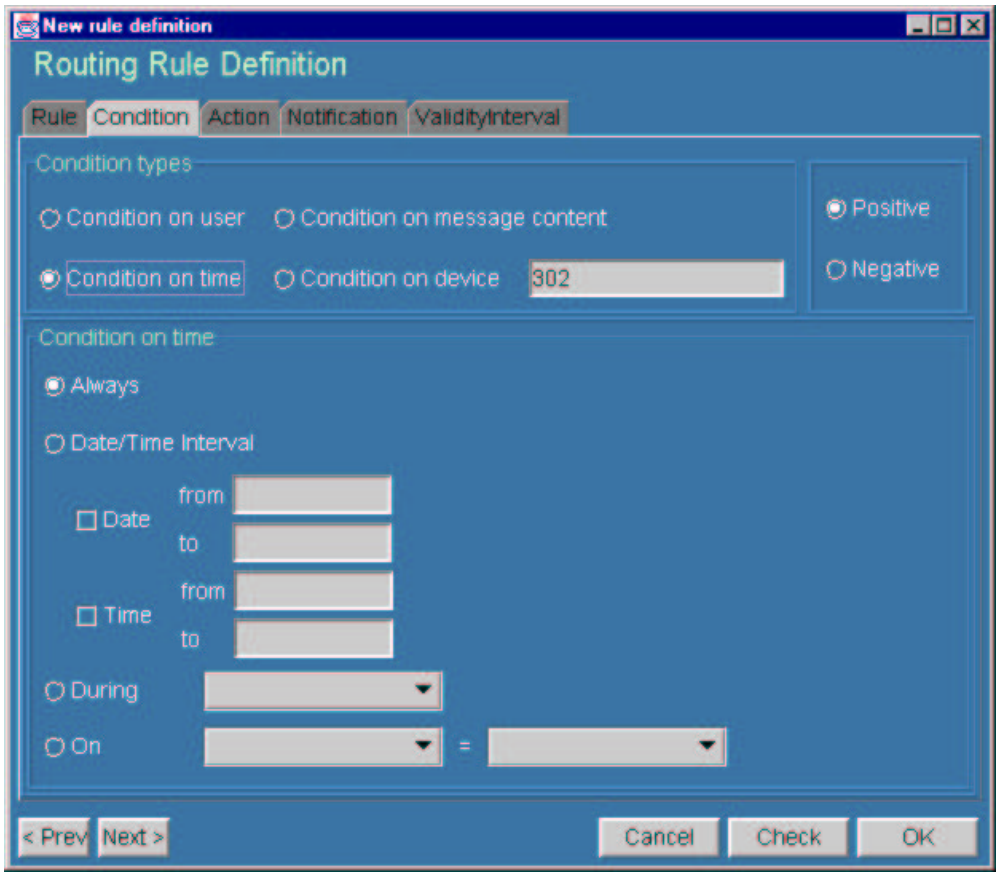


Figura 4.12: Interfaccia di definizione delle regole

di "servizio", ossia utili per controllare la base di dati e per verificare la correttezza della ricezione dei messaggi inviati ad un determinato UCI .

Il menù *Options* permette di ridefinire variabili di sistema o di decidere la politica di selezione delle regole da eseguire al momento della gestione di un messaggio in arrivo ad un certo UCI (Figura 4.13).

Quando l'instradamento di un messaggio richiede la valutazione delle re-

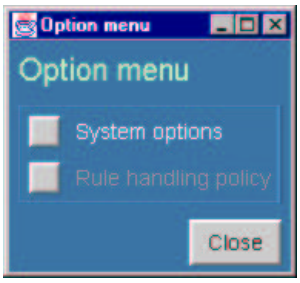


Figura 4.13: Opzioni di sistema

gole, il motore valuta tutte le regole dell'utente in ordine di definizione, fino a quando ne trova una valida e la esegue. Un'altra possibilità sarebbe quella di permettere all'utente di definire una priorità per ogni regola e, in tal caso, il sistema comincerebbe a valutare le regole con priorità maggiore. Essendo stata implementata solo la prima politica di selezione delle regole, la funzione che ne permette una scelta non è stata attivata.

La ridefinizione dei parametri di sistema è sempre legata alla gestione del messaggio, poiché le modifiche apportate influiscono sull'efficienza del sistema. In questa sezione viene data una breve descrizione di questi parametri, il cui funzionamento è stato analizzato nel Capitolo 3, Sezione 3. . I valori dei parametri di sistema possono essere impostati tramite l'interfaccia in Figura 4.14, i cui campi indicano rispettivamente:

- Il tempo che un dato messaggio rimane memorizzato nel MainBuffer[1] prima di essere gestito.
- Il tempo che trascorre tra due tentativi di consegna del messaggio, ossia l'intervallo di tempo che passa prima che un messaggio venga memorizzato nel MainBuffer per essere gestito nuovamente.
- Il tempo di vita di un messaggio, ossia il numero di volte che il sistema gestisce lo stesso messaggio prima di memorizzarlo nella base di dati e spedire una notifica all'utente per avvertirlo dell'errore verificatosi.
- Il numero di volte che il sistema valuta le stesse regole, prima di memorizzare il messaggio nella base di dati e spedire una notifica all'utente per avvisarlo della presenza di un ciclo, verificatosi a causa delle regole l'utente stesso ha definito.

I valori di questi parametri devono essere impostati tenendo conto del numero di utenti appartenenti al sistema e del numero di messaggi che il sistema riceve.

Attraverso il menù *Services* (Figura 4.15) è possibile creare, cancellare o inizializzare tutte le tabelle che costituiscono la base di dati utilizzata dal sistema. Con l'inizializzazione vengono inseriti un certo numero di tipi e di modelli di dispositivi, alcuni formati e metodi di traduzione, e qualche utente che permette di controllare come avviene la consegna di un messaggio.

Per simulare la spedizione di un messaggio, è stata creata una maschera

[1] Molte delle informazioni in questo paragrafo si riferiscono all'architettura del motore descritta nel Capitolo 3.

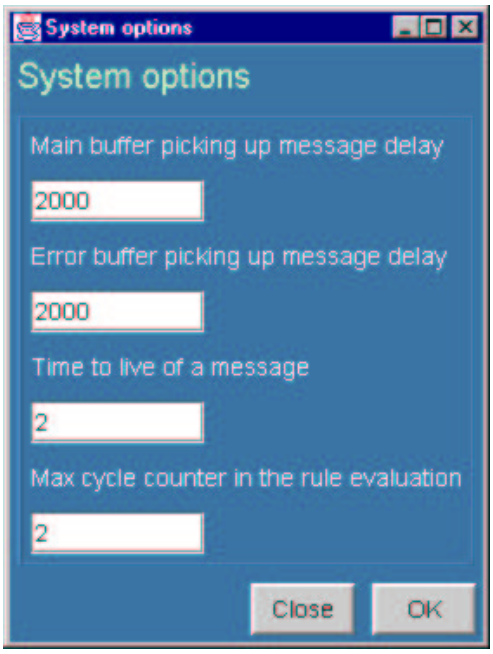


Figura 4.14: Variabili di sistema

nella quale è possibile scegliere mittente e destinatario del messaggio, tra gli utenti appartenti al sistema, il formato del messaggio da inviare, eventualmente il suo contenuto in formato testo e la sensitività dell'informazione trasportata. È possibile, inoltre, definire la firma digitale, lo stato di compressione o di *encryption*, e una *UserCard* tra quelle definite dal mittente. Queste ultime informazioni non sono necessarie, in quanto il mittente potrebbe desiderare di non dare alcuna informazione di sé oltre l'UCI, mentre le funzioni per comprimere o crittare il messaggio e la firma digitale sono state previste, ma la loro implementazione potrebbe riguardare una futura estensione al progetto.

Una volta spedito il messaggio, è possibile controllare, tramite un'interfaccia, su quali dispositivi dell'utente è stato ricevuto se la consegna è avvenuta correttamente, o la situazione di errore verificatasi. Se sono state inviate

notifiche, viene dato un resoconto anche sulla loro gestione. Un'altra maschera si apre per verificare il tipo di messaggio che è stato spedito.

Lo *Start runtime environment* è necessario per attivare il motore per la fase di *testing* della gestione dei messaggi, fatta attraverso la maschera per simulare la spedizione di un messaggio. Durante il normale funzionamento del sistema, il motore è sempre attivo e questa funzione è inutilizzata.

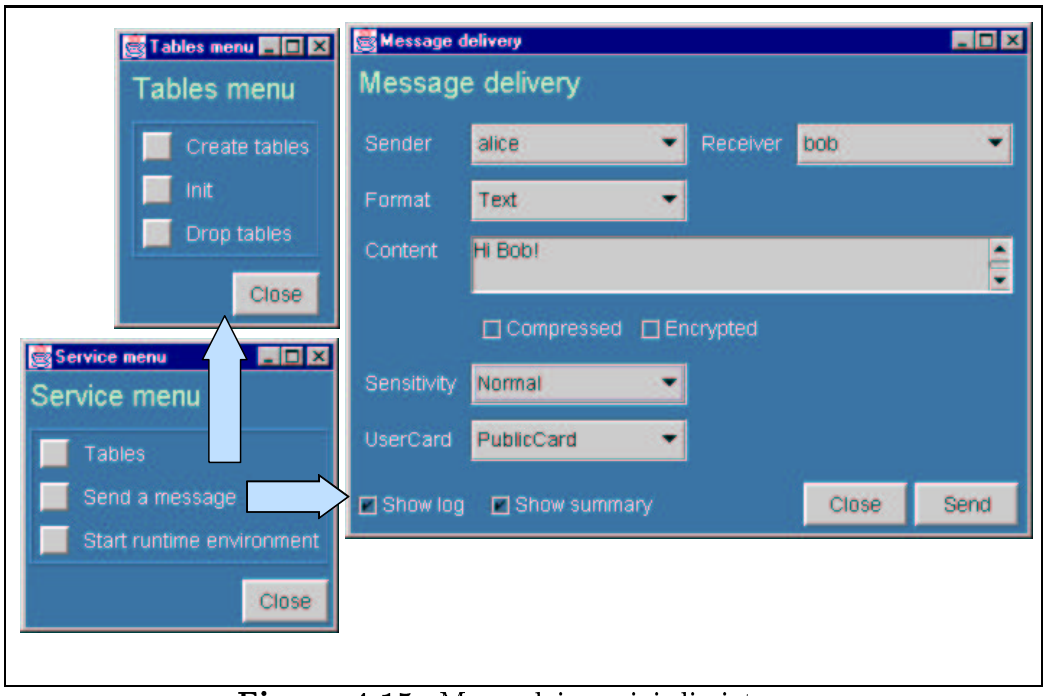


Figura 4.15: Menu dei servizi di sistema

4.5 Struttura dell'applicativo

La struttura data al progetto dipende sia da come il sistema interagisce con l'ambiente esterno sia dalla sua architettura[2]. Gli utenti e l'amminisrtatore interagiscono con il sistema attraverso delle interfacce, utilizzate per inse-

[2]Per maggiori dettagli leggere il Capitolo 3.

rire le informazioni da memorizzare nella base di dati. In questo modo, al momento di gestire un messaggio, il sistema preleva dalla base di dati il profilo dell'utente e, in base alle regole e ai dispositivi definiti dell'utente stesso, consegna il messsaggio. Questa breve descrizione del funzionamento del sistema è stata data per spiegare quali sono state considerate le componenti pricipali del progetto, per le quali sono stati creati dei *package*. I *package* che sono stati implementati all'interno del progetto UCI sono:

- *Db*, che contiene tutti i metodi interessati ad interagire con la base di dati.
- *Device*, che contiene le classi per l'istanza dell'oggetto dispositivo.
- *Manager*, che contiene tutti le classi interessate alla gestione del messaggio e che rappresentano le componenti fondamentali dell'architettura del motore.
- *Message*, che contiene le classi per l'istanza dei messaggi e delle notifiche.
- *Profile*, che contiene le classi per istanziare il profilo dell'utente e quello dell'amministratore.
- *RoutingRule*, che contiene le classi rappresentative di tutti i tipi di condizioni e di regole definibili dall'utente.
- *UI*, che contiene tutte le interfacce implementate.
- *Util*, che contiene i metodi ritenuti utili in fase di implementazione, ma non direttamente legati al contesto del progetto.

Ciascuna di queste parti è stata organizzata a seconda della realtà di applicazione delle componenti interne.

Il *package UCI.Db* è stato strutturato in modo da dividere i metodi a seconda delle entità, presenti nella base di dati, alle quali accedono. I *package* definiti all'interno di *UCI.Db* sono:

- *UCI.Db.Credential.* In questo *package* è stata implementata la classe contenente i metodi per la creazione, la modifica e la gestione delle credenziali e delle *usercard.*

- *UCI.Db.Device.* In questo *package* sono presenti le classi utilizzate per l'esecuzione delle varie operazioni permesse agli utenti sui loro dispositivi, e agli amministratori sui tipi e modelli dei dispositivi che il sistema vuole gestire.

- *UCI.Db.Message.* In questo *package* è contenuta la classe che permette la memorizzazione, nella base di dati, di un messaggio non consegnato dal sistema. Per consegnare i messaggi occorre sapere se un dispositivo è compatibile con essi oppure, eventualmente, quali metodi di traduzione supporta. Queste informazioni sono note poiché, ogni volta che l'amministratore introduce un nuovo modello di dispositivo, deve anche definire i formati che questo è in grado di ricevere direttamente. É stato, quindi, necessario creare una classe, i cui metodi si occupano solo della gestione dei formati dei messaggi e dei metodi di traduzione, e dell'esecuzione di tutte le operazioni su questi attributi del messaggio, richieste nella valutazione delle regole e nela gestione del messaggio stesso.

- *UCI.Db.Tables* In questo *package* sono presenti le classi che i cui metodi si occupano della cancellazione o eliminazione di tutte le tabelle presenti nella base di dati. Per poter fare dei test sul funzionamento della gestione dei messaggi, è stata creata una classe i cui metodi permettono di inizializzare la base di dati con un certo numero di utenti.

- *UCI.Db.Users* In questo *package* sono state implementate tutte le classi interessate alla gestione degli utenti. Nella classe *Users* sono presenti i metodi per la cancellazione e inserimento degli utenti, per la verifica delle loro parole chiave e per l'inserimento dell'amministratore. Nella classe *Group* sono presenti i metodi che si occupano della definizione e della cancellazione dei gruppi e della gestione degli utenti che vi appartenengono.

Al momento dell'attivazione del sistema, viene creata la connessione alla base di dati attraverso l'istanziazione della classe *UCIDB* definita nel *package* *UCI.Db* (Figura 4.16).

All'interno del *package* *UCI.Device* è contenuta la classe *SimpleDevice* che, quando viene istanziata, crea l'oggetto caratterizzato solo dall'idenificatore del dispositivo. La classe *Device* eredita dalla *SimpleDevice* l'identificatore del dispositivo e contiene, inoltre, il metodo di traduzione associato al dispositivo stesso. Queste due classi sono state create per la necessità di avere un oggetto che identificasse il dispositivo indipendentemente dal tipo del suo identificatore. Infatti una modifica ti tale tipo viene eseguita solo all'interno della classe *SimpleDevice* e non ogni volta che è stata utilizza-

```
public class UCIDB() {
    /* Costruttore della classe */
    public WNDB() {ConnectionToDB();}
    /* Registra il driver per accedere alla base di dati */
    private void registerDriver() {...}
    /* Crea la connessione alla base di dati */
    private Connection createConnection() {...}
    /* Richiama i due metodi precedenti per stabilire la
       connessione alla base di dati */
    private void ConnectionToDB() {...}
    /* Restituisce la connessione */
    public Connection getConnection() {...}
    /* Chiude la connessione */
    public void closeConnection() {...}
}
```

Figura 4.16: Struttura della classe **UCIDB**

ta la sua istanza. La classe *DeviceList* rappresenta una lista di oggetti di tipo *SimpleDevice*. Queste classi sono state implementate per rendere più efficiente la gestione dei dispositivi.

Il *package UCI.Manager* contiene tutte le classi per la gestione dei messaggi. *MessageDaemon*, *DelayDaemon*, *UCIRouter* e *ErrorRouter* rappresentano le componenti pricipali della gestione del messaggio all'interno del motore[3]. Gli algoritmi implementati in queste classi sono stati descritti in [2].

Il *package UCI.Message* contiene al suo interno le classi per la creazione di oggetti Java che rappresentano i messaggi e le notifiche. Essendo queste ultime un tipo particolare di messaggio, è stata implemnetata la classe *SimpleMessage* dalla quale le classi *Message* e *NotifyMessage* ereditano gli

[3]La descrizione del funzionamento di queste classi è contenuta nel Capitolo 3.

attributi comuni, come ad esempio l'identificatore.

Il *package UCI.Profile* contiene i profili dell'amministartore e dell'utente ciascuno con le informazioni di cui necessita. Poiché entrambi sono caratterizzati dall'UCI del proprietario, è stato creato un profilo generale da cui ereditare le caratteristiche comuni. Il profilo è molto importante durante il funzionamento del sistema perché contiene tutte le informazioni pricipali utilizzate per la gestione dei messaggi. Ad esempio, il profilo dell'utente contiene la lista degli identificatori dei dispositivi utilizzati dall'utente, la lista degli identificatori dei dispositivi indicati dall'utente come riceventi delle notifiche e le regole definite dall'utente. Il profilo dell'utente viene caricato in memoria come istanza della classe *UCI.UserProfile* ad ogni gestione di un messaggio ricevuto e ogni volta che l'utente accede al sistema tramite interfacce. Il profilo dell'amministratore viene caricato in memoria come istanza della classe *UCI.AdminProfile* ogni volta che l'amministratore accede al sistema tramite l'interfaccia di *Login*.

IL *package UCI.RoutingRule* contiene tutte le classi necessarie ad implementare i diversi tipi di condizioni presenti all'interno delle regole e le regole stesse. Una descrizione dettagliata delle regole e delle condizioni che le compongono è presente in [2].

Il *package UCI.UI* contiene tutte le interfacce di cui l'amministratore e gli utenti si servono. Sono divise in ulteriori pacchetti a seconda dell'argomento:

- *UCI.UI.Credential* contiene tutte le interfacce definite per la creazione, la modifica e la vista delle credenzilai e delle *usercard*.

- *UCI.UI.Device* contiene tutte le interfacce utilizzate dall'utente, per cancellare, inserire, prestare i dispositivi o per modificarne le caratteristiche, e dall'amministratore per inserire o cancellare tipi e modelli di dispositivi e modificarne i formati o gli stati supportati.

- *UCI.UI.Menu* contiene tutte le interfacce che contengono la lista di operazioni, organizzate a seconda del contesto, che utenti e amministratori possono eseguire.

- *UCI.UI.Message* le interfacce contenute in questo *package* sono state implementate per testare il funzionamento del sistema, attravesro la spedizione di un messaggio tra due utenti caratterizzati da un certo profilo.

- *UCI.UI.Otions* contiene l' interfaccia per la definizione dei parametri di sistema.

- *UCI.UI* contiene le interfacce per la definizione, la cancellazione o la modifica delle regole dell'utente.

- *UCI.UI.User* contiene le interfacce per eseguire le operazioni implementate per la gestione degli utenti e dei gruppi da loro definiti.

Il *package UCI.Util* contiene una classe in cui sono stati implementati i vari metodi utilizzati per l'implementazione, ma usati più volte senza essere legati a nessun contesto in particolare. Ad esempio, sono contenuti i metodi per convertire una stringa in un array di byte e viceversa; i metodi per covertire una stringa in una data; un metodo per la gestione delle eccezioni

```
public class Util() {
    /* Coverte una stringa in una data */
    public static java.sql.Date strToSqlDate(String dateExpr) {...}
    /* Converte un array di byte in una stringa */
    public static char[] byteArrayToCharArray(byte[] byteArray) {...}
    /* Converte una stringa in un array di byte */
    public static byte[] charArrayToByteArray(char[] charArray) {...}
    /* Gestisce le eccezioni che si possono verificare durante
       l'esecuzione di interrogazioni alla base di dati */
    public static void warnings(SQLWarning warning) {...}
}
```

Figura 4.17: Struttura della classe **Util**

che si possono verificare durante l'esecuzione di interrogazioni alla base di dati (Figura 4.17).

L'applicativo è stato sviluppato in Java e interagisce con una base di dati Oracle su cui vengono memorizzati i dati degli utenti, dei dispositivi e dei messaggi. Il progetto è stato realizzato in Microsoft Windows NT 4.0 utilizzando Oracle 8i Personal Edition e Borland Jbuilder 4.

Capitolo 5

Future estensioni

Le possibili estensioni che sono state presentate in questo capitolo vengono proposte con lo scopo di migliorare la qualità e l'efficienza del sistema implementato. Rendere il motore indipendente dalle interfacce e dalla base di dati permetterebbe di separare la gestione dei messaggi, l'interazione con gli utenti e il recupero delle informazioni. Mentre poter separare la memorizzazione delle informazioni in basi di dati defferenti, porterebbe diversi enti ad essere responsabili per i dati che controllano. Altre estensioni che sono state proposte, riguardano la gestione di dispositivi e messaggi. È stata presentata la possibilità di avere condivisione di dispositivi e di definire dispositivi di sistema per la memorizzazione dei messaggi non consegnati. Ciò comporta anche lo sviluppo di un metodo per dare all'utente la possibilità di recuperare questi messaggi.

Il capitolo è organizzato come segue. Nella Sezione 5.1 viene presentata la possibilità di rendere le interfacce indipendenti dal motore definendole su WEB. Nella Sezione 5.2 viene presentata la possibilità di rendere la base di

dati indipendente dal motore centralizzando gli accessi. Nella Sezione 5.3 viene presentata la possibilità di suddividere la memorizzazione delle informazioni in diverse basi di dati gestite da enti differenti. Nella Sezione 5.4 viene presentata la possibilità di memorizzare le politiche di instradamento in maniera differente da quella utilizzata, spiegando come questi metodi influiscono sulla loro gestione. Nella Sezione 5.5 viene presentata la possibilità di condividere l'utilizzo dei dispositivi e di permettere più livelli di prestito. Nella Sezione 5.6 viene presentata la possibilità di definire gruppi dinamici per la definizione di regole che permettono il filtraggio dei messaggio spediti da utenti appartenenti a tali gruppi. Nella Sezione 5.7 viene presentata la possibilità di definire dispositivi di sistema per la memorizzazione dei messaggio non consegnati e viene presentao il problema del recupero di tali messaggi.

5.1 Indipendenza del motore dalla interfacce

Dato il grande sviluppo che i sistemi informatici e le reti legate a Internet hanno avuto negli ultimi anni, una possibile estensione futura potrebbe essere quella di sviluppare le interfacce come pagine *WEB* per essere accettate da qualsiasi *browser*. Infatti, essendo Internet oramai accessibile a tutti e ovunque, sarebbe utile dare la possibilità agli utenti di poter modificare il proprio profilo in ogni momento. Sempre per il medesimo scopo, potrebbe risultare comodo per gli utenti avere la possibilità di impostare le proprie preferenze tramite telefono. Questo risultato può essere raggiunto se si pensa che questo grande sviluppppo ha colpito anche la telefonia mo-

bile, spingendo i telefoni portatili a diventare dei mini-computer, sempre più simili ai palmari, in grado di gestire messaggi multimediali, ossia telefonate, email... La possibilità di sviluppare le interfacce su *WEB*, permette di renderle completamente indipendenti dal motore, inoltre le informazioni scambiate tra le interfacce WEB e il motore stesso potrebbero essere definite in formato XML.

5.2 Indipendeza del motore dalla base di dati

Affiché il motore risulti indipendente anche dalla base di dati si potrebbe centralizzare l'accesso ai dati, migliorando l'efficienza del sistema. Infatti, poiché ogni gestione di un messaggio comporta l'accesso alla base di dati per il caricamento in memoria del profilo del ricevente, il gestore della base di dati potrebbe trovarsi a gestire più connessioni di quante ne supporta effettivamente. Una soluzione ad accesso centralizzato alla base di dati, permetterebbe di rendere indipendente il numero dei messaggi gestiti in un certo intervallo di tempo dal numero di connessioni alla base di dati, evitando, così, che il gestore della base di dati si trovi in una situazione di errore a causa delle troppe connessioni.

5.3 Ripartizione dei dati memorizzati

Tutte le informazioni relative agli utente, ai dispositivi, alle regole e ai messaggi sono state memorizzate all'interno della stessa base di dati. Tuttavia,

queste informazioni potrebbero essere amministrate da enti differenti. Per esempio le credenziali potrebbero essere gestite dalla Credential Authority e, quindi, essere memorizzate indipendentemente da tutte le altre informazioni. In questo modo, la veridicità delle informazioni contenute nelle credenziali, che gli utenti spediscono insieme ai messaggi per comunicare ai destinatari le proprie caratteristiche, sarebbe assicurata dall'ente che se ne occupa direttamente.
Il motore richiederebbe le informazioni di cui necessita ad un gestore dei dati posseduti dai diversi enti.

5.4 Memorizzazione delle politiche di instradamento

Le politiche di instradamento definite dagli utenti sono state memorizzate nella base di dati come stringa di byte. In questo modo, al momento della gestione di un messaggio, tutte le regole definite dall'utente vengono caricate in memoria attraverso una sola interrogazione alla base di dati. Tuttavia, se l'amministratore desiderasse sapere, ad esempio, quante regole sono valide in un dato momento, oppure il numero di regole che riguardano una determinata tipologia di dispositivo, il sistema si troverebbe a dover caricare in memoria tutte le regole degli utenti per valutarle. Per rendere più efficienti le interrogazioni di tipo statistico interessate alle regole, si potrebbero memorizzare separatamente le condizioni e le componenti di una regola. In questo modo, la lista di tutte le regole valide in un dato mo-

mento comporterebbe la selezione solamente delle regole caratterizzate da un intervallo di validità conforme alla richiesta definita nell'interrogazione. Questa soluzione rallenta, però, la gestione dei messaggi. Si potrebbe, allora, memorizzare nella base di dati le regole sia come stringa di byte che come parti separate. Le modifiche apportate alle regole potrebbero essere immediate per le regole memorizzate come stringa di byte, e avvenire subito dopo per quelle memorizzate in parti separate.

5.5 Prestito e condivisione

Le informazioni memorizzate all'interno della base di dati sui dispositivi, permettono di riconoscere i dispositivi utilizzati effettivamente dall'utente, vale a dire quelli di cui è proprietario e che non ha prestato ad altri utenti, e quelli che ha ricevuto in prestito. Infatti, il sistema implementato prevede un unico livello di prestito senza condivisione. Al fine di poter avere più livelli di prestito, che consentono ad un utente di prestare ad un altro utente un dispositivo ricevuto in prestito, si dovrebbe mantenere la lista dei prestiti avvenuti per poter risalire in ogni momento al proprietario e all'utilizzatore effettivi del dispositivo.

Per quanto riguarda la gestione dei dispositivi condivisi, permettere l'utilizzo di un dispositivo da parte di più utenti contemporaneamente porta a dover risolvere problemi legati alla sicurezza. Infatti, si dovrebbe controllare chi utilizza il dispositivo condiviso, e che il dispositivo sia utilizzato nei limiti delle capacità concesse dal proprietario del dispositivo.

5.6 Gruppi dinamici

Gli utenti hanno la possibilità di definire dei gruppi per filtrare i messaggi in arrivo, stabilendo condizioni sui gruppi stessi. L'utente potrebbe avere la possibilità di stabilire dei gruppi dinamici per definire regole come: "Rifiuta il messaggio se l'utente lavora per una data società". In questo modo, chi definisce il gruppo non conosce esattamente quali utenti lo compongono e, inoltre, questi possono cambiare. La definizione di gruppi dinamici permette, quindi, all'utente di filtrare i messaggi spediti da utenti accomunati da una certa caratteristica, senza sapere chi e quanti sono.

5.7 Memorizzazione e recupero dei messaggi non consegnati

Se il sistema non riesce a consegnare un messaggio, questo viene memorizzato nella base di dati. Potrebbero essere definiti dei dispositivi di sistema nei quali memorizzare i messaggi non consegnati, a seconda dei loro formati e della loro dimensione. In entrambi i casi dovrebbe essere implementato un metodo per dare la possibilità all'utente di recuperare i messaggi che non ha potuto ricevere e un'interfaccia per permettere al destinatario di poter controllare questi messaggi. Se la consegna del messaggio non è avvenuta in quanto l'utente non possiede dispositivi compatibili, il sistema potrebbe permettere il prestito dei suoi dispositivi affinché l'utente, se vuole, sia comunque in grado di ricevere il messaggio.

Capitolo 6

Conclusioni

Nella presente tesi è stato spiegato il progetto sviluppato per la gestione di identificatori universali per la comunicazione. Tale progetto è nato dalla neccesità degli utenti di poter comunicare tra loro indipendentemente dai dispositivi utilizzati. Infatti, il grande sviluppo tecnologico avvenuto negli ultimi tempi ha spinto le persone a possedere più dispositivi con caratteristiche differenti e in grado di gestire messaggi di diversi formati. Dover ricordare gli identificatori dei dispositivi posseduti dagli utenti con cui si vuole comunicare e i formati che questi supportano, ha portato alla definizione di identificatori universali per la comunicazione (UCI). Utilizzando i propri UCI, i mittenti si possono disinteressare completamente dei dispositivi a cui un messaggio verrà consegnato. Grazie alla possibilità di definire politiche di instradamento, i destinatari affidano al sistema la scelta dei dispositivi adatti alla ricezione del messaggio stesso tra quelli che verificano le regole stabilite.

La prima parte della tesi spiega i motivi che hanno portato allo studio

di un sistema di comunicazione indipendente dai dispositivi utilizzati. Sono state presentate inoltre le caratteristiche principali del progetto affrontate in questa tesi.

La seconda parte della tesi ha riguardato la progettazione del modello dei dati del motore a regole, analizzando separatamente tre parti: lo schema dei dispositivi, lo schema degli utenti, e lo schema dei messaggi. In questo modo è stato possibile creare una struttura in grado di acquisire, elaborare, trasmettere ed archiviare tutti i dati necessari alla gestione dei messaggi in arrivo agli utenti del sistema.

Nella terza parte della tesi è stata presentata l'architettura del sistema sviluppato, spiegando come questo interagisce con gli utenti, la base di dati e i servizi esterni forniti dal *provider*. Dopo aver mostrato il comportamento del sistema con l'ambiente esterno, è stato analizzato il comportamento interno del motore, basato su regole attive, sviluppato per instradare un messaggio, in arrivo ad un determinato UCI, sui dispositivi adatti a riceverlo. Utilizzando degli esempi sono state definite alcune regole, spigando come avviene la gestione effettiva dei messaggi e come il motore controlla eventuali errori che si possono verificare.

La quarta parte della tesi descrive la struttura delle interfacce implementate per l'amministratore e gli utenti e il funzionamento di alcune maschere create per testare la consegna dei messaggi. Inoltre, è stata presentata la struttura dell'applicativo, ossia i *package* la cui definizione è avvenuta in base alle le componenti pricipali del progetto.

Nella quinta parte della tesi sono state presentate alcune fra le possibili estensioni future, come ad esempio: l'idea di rendere il motore indipendente

dalle interfacce e dalla base di dati, la possibilità di avere dispositivi di sistema per memorizzare i messaggi non consegnati; permettere all'utente di definire gruppi dinamici utilizzati nella definizione delle regole; memorizzare le regole spezzandole nelle varie componenti piuttosto che come stringhe di byte. Altre estensioni possono essere trovate in [2].

I contributi apportati da questa tesi possono essere così schematizzati:

- Analisi dei problemi di instradamento dei messaggi;
- Realizzazione del modello dei dati del motore basato su regole;
- Definizione dell'architettura del motre basato sulle politiche di instradamento;
- Implementazione del motore basato su regole per la gestione di messaggi;
- Implementazione di un'interfaccia che consente di verificare il corretto funzionamento della gestione dei messaggi e delle maschere che permetteno agli utenti e all'amministratore di interagire con il sistema;

La tesi presentata può essere utilizzata come spunto per la realizzazione effettiva di un servizio basato su politiche di instradamento per permettere la comunicazione tra utenti attraverso oppurtuni identificatori personali. Utilizzando tale servizio la comunicazione tra gli utenti risulterebbe facilitata e la distinzione tra telefonia e Internet potrebbe assottigliarsi ulteriormnete.

Bibliografia

[1] Elena Ferrari Giovanna Guerrini Elisa Bertino, Barbara Catania. *Sistemi di basi di dati. Concetti e architetture.* CittàStudiEdizioni, prima edizione edition, 1997.

[2] Giuseppe Garau. Sviluppo... Master's thesis, Univetsitá degli Studi di Milano, July 2001.