UNIVERSITÀ DEGLI STUDI DI BOLOGNA

FACOLTÀ DI INGEGNERIA
Corso di Laurea in Ingegneria Informatica
Reti di Calcolatori L-A

SUPPORTO APPLICAZIONI GROUP AWARE IN SCENARI MOBILE AD-HOC NETWORK

Tesi di Laurea di:
Michele Baldassarro

Relatore:
Chiar.mo Prof. Ing. Antonio Corradi

Correlatore:
Ing. Dario Bottazzi

Anno Accademico 2003/2004 – III Sessione

INDICE

INTRODUZIONE

La crescente diffusione di dispositivi portatili e i recenti sviluppi delle tecnologie wireless aprono un nuovo scenario in cui gli utenti richiedono la possibilità di usufruire dei servizi rete in ogni momento ed ovunque essi si trovino. Nel nuovo scenario gli utenti non richiedono solo la possibilità di fruire dei tradizionali servizi Internet quali il WEB e l'e-mail, ma richiedono anche di poter collaborare con i vicini per raggiungere comuni obiettivi o per soddisfare necessità immediate. Esempi di servizi collaborativi avanzati che emergono nel nuovo scenario sono il file sharing, la gestione del traffico automobilistico o il supporto alla cooperazione fra operatori in scenari di protezione civile.
Le tecnologie wireless, ed in particolare le Mobile Ad-hoc NETwork (MANET), costituiscono un supporto adatto alla fornitura di servizi collaborativi nel nuovo scenario e consentono interazioni ubique fra gli utenti anche in assenza di infrastrutture di comunicazione pianificate. Le caratteristiche uniche delle MANET, ed in particolare la mobilità dei terminali utente, causano frequenti ed impredicibili variazioni nella topologia della rete. In particolare, connessioni, disconnessioni, partizioni e merge di rete sono eventi frequenti che causano transitori nella collaborazione fra utenti nuovi e precedentemente sconosciuti.
L'alto livello di dinamicità del nuovo scenario mina alla base le assunzioni di progetto dei sistemi collaborativi tradizionali. Il supporto di servizi collaborativi avanzati in scenjari MANET richiede perciò di investigare soluzioni innovative che seguano linee guida differenti. In questa tesi presentiamo AGAPE, un sistema di gestione dei gruppi context-aware, che supporta e promuove lo sviluppo di servizi collaborativi in scenari MANET. In particolare, AGAPE consente la creazione dei gruppi on demand, il discovery di gruppi e di possibili partner per la collaborazione e consente il monitoraggio della loro disponibilità on-line.
Nel capitolo 1 caratterizzeremo le Mobile Ad-hoc Networks, mentre il nel capitolo 2 presenteremo framework AGAPE. Nel capitolo 3 dettaglieremo l'analisi del Join/Leave Manager Service (J/LMS), mentre nel capitolo 4 descriveremo il View Manager Service (VMS). AGAPE verrà poi mostrato al lavoro nel capitolo 5 tramite un caso di studio nel campo della protezione civile.

CAPITOLO 1
RETI WIRELESS AD-HOC

L’ampia diffusione dei dispositivi portatili che possono beneficare di connettività wireless, l’aumento di possibilità di connessione e la nascita delle MANET aprono un nuovo scenario in cui gli utenti non richiedono più di beneficiare solo dei tradizionali servizi internet quali il web e la mail. In particolare, nel nuovo scenario, gli utenti richiedono servizi collaborativi che consentano loro di collaborare.
Le MANET (mobile ad-hoc network) rappresentano un sistema di nodi mobili capaci di organizzarsi liberamente e dinamicamente in topologie di rete arbitrarie e tipicamente temporanee, permettendo la comunicazione tra dispositivi senza un’infrastruttura preesistente.
Nel nuovo scenario MANET gli utenti che condividono interessi e obiettivi comuni richiedono di potere creare e dissolvere gruppi e unirsi o lasciare gruppi esistenti. Inoltre la mobilità di tali utenti fa sì che questi appaiano e scompaiano in modo non predicibile pur dovendo mantenere l’appartenenza ai gruppi.
Oltre questo problema c’è anche il fatto che gli utenti usano dispositivi eterogenei che hanno, quindi, diversa capacità di calcolo, diversa quantità di memoria, diversa ampiezza di banda.
Una classificazione largamente accettata per le reti wireless è in base all’area di copertura della rete.
Secondo questo criterio possiamo individuare le BAN, le PAN, le WLAN e le WWAN.
I dispositivi che compongono le reti wireless sono spesso dispositivi mobili alimentati a batteria che devono, quindi cercare di ridurre i consumi di energia.
Gli standard più diffusi per le reti wireless sono lo IEEE 802.11 e il bluetooth. Il primo è maggiormente usato nelle WLAN il secondo nelle BAN.

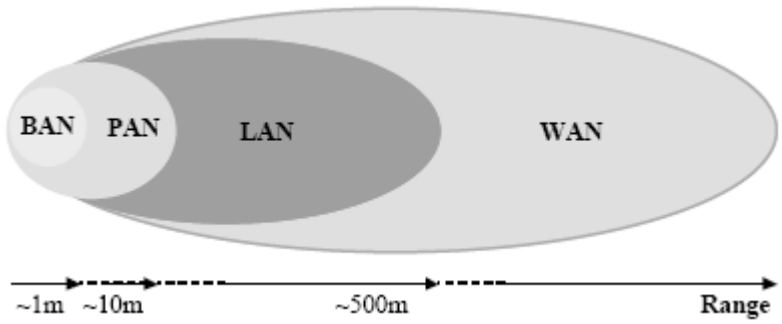

fig. 1.1 – Classificazione delle reti

1.1 BAN: Body Area Network

Le BAN rispondono alla necessità di collegare una moltitudine di dispositivi che possono essere "indossati" (microfoni, auricolari, palmari, display) detti wereable computer.
Le BAN consentono, quindi, una copertura dell'ordine di uno o due metri.
Uno dei primi esempi di BAN è il prototipo sviluppato da T.G. Zimmermann, che permette lo scambio dei dati utilizzando il corpo umano come canale. Zimmermann mostrò che potevano essere usate piccolissime correnti, dell'ordine di un milionesimo d'ampere, per trasferire dati attraverso la pelle. Quindi lo scambio di informazioni tra due persone potrebbe avvenire attraverso una semplice stretta di mano.

1.2 PAN: Personal Area Network

Le PAN vengono usate per connettere tra loro dispositivi mobili e dispositivi situati nell'ambiente intorno alla persona, come ad esempio, stampanti o punti di accesso a internet.
L'area di copertura delle PAN intorno ai dieci metri.
Una delle tecnologie usate per la creazione delle PAN è quella che sfrutta la banda 2.4 GHz ISM. Questa tecnologia usa lo spread spectrum per ridurre le interferenze.

1.3 WLAN: Wireless Local Area Network

Negli ultimi anni l'uso delle tecnologie wireless in ambiente LAN è diventato molto diffuso, in quanto le wireless LAN (WLAN) offrono una maggiore flessibilità rispetto alle LAN cablate.
Una WLAN deve soddisfare tutte le caratteristiche di una LAN cablata, incluse alta capacità, piena connettività tra le stazioni che compongono la rete e deve supportare il broadcast.
Le WLAN hanno un'area di copertura compresa dell'ordine delle centinaia di metri.
Oltre a rispettare le caratteristiche delle LAN, le WLAN hanno anche caratteristiche tipiche dell'ambiente wireless: devono supportare la mobilità dei nodi della rete. Inoltre hanno banda limitata a causa della trasmissione aerea.

1.4 WWAN: Wide Wireless Area Network

Le WWAN sono le reti con copertura maggiore. Tali reti sfruttano la copertura della rete GSM usata per i telefoni cellulari. Per questo motivo gli utenti possono usufruire dei servizi offerti dalla rete ovunque c'è copertura GSM. I servizi offerti da tali reti sono tipicamente l'uso della posta elettronica e la navigazione in internet. Gli standard più diffusi sono il GPRS e l'UMTS.
Tali standard sono stati progettati per funzionare, quindi, su telefoni cellulari, per mezzo di SIM card che li supportano.
L'Europa ed il Giappone hanno deciso di implementare la parte terrestre dell'UMTS (l'UTRA air interface - interfaccia aerea UTRA) nelle bande accoppiate 1920-1980 MHz e 2110-2170 MHz. L'Europa inoltre ha deciso di implementare l'UTRA nelle bande non accoppiate 1900-1920 MHz e 2010-2025 MHz.
Per quanto riguarda il GPRS, esso può usare alcune risorse del GSM con un bit rate massimo 171,2 Kbps.
Il GPRS consentendo alle informazioni di essere trasmesse più velocemente tramite le reti mobili risulta essere un efficiente ed economico mezzo di trasmissioni dati mobile, riuscendo a sfruttare i protocolli TCP/IP e X25.

1.5 IEEE 802.11

Lo sviluppo della tecnologia wireless ha portato allo sviluppo di standard.
Lo standard principale è lo IEEE 802.11b che lavora nella banda di 2.4 GHz con data rate massimo di 11 Mb/sec. Tale standard viene usato per costruire delle BAN e PAN.
Un altro gruppo ha sviluppato lo standard IEEE 802.11a il quale lavora nella banda di 5 GHz con data rate 54 Mb/sec, che maggiormente usato nelle WLAN.

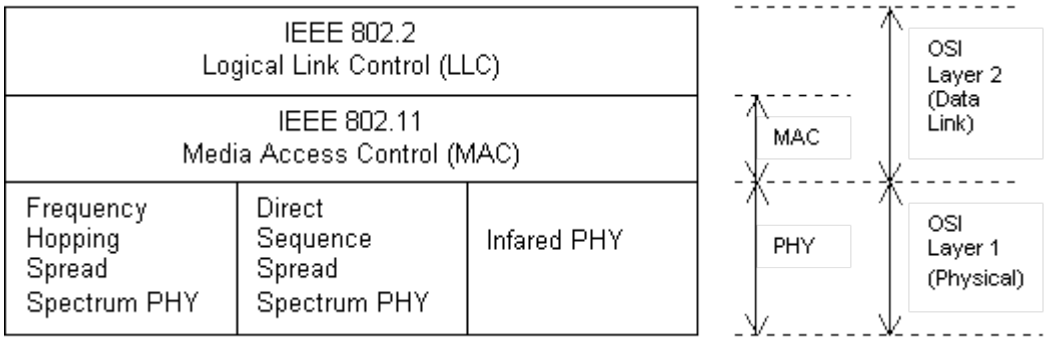

fig. 1.2 – 802.11: architettura

Lo IEEE 802.11 specifica lo strato fisico e lo strato MAC per le WLAN.
Per il livello fisico lo standard prevede o la comunicazione via infrarossi o via radio.
Per quanto riguarda il livello MAC, lo IEEE 802.11, prevede sia l' accesso al mezzo tramite un meccanismo contention-based, sia tramite un meccanismo cotention-free.
Il metodo d'accesso basilare è il *Distribuited Coordination Function* (DCF), che è un *Carrier Sensing Multiple Access with Collision Avoidance* (CSMA/CA). Il DCF è un sistema d'accesso a contesa.
Il metodo d'accesso al mezzo contention-free è il *Point Coordination Function* (PCF). Questo metodo d'accesso opera in modo simile al polling: c' è un coordinatore che fornisce, attraverso il polling, i diritti di trasmissione ad una stazione per volta.
Nelle reti ad-hoc il PCF non può essere usato.

1.5.1 DCF: Distribuited Coordination Function

Quando si usa il DCF prima che una stazione inizi a trasmettere, questa controlla che il canale non sia occupato da altri. Se il canale rimane libero per un intervallo di tempo superiore al *Distribuited InterFrame Space* (DIFS), la stazione inizia a trasmettere. Il primo pacchetto inviato contiene la durata della trasmissione prevista. Tale valore viene memorizzato da tutte le atre stazioni nel *Network Allocation Vector* (NAV), quindi il NAV contiene il periodo di tempo in cui il canale rimarrà occupato.
Il CSMA/CA non permette di rilevare le collisioni ascoltando il canale, quindi c'è bisogno di inviare un acknowledge (ACK) frame ogni qualvolta si riceve un messaggio che non contiene errori. Se chi invia un messaggio non riceve l' ACK deve programmare una ritrasmissione del messaggio. La rilevazione degli errori è affidata ad un algoritmo CRC.

L' ACK frame viene inviato dopo aver atteso un tempo pari al *Short InterFrame Space* (SIFS)dalla ricezione dei dati.
Se si verifica un errore di trasmissione o una collisione, il canale deve rimanere libero per almeno un *Extended InterFrame Space* (EIFS).
Al termine dell'EIFS viene attivato l' algoritmo di backoff che schedula la ritrasmissione.
Per ridurre la probabilità di collisioni, lo IEEE 802.11, usa un meccanismo di backoff: quando il canale viene trovato occupato, la stazione attende che il canale diventi libero; a questo punto non trasmette direttamente il messaggio, ma inizializza un contatore (backoff timer) selezionando un intervallo casuale (backoff interval) al termine del quale trasmetterà il messaggio.

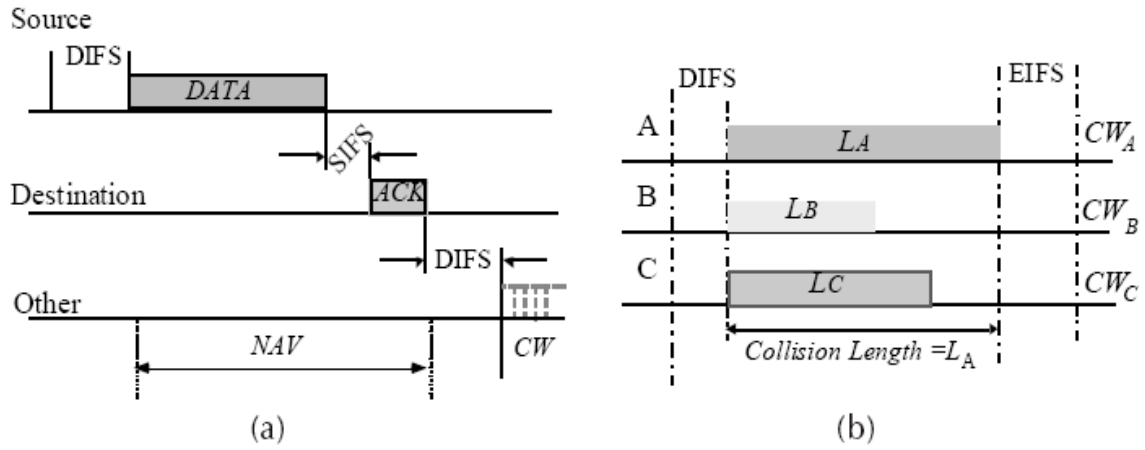

fig. 1.3 – (a) trasmissione conclusa con successo; (b) collisione

Una WLAN basata sullo standard IEEE 802.11 può essere implementata o con un approccio infrastruttura o utilizzando il paradigma ad-hoc. In quest' ultimo caso vengono chiamate Indipendent Basic Service Set (IBSS).
Essendo il CSMA/CA un protocollo molto flessibile, per poter funzionare in ad-hoc è sufficiente che tutte le stazioni siano sincronizzate.
L' IEEE 802.11 usa due funzioni principali per sincronizzare le stazioni:

- Acquisizione della sincronizzazione: questa funzionalità è necessaria per potersi unire ad un'IBSS preesistente. Consiste in una fase di ricerca delle IBSS effettuata scandendo tutte le frequenze radio della banda alla ricerca di un particolare frame di controllo. Se tale frame viene trovato, allora la stazione si sincronizza altrimenti crea una nuova IBSS.
- Mantenimento della sincronizzazione: a causa della mancanza di una stazione centrale che fornisce un clock comune, viene trasmesso un

segnale a intervalli regolari ad una frequenza nominale nota a tutti. L' intervallo di tempo è deciso dalla stazione che crea l'IBSS.

Un problema molto sentito nella progettazione delle WLAN IEEE 802.11 che adottano come metodo d'accesso al mezzo trasmissivo il CSMA/CA è quello delle stazioni nascoste. Due stazioni si dicono nascoste fra loro se non possono ascoltare le trasmissioni l'una dell'altra.
Questo problema può causare un incremento delle collisioni, in quanto se una stazione trova il canale libero non è detto che lo sia realmente perché potrebbe essere occupato da una stazione troppo lontana di cui, quindi, non riesce a sentire le trasmissioni.
Per ovviare a questo problema il DCF è stato esteso con il *Request To Send/Clear To Send* (RTS/CTS). Una volta guadagnato l'accesso al mezzo, la stazione trasmittente, prima di iniziare la trasmissione dei dati, invia alla stazione ricevente un pacchetto RTS. Se il ricevente è pronto a ricevere i dati risponde con un CTS. Questi due pacchetti contengono la durata prevista della trasmissione che viene memorizzata da tutte le stazioni nel NAV.
Usando questo sistema la collisione può avvenire solo quando s'invia un RTS.

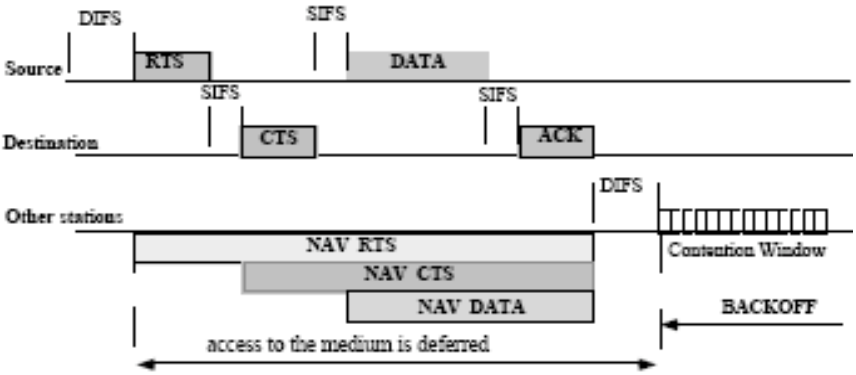

fig. 1.4 – RTS/CT

1.5.2 Bluetooth

Le reti Bluetooth appartengono alla categoria delle reti contention-free token-based multi-access ovvero quelle reti in cui il protocollo d'accesso al mezzo trasmissivo non è a contesa ma è gestito da un master che decide a quale stazione assegnare il mezzo trasmissivo.

In una rete Bluetooth, una stazione ha il ruolo di master e tutte le altre quello di slave. La stazione master decide quale degli slave può accedere al canale. Le stazioni che condividono lo stesso canale, ovvero che sono gestite dallo stesso master, formano una *piconet*, che è il blocco fondamentale delle reti Bluetooh. Una piconet contiene una stazione master e fino a sette stazioni slave attive contemporaneamente. Ciò significa che, in una piconet, possono esserci delle stazioni sincronizzate con il master, ma che non partecipano ad alcuno scambio. Tali stazioni sono in uno stato detto *parcking state*.
Può succedere che delle piconet indipendenti abbiano delle aree di sovrapposizione ovvero ci sono stazioni attive in più di una piconet. In questo caso, le piconet sovrapposte, formano una *scatternet*.
Uno slave può, quindi, far parte di più piconet contemporaneamente, ma può trasmettere in una piconet per volta. Questo accade perché uno slave può essere sincronizzato con il clock di una piconet per volta. Per trasmettere in un'altra piconet deve cambiare i parametri di configurazione.
Lo standard Bluetooth definisce un insieme di protocolli per i diversi livelli dello stack: Bluetooth radio, Baseband, Link Manager Protocol (LMP) e Logical Link Control and Adaptation Protocol (L2CAP).
Bluetooth radio è il protocollo del livello fisico, definisce, quindi, la comunicazione tre i dispositivi a livello fisico, Baseband fornisce il servizio di trasporto dei pacchetti sul livello fisico. Il protocollo L2CAP è usato solo per la trasmissione dei dati. I servizi principali forniti da tale protocollo sono il protocol multiplexing, segmentazione e ricomposizione.
L'ultima funzione è necessaria perché i pacchetti del Baseband sono più piccoli rispetto a quelli usati dai protocolli posti a livelli superiori.
Le stazioni operano in una banda di 2.4 GHz, in cui sono definite settantanove frequenze radio suddivise in canali di 1 MHz.
Lo strato fisico utilizza la tecnica di trasmissione *Frequencing Hopping Spread Spectrum* (FHSS). Tale sistema è stato scelto per ridurre le interferenze tra sistemi vicini che operano nello stesso range di frequenze e per garantire maggiore sicurezza nelle trasmissioni.
Nel sistema FHSS il segnale ad una data frequenza viene fatto "saltare" da una canale all'altro, distribuendosi su una banda di frequenze. La tecnologia consente a più utenti di condividere lo stesso insieme di frequenze cambiando automaticamente la frequenza di trasmissione fino a 1600 volte al secondo, al fine di una maggiore stabilità di connessione e di

una riduzione delle interferenze tra canali di trasmissione. Lo spectrum spreading consiste in una continua variazione di frequenza utilizzando una modulazione di frequency hopping. Gli hops corrispondono ai salti di frequenza all'interno della gamma assegnata (2,402 Ghz -2,480 Ghz salti di 1 Mhz, complessivamente 79 hops set).
Il sistema FHSS risulta, quindi, molto sicuro contro interferenza e l'intercettazione in quanto risulta statisticamente impossibile poter ostruire tutte le frequenze che possono essere usate.

1.5.2.1 Piconet e scatternet

Una stazione che appartiene ad una piconet prima di iniziare la trasmissione dei dati, deve scoprire se ogni altra stazione è nel suo spazio operativo. Per fare questo la stazione entra nell'*inquiry state*. Quando è in questo stato, la stazione invia in continuazione un inquiry message, che contiene solo il codice di accesso.
Durante questa fase la stazione utilizza una sequenza di frequency hopping composta di 32 sequenze derivate dal codice d'accesso. Tali frequenze sono divise in due successioni di 16 frequenze. Ogni sequenza da 16 frequenze deve essere ripetuta almeno 256 volte prima di passare alla successiva. Per garantire un numero sufficiente di risposte bisogna utilizzare molte sequenze.
Una stazione può rispondere all'inquiry message solo se sta ascoltando il canale alla ricerca di un inquiry message e solo se è sintonizzato alla stessa frequenza della stazione trasmittente.
Un'unità che riceve un inquiry message non è obbligata a rispondere, ma se lo fa invia un pacchetto FSH che contiene il suo indirizzo Bluetooth e il suo clock nativo.
Una volta terminata la fase di inquiry l'unità ha scoperto gli indirizzi e clock delle unità attorno ad essa. A questo punto se volesse attivare una nuova connessione non deve far altro che distribuire il proprio indirizzo e il proprio clock utilizzando le *routine di paging*. L'unità che inizia il paging è automaticamente eletta master della connessione. Tale unità un pacchetto detto page message che contiene solo il Device Access Code (DAC) che viene direttamente derivato dall'indirizzo dell'unità slave.
Dopo questa procedura lo slave ha piena conoscenza del clock del master e del codice di accesso al canale, quindi il master e lo slave possono entrare

nello stato di trasmissione. La trasmissione vera e propria inizierà solo quando il master invierà un polling message allo slave.
Lo slave mantiene la sincronizzazione con il master ascoltando il canale per ogni slot di trasmissione dal master allo slave.
Gli altri slave che non partecipano alla comunicazione una volta ascoltato il canale ritornano in basso consumo per tutta la durata della trasmissione.
Dato che molti dei dispositivi che usano il Bluetooth sono portatili, lo standard definisce alcuni stati per il risparmio energetico: Sniff, Hold e Park Mode.

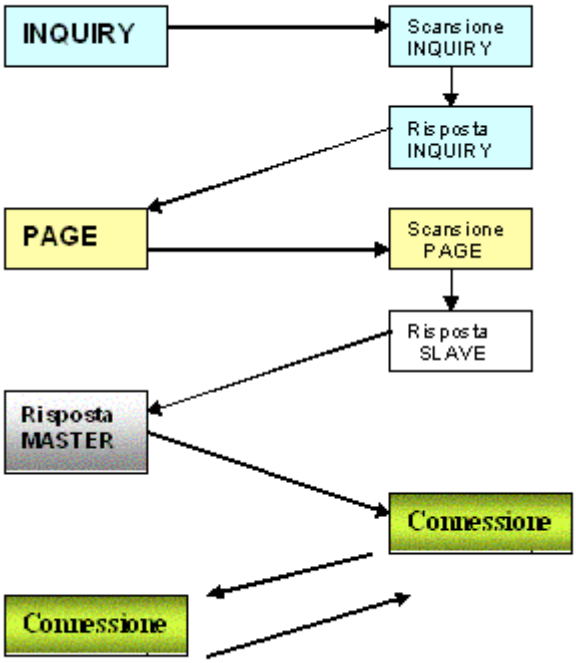

fig. 1.5 – Creazione della connessione tra dispositivi Bluetooth

Le specifiche del Bluetooth definiscono un metodo per interconnettere fra loro più piconet: le scatternet.
Si ha una scatternet ogniqualvolta una stazione appartiene a più piconet.
Tali reti vengono costruite dinamicamente utilizzando il paradigma ad-hoc.
La comunicazione tra le piconet avviene utilizzando l'unità slave condivisa.
Un nodo della rete può essere sincronizzato con un clock per volta, quindi può essere attivo in una piconet per volta. Ciò significa che per essere attivo in più piconet deve usare la modalità time-multiplexed, ovvero ogni volta che deve trasmettere in una piconet diversa dall'attuale, la stazione deve cambiare i parametri di trasmissione.

Lo standard Bluetooth definisce solo il concetto di scatternet ma non fornisce alcun meccanismo per realizzarle.

1.5.2.2 Trasmissione dei dati

Il Bluetooth stabilisce due modi per connettere fisicamente due dispositivi: *Synchronous Connection-Oriented* (SCO) e *Asynchronous Connection-Less* (ACL).
Il primo è una connessione simmetrica punto a punto tra master e slave ed è usato per trasmettere il traffico sensibile al ritardo come, ad esempio, la voce. L'SCO ha un link rate di 64 Kb/sec, ed è realizzato riservando due slot consecutivi per la trasmissione tra master e slave e per l'immediata risposta dello slave al master. L'SCO è, quindi, una connessione a commutazione di circuito tra il master e uno slave.
L'ACL è invece una connessione tra il master e tutti gli slave attivi della piconet, è, quindi, una connessione punto-multipunto. Può essere considerata come una connessione a commutazione di pacchetto.
L'accesso al canale è stabilito dal master utilizzando il polling. Il master decide quale stazione può trasmettere inviandole un pacchetto. Tale pacchetto può contenere o meno dei dati. Quando una stazione riceve il pacchetto dal master è autorizzata a trasmettere nel time slot successivo. Nel caso dell'SCO il master invia il pacchetto di polling periodicamente allo slave con cui sta comunicando, mentre nel caso ACL il polling è asincrono.
Una piconet ha un bit rate complessivo di 1 Mb/sec, ma il polling e i pacchetti di controllo dei protocolli limitano la quantità di dati utente che possono essere trasmessi da una piconet.

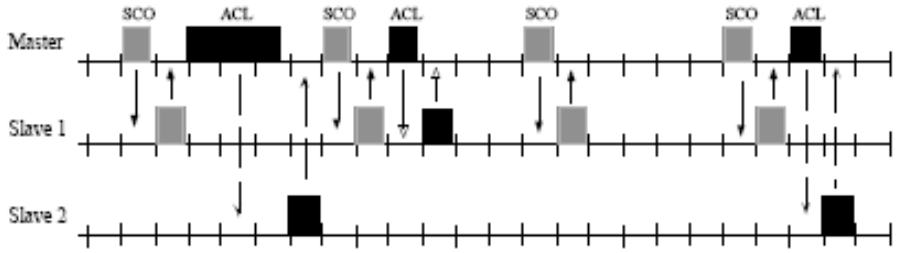

fig. 1.6 – Trasmissione in una piconet

Per quanto riguarda le politiche di gestione del polling da parte del master, le specifiche del Bluetooth indicano l'uso dell'algoritmo Round Robin (RR) per decidere l'ordine delle stazioni a cui inviare il pacchetto di polling. Utilizzando l'algoritmo RR l'ordine è ciclico. L'uso del RR in

caso di traffico non bilanciato può causare un grande spreco di banda, quindi ci sono state delle proposte alternative.
Una di tali proposte è chiamata *Efficient Double-Cycle* (EDC). Questo algoritmo regola l'ordine delle stazioni in base alle condizioni di traffico in modo da evitare lo spreco della banda causato dall'invio dei pacchetti di polling a stazioni che non trasmettono.
Il master deve, quindi, conoscere l'ammontare del traffico da e per lo slave che sta esaminando.
Il master conosce con precisione l'ammontare di traffico per lo slave, mentre, quanto riguarda il traffico in direzione opposta, può solo stimare la probabilità che uno slave invierà un pacchetto NULL. Tale probabilità viene calcolata utilizzando la conoscenza del comportamento dello slave nel ciclo di polling precedente.

1.6 Modelli wireless

Per implementare le reti wireless si possono utilizzare differenti modelli:

- Modello a cella: ricopre una certa area possibilmente divisa in celle sovrapposte in ognuna delle quali c'è una base station (BS). Le BS sono dei punti d'accesso alla rete. Essi sono collegati tra loro tramite una rete cablata. I nodi mobili possono spostarsi da una cella all'altra e comunicano tra loro per mezzo della BS presente nella cella in cui si trovano. In pratica il nodo mobile deve stabilire una connessione wireless con la BS per comunicare con gli altri nodi della rete. Se il nodo con cui si vuol comunicare è situato nella stessa cella la BS inoltrerà i messaggi al nodo attraverso un altro collegamento wireless, altrimenti invierà il messaggio attraverso la rete cablata alla BS della cella in cui è locato il nodo. La rete delle BS può fornire dei servizi ai nodi mobili: ad esempio può essere collegata alla rete telefonica per fornire accesso ad internet.
- Modello a cella virtuale (VCN): è molto simile al modello a cella. La differenza sostanziale sta nel fatto che nelle VCN anche le BS sono dei nodi mobili. Ogni BS coordina i nodi mobili situati nelle sue vicinanze. Essendo mobili anche le BS il grafo della rete delle BS può cambiare istante per istante quindi c'è bisogno di algoritmi distribuiti che lavorino incessantemente per mantenere stabile la rete.
- Modello ad-hoc: non ci sono stazioni fisse. Le reti ad-hoc sono delle reti peer-to-peer formate da un insieme di nodi che configurano se

stessi per formare dinamicamente reti temporanee. Nelle reti ad-hoc non c'è un controllore fisso quindi le decisioni vanno prese collegialmente da tutti i nodi. Essendo tutti i nodi della rete mobili, la posizione dei vicini varia in maniera nel tempo in maniera non predicabile quindi anche la topologia della rete varia in modo arbitrario.

Nel seguito del capitolo verranno trattate in particolare le reti ad-hoc focalizzando l'attenzione sul routing unicast e multicast.

1.7 Routing nelle reti ad-hoc

Il routing nelle MANET è un problema molto sentito, in quanto i nodi della rete sono mobili. La mobilità dei nodi porta a frequenti cambiamenti di topologia, quindi aumenta la complessità del routing tra i nodi della rete. Inoltre, per quanto riguarda il routing unicast, una stazione potrebbe voler comunicare con un'altra che non è nella sua area di copertura. Questo implica che i pacchetti dovranno attraversare diversi nodi della rete prima di giungere a destinazione.

I protocolli di routing sviluppati per le MANET, rispetto ai protocolli per le reti cablate, devono, quindi, far fronte a tutti i problemi riguardanti la mobilità dei nodi della rete, le limitazioni tecnologiche, come la durata limitata delle batterie, e problemi riguardanti la sicurezza.

1.7.1 Unicast Routing Protocol

Uno dei compiti principali dei protocolli di routing unicast è trovare e mantenere un percorso stabile tra una coppia di nodi della rete in modo da recapitare i messaggi in modo affidabile.

Questo è l'obiettivo dei classici link state e distance vector, ma le caratteristiche delle MANET fanno sì che tali protocolli non possano essere usati direttamente.

Infatti, tali protocolli sono stati progettati per delle reti con topologia statica, dove erano installati su particolari nodi della rete, in cui avevano a disposizione tutta la potenza di calcolo e l'energia sufficienti a portare a termina il proprio lavoro.

Nelle MANET, invece, ci sono frequenti cambiamenti di topologia e, inoltre, i protocolli di routing girano su dispositivi con risorse limitate.

Per risolvere il problema del routing unicast nelle MANET sono stati proposti molti protocolli, i quali adattano il link state o il distance vector alle MANET o propongono delle soluzioni diverse.
Tali protocolli devono avere delle caratteristiche che ottimizzano le risorse limitate e supportano le topologie dinamiche.
Una classificazione largamente diffusa dei protocolli di routing unicast per le MANET li divide in tre categorie: proattivi, reattivi e ibridi.

1.7.1.1 Protocolli proattivi

Sono quei protocolli che si basano sul mantenimento delle tabelle di routing. Il nome proattivi deriva dal fatto che calcolano ogni possibile percorso indipendentemente dall'effettivo utilizzo, rendendo quindi, subito disponibili i percorsi per ogni destinazione. Questa caratterista può essere utile, ad esempio, per le applicazioni interattive. Sono quei protocolli che adattano il link state e il distance vector alle MANET.
I meccanismi principali adottati nei protocolli proattivi sono i seguenti:

- Aumentare la quantità di informazioni sulla topologia memorizzate in ogni nodo in modo da evitare i loop e velocizzare la convergenza del protocollo;
- Variare dinamicamente la quantità degli aggiornamenti e la frequenza di aggiornamento;
- Ottimizzare il flooding;
- Combinare Link State e Distance Vector.

Nel paragrafo successivo verrà illustrato in dettaglio un esempio di protocollo proattivo.

1.7.1.1.1 Destination Sequenced Distance Vector

Il Destination Sequenced Distance Vector (DSDV) fu uno dei primi protocolli sviluppati per le MANET.
E' stato sviluppato adattando il classico Distance Vector (DV) alle MANET. I suoi punti chiave sono:

- Un meccanismo di aging basato su numeri di sequenza crescenti, che indicano la freschezza del percorso e sono usati per evitare i loop e il problema del conto all'infinito
- L'uso di aggiornamenti periodici dei percorsi completi o di parti del percorso in caso di cambiamenti di topologia

- Ritardare gli aggiornamenti dei percorsi che sono instabili, per i quali probabilmente è già in arrivo un aggiornamento del percorso causato da un cambiamento nella topologia.

Nelle entry delle tabelle di routing compare il numero di sequenza. Un numero dispari indica una stazione irraggiungibile, mentre i numeri pari vengono usati dalla destinazione per contraddistinguere gli aggiornamenti del percorso.
Le stazioni aggiornano i percorsi usando gli aggiornamenti provenienti dalle destinazioni.
Le entry della tabella di routing vengono aggiornate solo se il numero di sequenza dell'aggiornamento del percorso è maggiore di quello presente nella tabella o se ha lo stesso numero di sequenza ma il percorso ha una metrica migliore. Una entry viene cancellata della tabella se non arrivano informazioni sul percorso per più di un certo numero di intervalli di aggiornamento.
Nel protocollo DSDV, i percorsi con una metrica pari ad infinito vengono informati senza alcun ritardo. Per quanto riguarda gli altri nodi si applica un ritardo basato sul *settling time* medio. Facendo questo è possibile prevenire l'invio d'aggiornamenti riguardanti percorsi instabili, i quali saranno presto rimpiazzati con percorsi migliori in arrivo in un futuro molto prossimo. Ciò può accadere perché gli aggiornamenti per la stessa destinazione possono arrivare ad un nodo in ogni ordine.
Per calcolare il settling time medio i nodi utilizzano una tabella che contiene per ogni entry: l'indirizzo di destinazione, l'ultima misurazione del settling time e il settling time medio per quella destinazione.
Gli aggiornamenti dei percorsi vengono inviati periodicamente e in caso di cambiamenti della topologia della rete.

1.7.1.2 Protocolli reattivi

Un approccio diverso al problema del routing unicast è quello usato nei protocolli reattivi.
Tali protocolli si basano sul routing a richiesta: il percorso viene calcolato solo quando la stazione deve trasmettere.
Scompaiono, quindi, le tabelle di routing e la conseguente loro gestione.
Essendo basati sul routing a richiesta calcolano il percorso prima di trasmettere quindi se i nodi non generano traffico l'attività di routing è

completamente assente. Il nome reattivi deriva proprio da quanto detto in precedenza.
Per gestire i percorsi, i protocolli reattivi, usano le seguenti procedure:

- Scoperta dei percorsi
- Manutenzione dei percorsi
- Eliminazione dei percorsi

L' invio dei dati è invece realizzato principalmente attraverso due tecniche: source routing e hop-by-hop.
La procedura di scoperta dei percorsi è realizzata attraverso un ciclo query-reply che usa il flooding per propagare le query.
Quando la destinazione riceve una query genera almeno una reply.
La scoperta dei percorsi viene innescata in modo asincrono: quando una stazione deve trasmettere dei dati e non conosce un percorso per la destinazione, allora attiva la ricerca del percorso. Tale richiesta non viene fatta per la trasmissione di ogni singolo pacchetto perché i percorsi scoperti sono validi per un periodo di tempo.
Come risultato della scoperta dei percorsi, la stazione, acquisisce un nuovo "routing state", che memorizza il percorso scoperto.
I percorsi scoperti vengono mantenuti per mezzo della procedura di manutenzione dei percorsi finché vengono usati, altrimenti i percorsi vengono cancellati.
Nel paragrafo successivo è riportato un esempio di protocollo reattivo.

1.7.1.2.1 Ad-hoc On Demand Distance Vector

Il protocollo Ad-hoc On Demande Distance Vector (AODV) usa i numeri di sequenza per sostituire gli elementi della cache troppo vecchi e per evitare i loop.
La procedura di scoperta dei percorsi viene attivata da un nodo S che deve inviare un pacchetto ad un nodo D e non ha nella cache un percorso valido per raggiungere D.
La scoperta dei percorsi viene effettuata diffondendo per mezzo del flooding un pacchetto RREQ.
Appena un nodo invia un pacchetto RREQ, costruisce automaticamente un percorso inverso da se stesso al nodo sorgente S memorizzando l'indirizzo del nodo da cui ha ricevuto il primo RREQ.

Allo stesso modo quando un RREQ raggiunge la destinazione, il nodo costruisce tutti i percorsi inversi per raggiungere la sorgente. Una volta scelto il migliore, gli altri vengono cancellati dopo un certo timeout.
Alla fine della fase di scoperta dei percorsi, un nuovo routing state è creato dal nodo. Il routing state è composto dalle entry che memorizzano il nodo successivo presente sul cammino per raggiungere la destinazione.
I percorsi vengono cancellati se non vengono usati entro un tempo dato (*route expiration time interval*). Ogni volta che il percorso viene usato il timer viene resettato.
La manutenzione dei percorsi è basata sulla trasmissione periodica del messaggio HELLO. Appena un nodo rileva un link-failure invia un pacchetto Unsolicted Route Reply a tutti i suoi vicini che lo precedono nel percorso verso la destinazione invalidando tutti i percorsi che usano quel link. Tali nodi inviano a turno il pacchetto a tutti i loro vicini che li precedono nel percorso, in modo tale da avvisare tutti i nodi attivi. Quando il pacchetto Unsolictec Route Reply raggiunge la sorgente questa emette un pacchetto RREQ per cercare un nuovo percorso per raggiungere il nodo destinazione.

1.7.1.3 Protocolli ibridi

I protocolli ibridi associano alcune caratteristiche dei protocolli proattivi con alcune dei protocolli reattivi.
In tali protocolli il calcolo dei cammini avviene in parte in modo proattivo, in parte in modo reattivo: generalmente per i nodi vicini è possibile calcolare i percorsi in modo proattivo, mentre per gli altri nodi che non appartengono alla località i percorsi vengono calcolati in modo reattivo.
Questo tipo di strategia è corretta se le applicazioni che utilizzano i protocolli routing prediligono la comunicazione locale a quella con i nodi lontani. Le simulazioni dimostrano, tuttavia, che i protocolli ibridi hanno delle prestazioni migliori sia dei protocolli proattivi che di quelli reattivi, grazie ad un throughput maggiore e ad una latenza e overhead inferiori.

1.7.1.3.1 Zone Routing Protocol

Lo Zone Routing Protocol (ZRP) è un protocollo ibrido che ha lo scopo di combinare i vantaggi degli approcci reattivi e proattivi in modo da ridurre

la latenza necessaria ad acquisire un nuovo percorso, e l'overhead del protocollo.
Un punto centrale dello ZRP è il concetto di *zona*. Una zona Z(k,n) per un nodo n con raggio k è definita come l'insieme di tutti i nodi ad una distanza massima di k hop da n. Il nodo n è detto nodo centrale della zona di routing, mentre i nodi che sono a distanza k da n sono detti nodi periferici di n. Il valore di k è generalmente più piccolo delle dimensioni dell'itera rete e può essere ottimizzato in base alle condizioni di traffico e di mobilità dei nodi.
Il protocollo è organizzato in quattro componenti principali: IntrAzone Routing Protocol (IARP), IntErzone Routing Protocol (IERP), Bordercast Protocol (BRP) e il protocollo di livello due Neighbor Discovery/Maintenance Protocol (NDP).
Lo IARP è la parte proattiva dello ZRP. Viene usato per scoprire i percorsi all'interno della zona. Usa un approccio proattivo, ma gli aggiornamenti dei percorsi non vengono propagati oltre k hop.
Lo IARP usa NDP per scoprire informazioni sui nodi vicini.
Per i nodi situati ad una distanza maggiore di k, ZRP usa lo IERP per scoprire i percorsi. Lo IERP usa un approccio reattivo, quindi i percorsi per un nodo che si trova a più di k hop da n vengono calcolati su richiesta.
Lo IERP utilizza un flooding selettivo che utilizza la struttura costruita dallo IARP. In pratica vengono inviate le richieste solo ai nodi periferici della zona, usando un particolare tipo di trasmissione multicast chiamato *bordercast*. Quando un nodo riceve una query, se la destinazione fa parte della sua zona di routing, replica alla sorgente inviando un pacchetto di reply, oppure invia la query utilizzando il bordercast. Il percorso scoperto può essere accumulato nel pacchetto di query, oppure in quello di reply.
Quando s'inviano dei pacchetti lungo un percorso esterno alla zona di routing si usa un routing modificato: il percorso contiene solo i nodo periferici che devono essere attraversati. Invece se il pacchetto viene inviato ad un nodo all'interno della zona allora si sfruttano le tabelle di routing costruite dallo IARP.
Dato che non c'è coordinamento tra i nodi molto spesso le zone si sovrappongono, può accadere quindi che un nodo faccia parte di molte zone. Questo può portare ad avere delle performance peggiori rispetto alla ricerca fatta con il flooding normale.

La manutenzione dei percorsi viene usata per i percorsi fuori della zona. Tale procedura ha lo scopo di riparare i broken link cercando di ridurre la necessità di una ricerca dei percorsi globale.

1.7.2 Routing multicast

Lo scopo del routing multicast è quello di permettere la comunicazione tra i membri di un gruppo: se un membro invia un messaggio, questo deve essere ricevuto da tutti gli altri componenti del gruppo.
Un gruppo non è nient'altro che un insieme di entità che hanno uno scopo comune. Tutte le entità di un gruppo sono identificate con lo stesso indirizzo.
Un'entità può appartenere a più gruppi contemporaneamente e un nodo della rete può ospitare una o più entità, quindi un nodo è membro di tutti i gruppi a cui appartengono le entità del nodo stesso.
L'appartenenza ad un gruppo è dinamica: un'entità può unirsi o uscire da un gruppo in ogni istante. Tipicamente non ci sono restrizioni riguardo la posizione dei membri di un gruppo o sul loro numero.
Un host non deve necessariamente appartenere al gruppo per poter inviargli un messaggio.
I nodi che provvedono ad inoltrare i pacchetti multicast sono detti router multicast. Ogni router costruisce e mantiene una struttura di distribuzione dei pacchetti multicast. Tali strutture connettono i trasmettitori e i ricevitori di ogni gruppo, permettendogli di comunicare.

1.7.2.1 On-Demand Multicast Routing Protocol

L'On-Demand Multicast Routing Protocol (ODMRP) usa il concetto di forwarding group cioè solo un sottoinsieme di nodi inviano i pacchetti multicast.
In ODMRP l'appartenenza ai gruppi e i percorsi multicast sono stabiliti e aggiornati a richiesta dal nodo sorgente.
Quando un nodo S deve inviare dati ad un gruppo multicast e non conosce alcun percorso per raggiungere il gruppo, invia in broadcast all'intera rete un pacchetto JOIN_DATA. Tale pacchetto viene anche inviato periodicamente per aggiornare l'appartenenza ai gruppi e per aggiornare i percorsi. Quando un nodo intermedio riceve un JOIN_DATA memorizza nella propria cache l'ID della sorgente e il numero di sequenza del

messaggio in modo da individuare eventuali duplicati, mentre la tabella di routing viene aggiornata inserendo l'ID del nodo da cui il messaggio è stato inviato in modo d costruire il percorso inverso per risalire alla sorgente.
Se il messaggio non è un duplicato e il time-to-live (TTL) è maggiore di zero viene rinviato in broadcast.
Quando un JOIN_TABLE raggiunge un ricevitore multicast, viene creata e inviata in broadcast ai vicini un JOIN_TABLE. Quando un nodo riceve questo tipo di messaggio, controlla se una delle entry combacia con il proprio ID. Se ciò accade, il nodo capisce che si trova sul percorso verso la sorgente e che quindi fa parte del forwarding group e setta il forwarding group flag (FG_FLAG). A questo punto invia in broadcast la join table. Il campo next hop node ID viene riempito estraendo informazioni dalla tabella di routing. In questo modo, ogni membro del forwarding group propaga la JOIN_TABLE utilizzando il percorso più breve.
Quando un nodo riceve un JOIN_TABLE deve anche costruire la propria tabella di routing multicast per poter inviare pacchetti multicast.
Questo processo costruisce o aggiorna i percorsi dalla sorgente al ricevente e costruisce una rete di nodi chiamati forwarding group.
Una volta creati i forwarding group e scoperti i percorsi, le sorgenti possono inviare pacchetti alle stazioni riceventi. Finché ha dei dati da inviare, la sorgente invia periodicamente il pacchetto JOIN_DATA per aggiornare i percorsi e i forwarding group. Quando un nodo riceve dei dati multicast, li invia solo se non sono duplicati e se ha l'FG_FLAG attivo.
Questa procedura minimizza l'overhead ed evita di inviare pacchetti utilizzando percorsi vecchi.
In ODMRP non ci sono pacchetti speciali usati per unirsi o per uscire da un gruppo. Se una sorgente multicast vuole uscire da un gruppo, non deve far altro che non inviare più i pacchetti JOIN_DATA.
Se un ricevitore multicast vuole abbandonare il gruppo non deve inviare il join reply a quel gruppo.
I nodi dei forwarding group sono rimossi da quel ruolo se non ricevono per un certo tempo una JOIN_TABLE.

CAPITOLO 2
AGAPE: ALLOCATION AND GROUP-AWARE PERVASIVE ENVIRONMENT

AGAPE è un middleware per la gestione dei gruppi che fornisce le funzioni di supporto per applicazioni collaborative in scenari pervasivi.
AGAPE adotta la metafora gruppo come caratteristica base per abilitare i servizi e la condivisione delle risorse.
Un gruppo non è altro che un insieme di entità che hanno delle caratteristiche comuni gestite a livello applicativo.
La caratteristica principale di AGAPE è quella di essere un middleware location-aware, quindi la gestione dei gruppi viene effettuata in base alla posizione degli utenti e dei dispositivi. Il tutto si basa sull'assunzione che utenti situati nella stessa località collaboreranno più facilmente di quelli situati in località differenti.
Quindi AGAPE è progettato in modo tale da ridurre la necessità di avere delle viste globali di tutti gli utenti presenti nella rete, mentre realizza delle viste dipendenti dalla località le quali sono rapidamente disponibili anche in caso di partizionamenti della rete.
Tali viste contengono solo le entità presenti nella località, quindi per avere delle viste globali bisogna unire le viste locali.
Nelle reti cablate i sistemi per la gestione dei gruppi sono progettati per mantenere una visione globale e sincronizzata di tutti gli appartenenti al gruppo. I gruppi vengono costruiti utilizzando liste di membri attivi e connessi. Quando ci sono dei cambiamenti in tali liste questi vengono immediatamente propagati a tutti i membri del gruppo.
Questi sistemi funzionano bene in reti con larga banda, in cui la comunicazione è affidabile, in cui la disconnessione di un utente è considerato un evento raro e in cui un utente si collega alla rete sempre dallo stesso punto.
Nelle MANET invece, la banda è limitata e gli utenti collegati sono in movimento e ci sono, di conseguenza, frequenti disconnessioni dalla rete. I dispositivi che compongono la MANET, inoltre, possono essere eterogenei, quindi avere diversa capacità di calcolo.
Quindi l'uso dei sistemi tradizionali per la gestione dei gruppi produrrebbe un overhead rilevante e per poter sincronizzare tutti i membri di un gruppo bisognerebbe usare solo dispositivi con un'alta capacità di calcolo. La gestione dei gruppi, quindi, non può essere affidata ai sistemi classici, ma va affidata a sistemi che possano far fronte alle problematiche introdotte dalle MANET.
Inoltre, nelle reti ad-hoc, è impossibile garantire nomi univoci, quindi anche i sistemi classici per la consegna dei messaggi non sono adeguati.

Per lo sviluppo di tali sistemi bisogna tener conto del fatto che probabilmente, utenti co-locati, richiederanno una reciproca collaborazione più facilmente che con utenti situati in aree diverse della rete, oltre alla necessità di sviluppare un sistema di gestione dei gruppi context-aware.
L' uso sistemi context-aware ha il vantaggio di permettere la costruzione rapida, di viste immediatamente disponibili nella località.
Il nuovo sistema dovrà, quindi, sfruttare le informazioni di contesto per dare visibilità a tutti i membri del gruppo, di tutti i vicini e dei loro profili.
In oltre si può ottenere una vista globale di tutti i membri del gruppo unendo le viste locali ottenute coordinando i vari gestori dei gruppi.
In base alle informazioni di contesto, il sistema, può stabile i percorsi di minor lunghezza tra i membri del gruppo, fornendo così un servizio migliore e utilizzando meglio la banda.

2.1 Il framework AGAPE

AGAPE supporta lo sviluppo di applicazioni context-aware collaborative in ambiente MANET. La collaborazione, in AGAPE, è basata sul concetto di gruppo.
Ogni gruppo è identificato da un GID (Group IDentifier) e da uno specifico profilo di gruppo, che specifica gli interessi, gli obbiettivi e le preferenze degli appartenenti al gruppo.
L' insieme dei membri di un gruppo può cambiare in modo dinamico a causa della mobilità degli utenti, connessione/disconnessione dei dispositivi e partizionamento/unione del gruppo.
Solo gli appartenenti ad uno stesso gruppo possono collaborare tra loro.

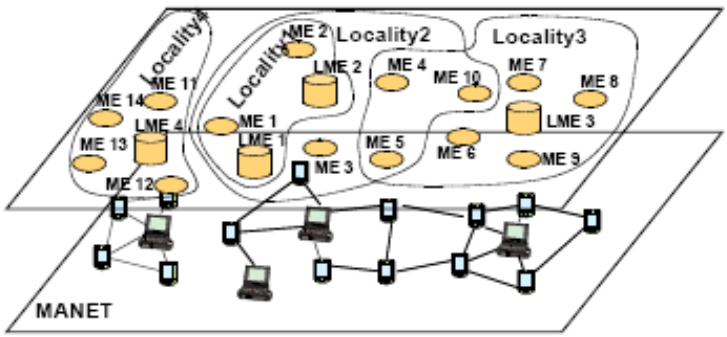

fig. 2.1 – Il modello AGAPE

Un altro concetto di base in AGAPE è la località.

La località consiste in una serie di entità che si trovano alla distanza pochi hop. Le entità in una località possono appartenere a gruppi diversi, quindi i gruppi possono essere disposti su più località. Possono, quindi, essere partizionati in sottoinsiemi disgiunti chiamati cluster. In una località possono coesistere più cluster appartenenti a gruppi diversi, ma può esserci solo un cluster per gruppo.
La figura 2.1 mostra alcuni esempi di località con h pari a 2 (h definisce il numero di hop): possiamo osservare che possono esistere località disgiunte oppure sovrapposte.
Il framework AGAPE identifica due tipi di entità diversi: LME, ME.
Gli LME sono i dispositivi con potenza di calcolo più elevata (ad es. computer portatili) e gestiscono le entità presenti nel cluster. Possono gestire più gruppi contemporaneamente.
Gli ME sono, invece, dispositivi con potenza di calcolo più bassa (ad es. palmari), che hanno un set di operazioni più limitato rispetto agli LME: non sono in grado di gestire i gruppi e fanno parte di un gruppo per volta.
Gli ME possono spostarsi liberamente attraverso le località e possono appartenere a più località contemporaneamente. Inoltre può accadere che due LME definiscano la stessa località, come ad esempio nel caso riportato in figura 2.1 in cui ci sono due LME a distanza di un hop.
Ogni entità, sia essa LME o ME, è caratterizzata da un PID (Personal IDentifier) che identifica univocamente l' entità all' interno di un gruppo.
Il GID e il PID vengono estratti in modo casuale: l' univocità degli stessi è garantita statisticamente.
Il framework AGAPE permette alle entità di creare gruppi, unirsi a gruppi già esistenti e di ricevere viste aggiornate dei gruppi preseti nella località.
Si possono ottenere viste globali dei gruppi unendo le viste locali.

2.2 Il middleware AGAPE

Il middleware AGAPE fornisce i servizi necessari per la gestione dei gruppi in un ambiente MANET.
I servizi principali forniti dal middleware sono: il PENS, il PS, l'NMS, il JLMS, il VMS e il VCS.
Il linguaggio usato per l'implementazione del middleware è JAVA.

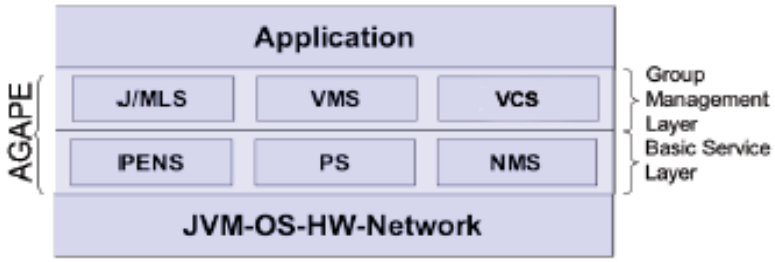

fig. 2.2 – Il middleware AGAPE

Possiamo dividere i servizi forniti da AGAPE in due livelli:
il Basic Service Layer e il Group Management Layer.
Il Basic Service Layer include tutti i servizi che gestiscono il naming delle entità, la comunicazione e la scoperta e il monitoraggio della disponibilità on-line dei membri.
Il Group Management Layer è uno strato di livello più alto che sfrutta il livello sottostante per realizzare la gestione dei gruppi, ovvero creazione di un gruppo, join/leave da un gruppo, gestione delle viste locali.

2.2.1 NMS: Network Management Service

L'NMS è il servizio del middleware che permette la comunicazione punto-punto e punto-multipunto basata sullo scambio di messaggi. L'NMS, quindi, astrae le primitive di comunicazione unicast, broadcast e multicast garantendo, a seconda della tecnologia di rete, il delivery efficiente dei messaggi. L'NMS prevede funzioni di controllo e funzioni di comunicazione.
Le prime servono alle applicazioni per scegliere quali interfacce di rete devono essere usate dall'NMS e di controllare l'attività, la modalità operativa e le proprietà delle interfacce configurate.
Per quanto riguarda le primitive di comunicazione, nell'NMS sono state sviluppate delle primitive connection-less che permettono la comunicazione punto-punto e punto-multipunto.
Le primitive punto-multipunto possono essere di tipo broadcast e multicast.

2.2.2 PS: Proximity Service

Questo servizio permette alle entità AGAPE, siano esse LME o ME, di diffondere la propria disponibilità on-line. Il PS invia in broadcast ad intervalli regolari un beacon a tutti i suoi vicini. Il beacon contiene la

coppia GID/PID e il ruolo ricoperto dall' entità. Inoltre contiene anche un TTL (Time To Live) che indica il numero di hop per cui il messaggio deve essere propagato.
Il nodo che riceve il messaggio decrementa il TTL e rinvia il beacon con un probabilità p(n), dove n è il numero delle entità vicine, che decresce al crescere delle entità vicine.
La disponibilità on-line di un' entità viene avvertita da tutte le altre entità presenti in un numero di hop pari al TTL siano esse facenti parte dello stesso gruppo o meno.

2.2.3 PENS: Proximity Enabled Naming Service

È il servizio che genera gli identificatori dei gruppi e delle singole entità in modo statisticamente unico. Inoltre fornisce le primitive che permettono di sapere i gruppi presenti nella località, tutte le entità presenti e i rispettivi ruoli (LME o ME). Per ogni entità fornisce inoltre: l' indirizzo IP, il GID, il PID e lo stato.
Il PENS costruisce e tiene aggiornata una tabella che contiene tutti dati sopra elencati.
Tale tabella viene costruita a partire dai beacon inviati in broadcast dal PS che contengono le informazioni riguardanti l' entità che li invia.
Quando non vengono più ricevuti beacon da una certa entità, viene cancellata dalla tabella la relativa entry.
Ogni volta che c'è un cambiamento nella tabella, viene notificato a tutte le entità interessate per mezzo di un evento.

2.2.4 J/LMS: Join/Leave Manager Service

Si occupa di gestire la creazione di gruppi, l'unione di un'entità ad un gruppo già esistente e l'uscita da un gruppo di un'entità. Oltre a questi servizi fornisce anche la possibilità di scoprire tutti i profili dei gruppi presenti nella località o di trovare solo quelli i cui profili superano il profile matching.
Sono state sviluppate due versione del J/LMS una per gli LME e una per gli ME. Tale distinzione è dovuta al fatto che gli ME hanno una potenza di calcolo più bassa e che quindi non sono in grado di gestire al meglio i gruppi.

Gli LME possono gestire contemporaneamente tutti i gruppi di cui fanno parte e possono creare nuovi gruppi. Gli ME invece non possono creare nuovi gruppi e possono far parte di un gruppo per volta anche se possono fare il join a più gruppi. Inoltre solo gli LME possono rispondere alle richieste di un entità di join o di discover dei profili. Gli ME hanno la funzione di switch ad gruppo che permette di passare dal gruppo esistente ad un altro gruppo di cui l' entità è membro.
Il J/LMS si coordina con gli altri servizi per poter realizzare le sue funzioni.
Quando viene creato un gruppo, il J/LMS, chiede al PENS di estrarre il GID per il nuovo gruppo e il PID del primo membro del gruppo, inoltre chiede al PS di diffondere l' esistenza del nuovo gruppo.
Quando, invece, viene fatto il JOIN ad un gruppo esistente, il J/LMS invoca il PENS per scoprire se ci sono degli LME del gruppo, poi se il join va a buon fine invoca il PS per diffondere la nuova coppia GID/PID.
Quando viene eseguita l' operazione di LEAVE da un gruppo viene solo chiesto al PS di non inviare più la coppia GID/PID.
Se invece, viene eseguita la funzione di SWITCH ad un altro gruppo viene invocato due volte il PS in modo da non trasmettere più la coppia GID/PID del vecchio gruppo e da iniziare a trasmettere la nuova coppia GID/PID.
In tutti questi casi il J/LMS si coordina anche con il VMS che ha l' incarico di gestire le viste dei gruppi.
Nel capitolo successivo verranno mostrati i dettagli implementativi del J/LMS.

2.2.5 VMS: View Manager Service

Ha l'incarico di creare, gestire e diffondere le viste di un gruppo.
Come per il J/LMS, anche per il VMS sono state sviluppate la versione LME e la versione ME. I motivi che hanno portato a questa scelta sono quelli descritti per il J/LMS.
Il VMS crea una nuova vista quando il join ad un gruppo va a buon fine, e cancella una vista quando viene fatto il leave da un gruppo.
L' LME che gestisce il gruppo invia a tutti i membri del gruppo periodicamente una vista chiamata CONTEXT VIEW. Tale vista contiene tutto l'elenco dei membri del gruppo presenti nella località e i rispettivi profili.

Il VMS interroga periodicamente il PENS in modo da controllare se qualche membro è uscito dal gruppo e quindi va eliminato dalla vista, e in modo da controllare se ci sono dei membri del gruppo che non fanno parte della vista perché sono appena entrati nella località o hanno appena fatto lo switch al gruppo.
In quest'ultimo caso il VMS richiede al membro l'invio del suo profilo, in modo da poterlo memorizzare nella context view.
La versione ME del VMS non aggiorna le viste interrogando il PENS, ma riceve le context view dagli LME della sua località e trattiene solo quelle con il numero di sequenza più alto.
Il VMS può essere interrogato in modo da fornire all' applicazione parte della vista in base al particolare profilo fornito.

2.2.6 VCS: View Coordination Service

Permette agli LME di decidere se distribuire le context view o meno. Il VCS aiuta a ridurre la propagazione di viste non necessarie: quando più LME di uno stesso gruppo fanno parte della stessa località, disseminano la stessa context view ai membri colocati, quindi ci sarebbe uno spreco di risorse.
Per risolvere questo problema ogni volta che un LME deve inviare la propria context view richiede l'autorizzazione del VCS.
Il VCS compara le entry delle viste ricevute dagli altri LME con la context view da inviare. Se le viste ricevute non hanno le stesse entry della context view il VCS autorizza l'invio della vista. Se, invece, le entry delle viste ricevute sono le stesse della context view da inviare, allora solo un LME può propagare la vista.
In quest'ultimo caso l'LME viene scelto in base al seguente criterio: può inviare la vista solo l'LME che ha nella sua vista il campo livello batteria maggiore di una soglia predefinita, se più LME hanno lo stesso livello batteria viene scelto quello con il PID maggiore.

CAPITOLO 3
JOIN/LEAVE MANAGER SERVICE (J/LMS)

Il Join/Leave Manager Service (J/LMS) è un servizio del middleware AGAPE che permette di gestire la creazione dei gruppi e l'appartenenza delle entità ai gruppi.
Le funzioni principali offerte dal servizio alle applicazioni sono:

- **Creazione di un gruppo:** permette di creare un nuovo gruppo fornendo un insieme di vincoli e un profilo di gruppo;
- **Join ad un gruppo:** permette all'entità di unirsi ad un gruppo esistente;
- **Leave da un gruppo:** permette all'entità di lasciare un gruppo di cui fa parte;
- **Discover di tutti i gruppi:** scopre, interrogando un LME per gruppo, i profili di gruppo;
- **Discover dei gruppi il cui profilo combacia:** scopre i profili di gruppo che superano il profile matching.
  Tale funzione è stata implementata in due modi diversi. Nel primo il profile matching viene svolto in locale nel secondo in remoto.

In AGAPE sono presenti due tipi di entità gli LME e gli ME che hanno differenti caratteristiche in fatto di capacità di calcolo e durata delle batterie.
Per questo motivo sono state sviluppate due versioni diverse del servizio.
In particolare la versione sviluppata per gli LME eredita tutti i casi d'uso sviluppati per gli ME, aggiungendo delle caratteristiche proprie al J/LMS.
La versione LME contiene la parte di gestione vera e propria dei gruppi, in quanto permette anche di creare nuovi gruppi.
Solo gli LME gestiscono i gruppi: quando un' entità effettua l'operazione di join o di discover, inoltra le richieste ad uno degli LME che gestisce il gruppo.
Inoltre sono state implementate delle primitive che permetto di interrogare il servizio in modo da ottenere alcune informazioni come, ad esempio, l'insieme dei GID dei gruppi di cui fa parte l' entità, o i profili dei gruppi scoperti oppure il PID dell' entità in un certo gruppo GID.
Gli ME hanno le stesse funzioni degli LME con alcune limitazioni: un ME non può creare un gruppo, anche se ha fatto il join a più gruppi, può far parte di un gruppo per volta, un ME non gestisce alcun gruppo e il leave è svolto in modo diverso.
Proprio perché un ME può far parte di un gruppo per volta è stata implementata la funzione di SWITCH. Tale funzione permette di passare

dal gruppo attuale, ad un altro gruppo a cui l' entità aveva fatto il join precedentemente.
Il servizio è stato sviluppato utilizzando altre parti del progetto AGAPE realizzate in altre tesi: PS, PENS, NMS, VMS e profili. Nel seguito verranno descritte tutte le funzioni fornite dal J/LMS con maggior dettaglio, sfruttando i diagrammi UML.

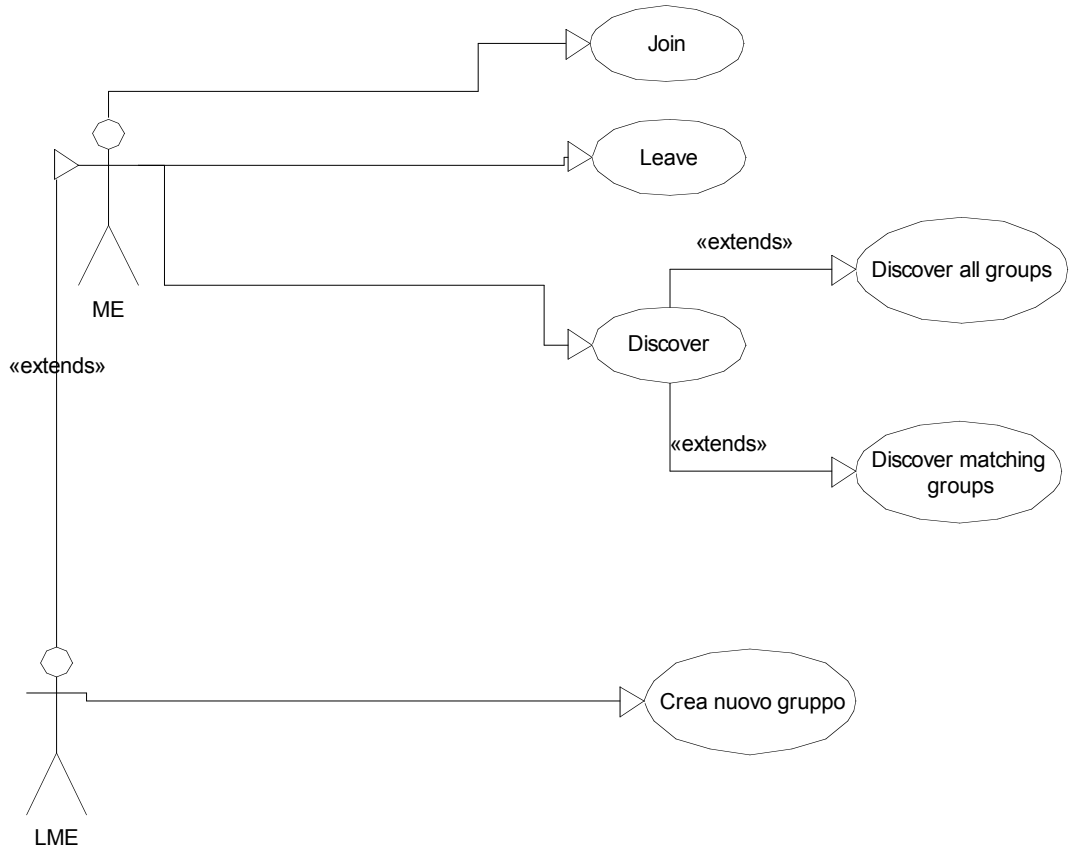

fig. 3.1 – J/LMS: casi d'uso

3.1 J/LMS LME

Il servizio è stato realizzato per mezzo del pattern singleton. La classe JLMService è la classe su cui vengono invocate le chiamate ai vari metodi che implementano le funzionalità del servizio. Tale classe estende la classe ActiveComponent che è un AgapeComponent attivo. Gli ActiveComponent hanno un thread che esegue il task per cui il componente è stato progettato. Possono essere attivati per mezzo del metodo start() e deattivati per mezzo del metodo stop(). Inoltre, invocando il metodo isActive(), è possibile sapere se il componente è attivo o meno. Il task da eseguire è realizzato nel metodo task().

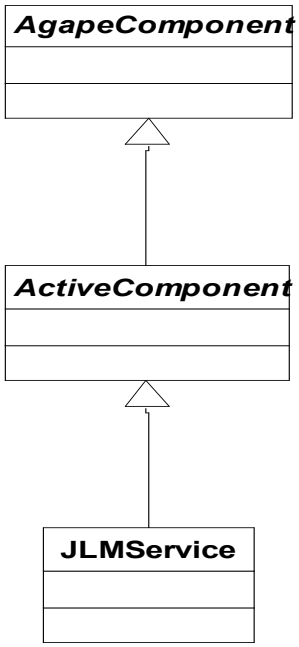

fig. 3.2 – Tassonomia della classe JLMService

Il metodo task dell'LME contiene, quindi, il thread che realizza la parte non statica del servizio.
L' invio delle richieste da parte delle entità agli LME avviene per mezzo di messaggi la cui struttura è mostrata in figura 3.2.
Quando viene attivato, il thread J/LMS, rimane in attesa dell'arrivo di o di un JoinRequestMessage o di un JLMSDiscoverRequest o di un JLMSMatchRequest.
Il JoinRequestMessage è il messaggio che contiene la richiesta di Join ad un gruppo. Viene inviato da un'entità ad un LME del gruppo a cui vuole unirsi.

Quando viene ricevuto un messaggio di questo tipo, LME elabora i dati contenuti in esso e risponde al mittente con un AckMessage, se il pari è stato accettato nel gruppo o con un NackMessage.
I messaggi JLMSDiscoverRequest e JLMSMatchRequest servono per richiedere ad un LME il profilo di gruppo. Quando viene ricevuto il primo l' LME risponde con un JLMSDiscoverResponse contenente il profilo di gruppo, se riceve il secondo, l' LME effettua prima il profile matching ed invia un JLMSMatchResponse. Il messaggio contiene un campo booleano che vale true se il profile matching ha dato esito positivo, false in caso contrario.

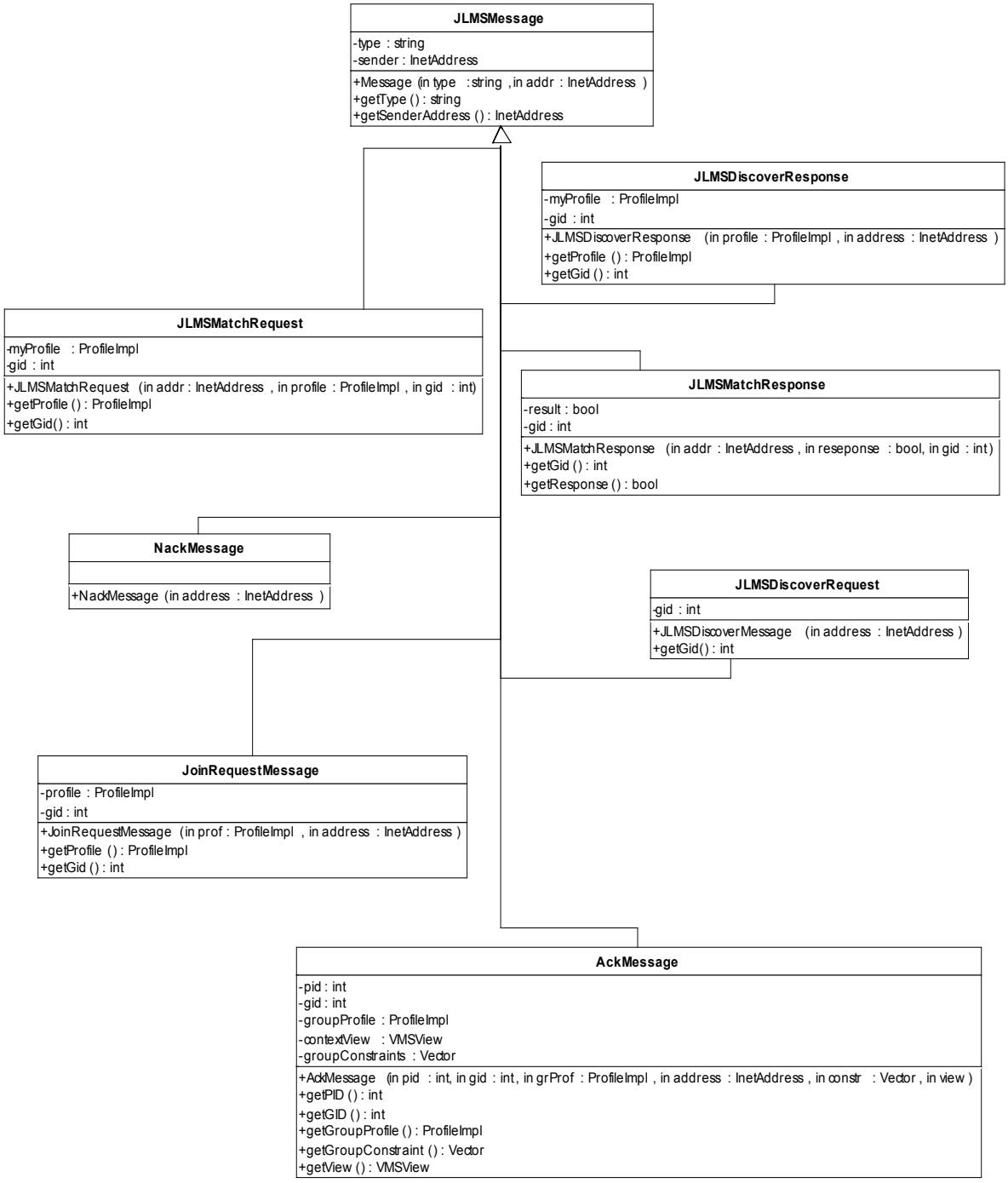

fig. 3.3 - Messaggi: diagramma statico delle classi

3.1.1 Diagramma statico delle classi

La classe principale del servizio è la JLMService. Questa classe rappresenta il servizio vero e proprio: le applicazioni invocano su di essa i metodi per creare nuovi gruppi, fare la discover dei gruppi, fare il join e il leave dai gruppi, oltre ai metodi per avere informazioni sui gruppi scoperti o sui gruppi di cui l'entità fa parte.
Le altre classi sono state sviluppate come supporto alla classe principale.
La classe JLMService, infatti, usa le altre classi per portare a termine le proprie funzioni: usa la classe NetUtil per trasmettere e ricevere i messaggi e usa i thread per lavorare in modo completamente asincrono.
Tutti metodi della classe sono statici.
Quando viene invocato il metodo startJLMS() viene attivato un thread che rappresenta il servizio. Tale thread attende in continuo l'arrivo di messaggi di richiesta di join o di discover dei profili di gruppo. Quando arriva un messaggio viene invocato un thread specifico che lo elabora e risponde al mittente con un altro messaggio.
Per l'invio e la ricezione dei messaggi, come per la creazione e la chiusura delle porte, vengono usati i metodi della classe NetUtil che si basano sull'NMS.
La classe JLMService, contiene inoltre, delle strutture dati per mantenere le informazioni relative ai gruppi a cui appartiene l'entità e ai gruppi scoperti.
Per quanto riguarda l'appartenenza ai gruppi, vengono gestite tre hashTable, una contiene le coppie GID/PID, una contiene le coppie GID/Profilo di gruppo e l'ultima contiene le coppie GID/constraints.
Per quanto riguarda la terza hashtable, i constraint sono i vincoli con cui effettuare il profile matching in caso di richiesta di join al gruppo.
Le informazioni riguardanti i gruppi scoperti vengono mantenute in due vector. Il primo contiene le informazioni riguardanti tutti i gruppi presenti nella località, il secondo solo quelle il cui profilo di gruppo combacia con quello dell'entità.

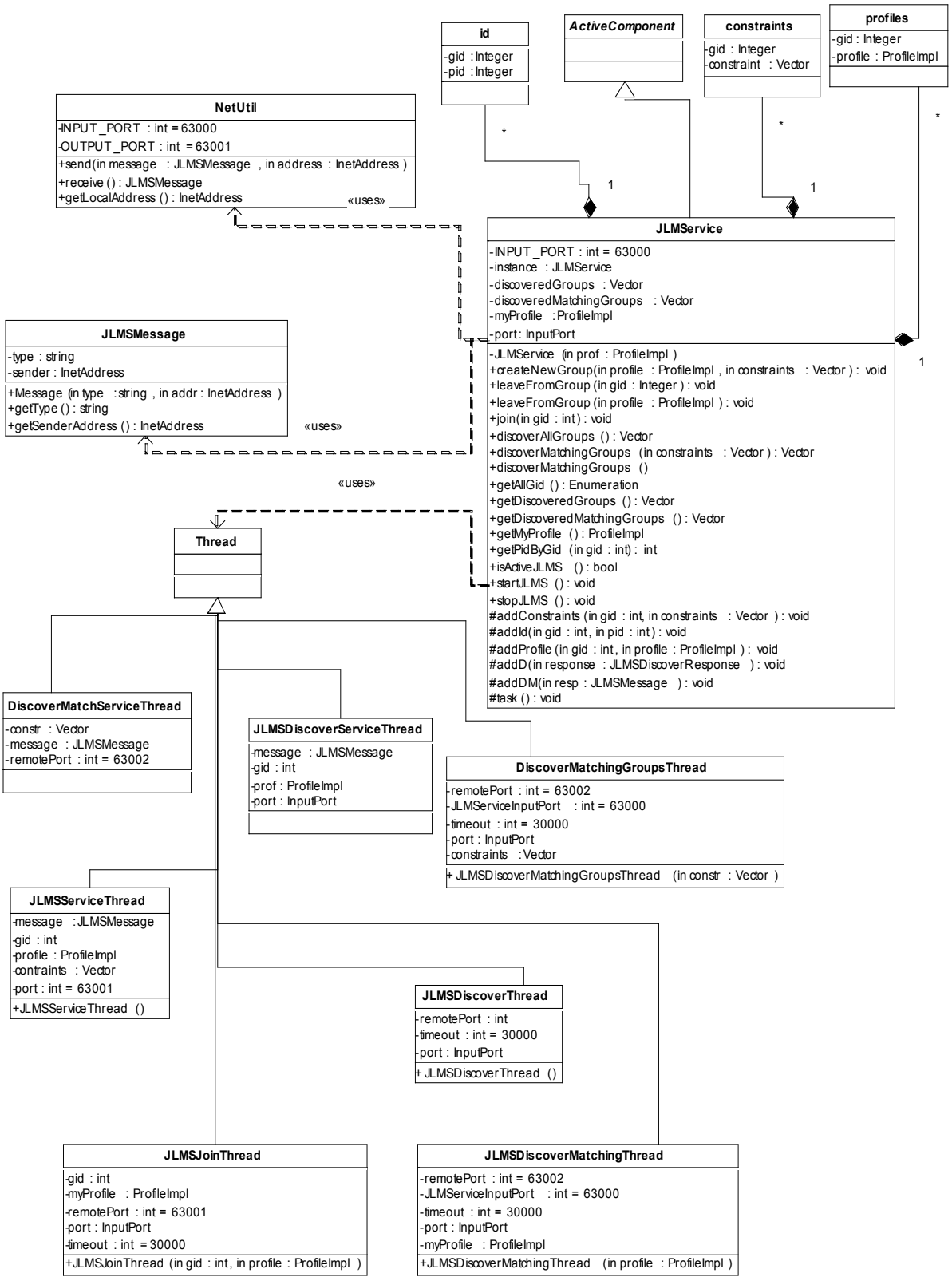

fig. 3.4 - Diagramma statico delle classi

3.2 J/LMS ME

Come per gli LME, anche per gli ME è stato scelto il pattern singleton per realizzare il servizio.
La classe principale del progetto è la JLMService, che eredita da ActiveComponent.
Dato che un ME non deve gestire i gruppi, allora il metodo task non contiene codice. Quindi la classe viene usata solo per la sua parte statica.
Il servizio usa i messaggi, la cui struttura è mostrata il figura 3.2, per scambiare informazioni con gli LME.
Quando vengono invocati i metodi del servizio, vengono creati dei thread che inviano le richieste agli LME e aspettano le risposte. Quando arrivano le reponse, i thread aggiornano le strutture dati del servizio e terminano la propria esecuzione.
Per l'invio e la ricezione dei messaggi vengono usati i metodi statici della classe NetUtil. In tale classe sono stati implementati anche dei metodi per la creazione e la chiusura delle porte.
Nella classe JLMService ci sono due hashtable che contengono le coppie GID/PID e le coppie GID/profilo di gruppo.
Tali tabelle vengono aggiornate quando si effettuano le operazioni di join o di leave.
Per mantenere i dati riguardanti l'operazione di discover ci sono due vector: uno per la discover di tutti i gruppi, l'atro per la discover dei gruppi il cui profilo combacia con quello dell'entità.
Il GID del gruppo di cui fa parte l'entità è quello contenuto nel campo actualGid.
L'entità gestisce a livello di VMS solo le viste riguardanti l'actualGid, mentre il PS invia solo le coppie GID/PID riguardanti quel gruppo.
La figura 3.5 mostra il diagramma statico delle classi del J/LMS di un ME.

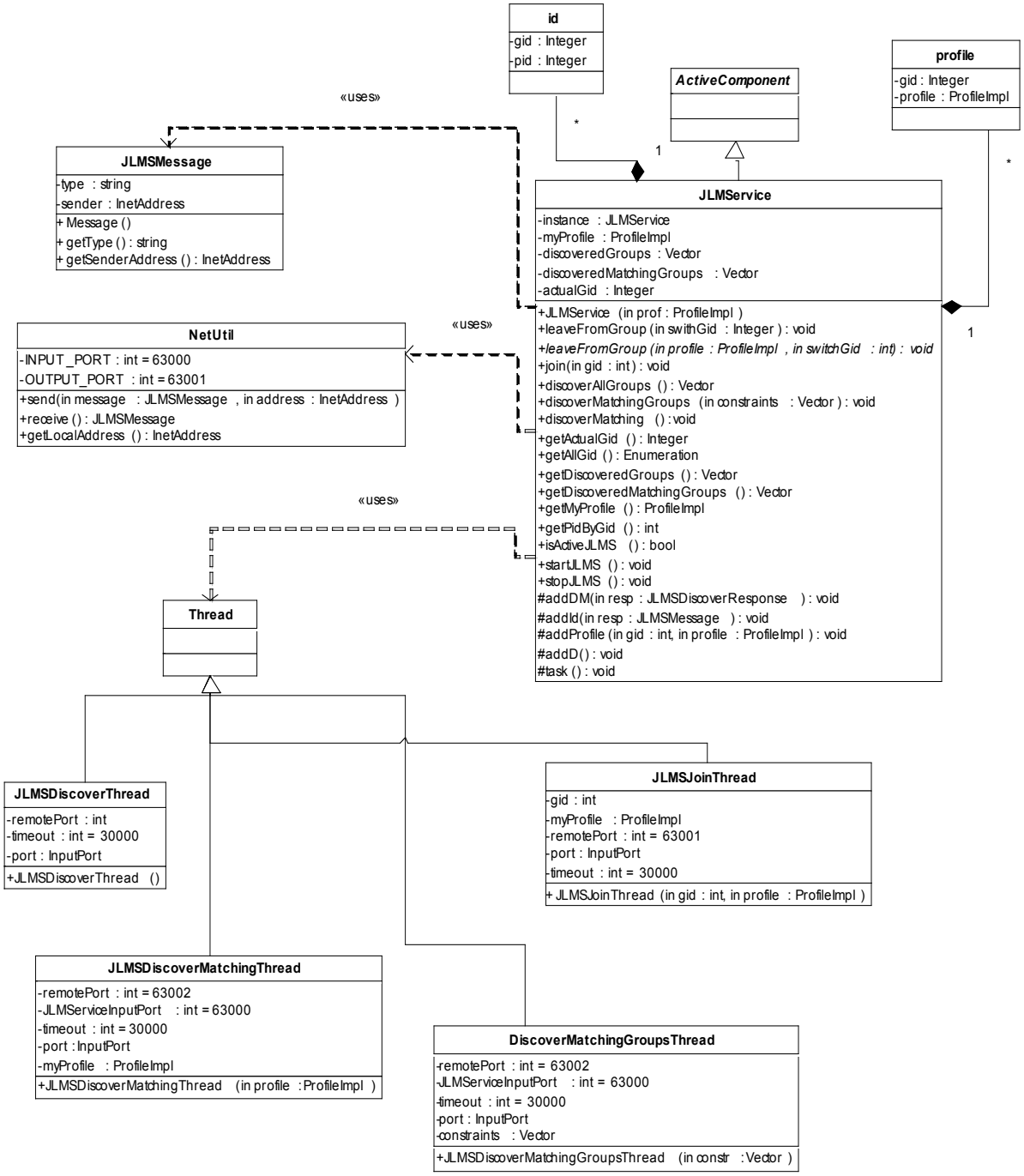

fig. 3.5 - J/LMS ME: diagramma statico delle classi

3.3 Creazione di gruppi

Si può creare un gruppo invocando il metodo CreateNewGroup passandogli come parametri il profilo di gruppo e i vincoli di gruppo.

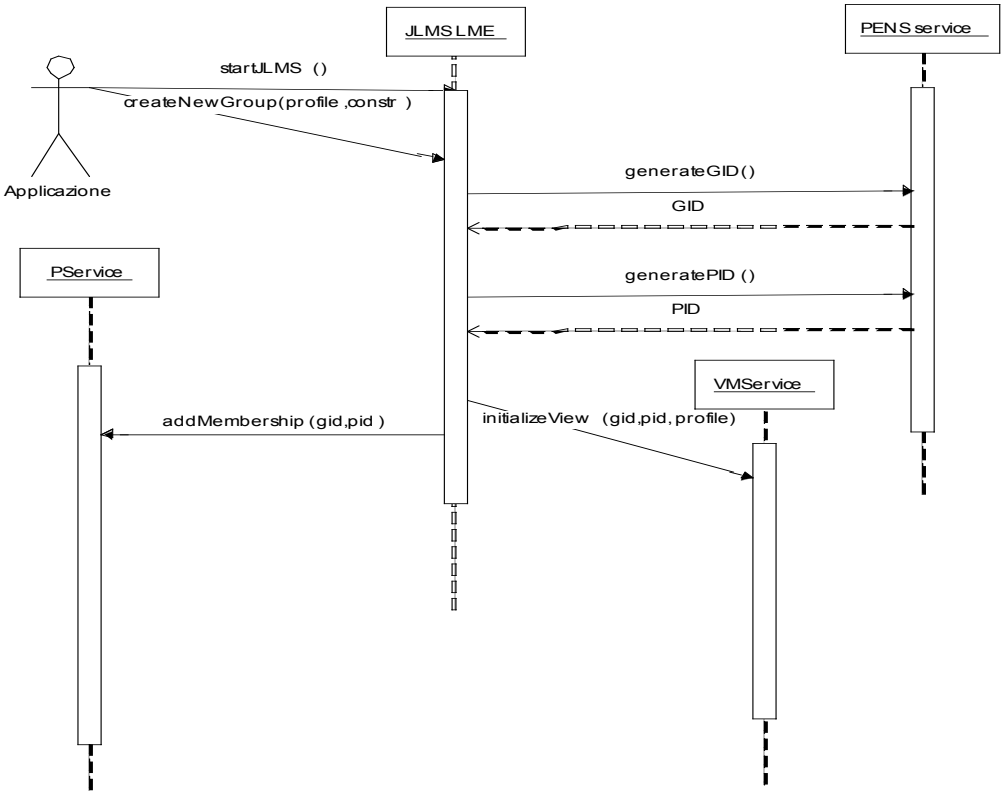

fig. 3.6 - Creazione di gruppi: diagramma di sequenza

Quando si invoca tale metodo il J/LMS invoca alcuni metodi del PENS, del PS e del VMS per portare a termine la creazione del gruppo.
Per prima cosa invoca i metodi generateGID() e generatePID() del PENS. Il primo serve per generare il GID del nuovo gruppo, il secondo per generare il PID dell'entità che sta creando il gruppo, che è di conseguenza il primo membro del gruppo.
Una volta generati GID e PID viene invocato il metodo initializeView del VMS che inizializza una nuova vista contenente solo un membro.
A questo punto vengono inseriti i dati riguardanti, il nuovo gruppo nelle hashtable e viene invocato il metodo addMembership del PS passando

come parametri il GID e il PID appena generati, per far si che il PS inizi a diffondere i beacon relativi al nuovo gruppo.
A questo punto la creazione del gruppo è terminata e i PENS che sono in esecuzione sulle entità presenti nella località rilevano la presenza del nuovo gruppo.

3.4 Join ad un gruppo

Per unirsi ad un gruppo presente nella propria località bisogna invocare il metodo Join a cui va passato come parametro il GID del gruppo a cui ci si vuole unire.
Quando un'applicazione invoca tale metodo, il J/LMS, sia esso versione LME o ME, crea un JLMSJoinThread che crea ed invia ad un LME, che appartiene al gruppo GID, un JoinRequestMessage.
Tale messaggio contiene il proprio profilo e il GID del gruppo a cui l'entità sta facendo il join.
Una volta inviato il messaggio, il thread, aspetta per trenta secondi l'arrivo di una risposta.
La risposta viene generata dal J/LMS dell'LME dopo aver elaborato i dati contenuti nel JoinRequestMessage e può essere o un AckMessage o un NackMessage.
Quando l' LME riceve un JoinRequestMessage crea un JLMSServiceThread al cui costruttore passa il messaggio, l'hashtable dei profili di gruppo e l'hashtable dei vincoli.
A questo punto il thread estrae il profilo dal messaggio ed effettua in profile matching. Se dà esito positivo invoca il metodo generatePID del PENS per generare il PID del nuovo membro del gruppo, invoca il metodo addMember del VMS per aggiungere un membro alla vista, genera un AckMessage e lo invia all'entità mittente.
Se il profile matching non viene superato, il thread genera un NackMessage e lo invia al mittente.
Quando il JLMSJoinThread riceve un AckMessage, per prima cosa aggiorna le strutture dati del J/LMS e del VMS con i dati contenuti nel messaggio, poi invoca il metodo addMembership del PS, infine termina la sua esecuzione.
Se riceve un NackMessage, termina la sua esecuzione.

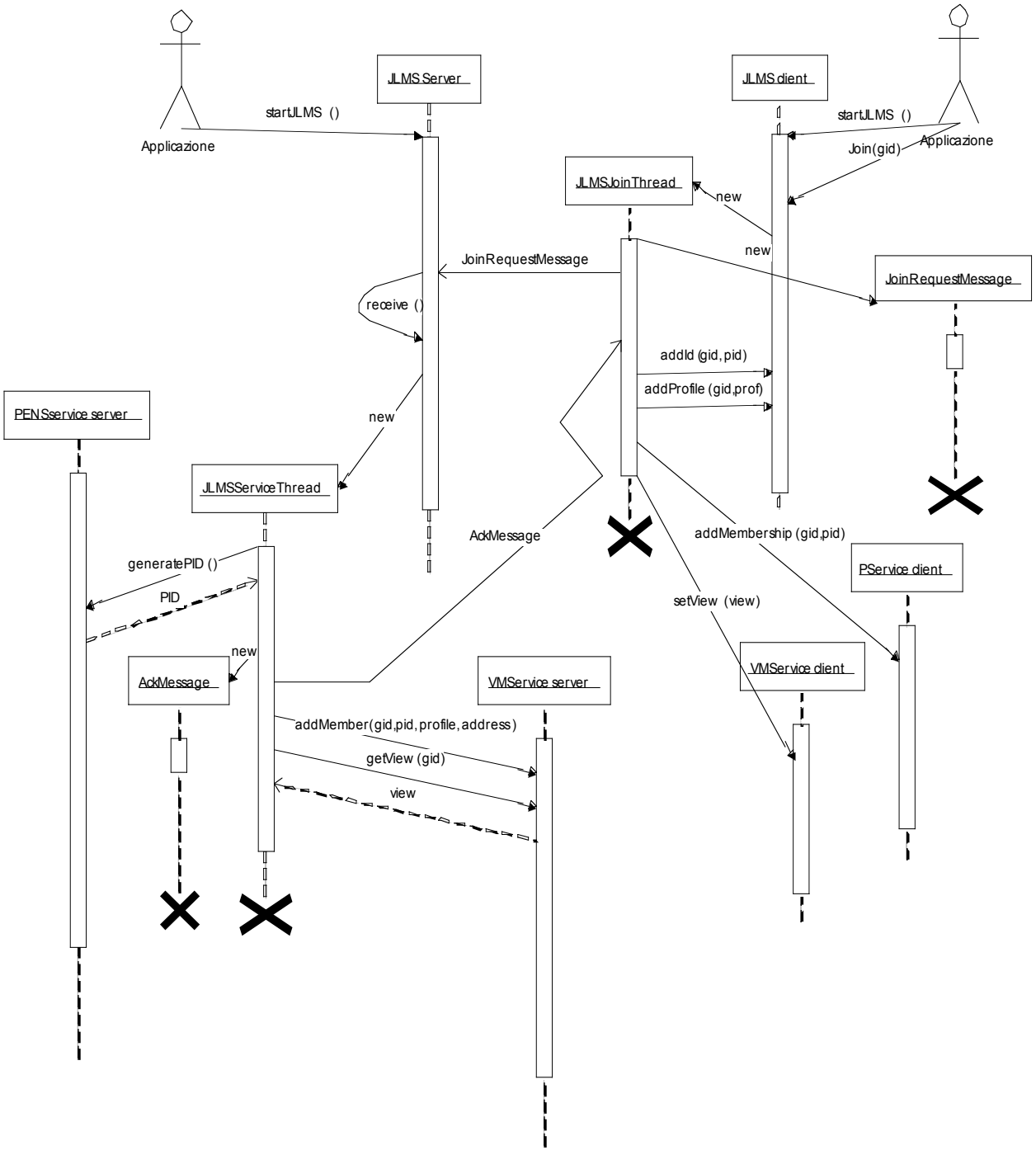

fig. 3.7 - Join ad un gruppo: diagramma di sequenza

3.5 Leave da un gruppo

Il leave da un gruppo non richiede interazioni con gli LME che gestiscono il gruppo. Il meccanismo del leave si basa sul funzionamento del PS e del PENS.
Quando viene invocato il metodo leave di un LME bisogna passare come parametro il GID del gruppo da cui uscire.
Il metodo in base al GID elimina i dati memorizzati nelle hashtable, e invoca il metodo delMembership del PS in modo da non trasmettere più il beacon relativo a quel gruppo. A questo punto chiede al VMS di eliminare la vista di quel gruppo.
I VMS che girano sugli LME della località si accorgeranno che nelle tabelle del PENS non c'è più un elemento e aggiorneranno la vista del gruppo.

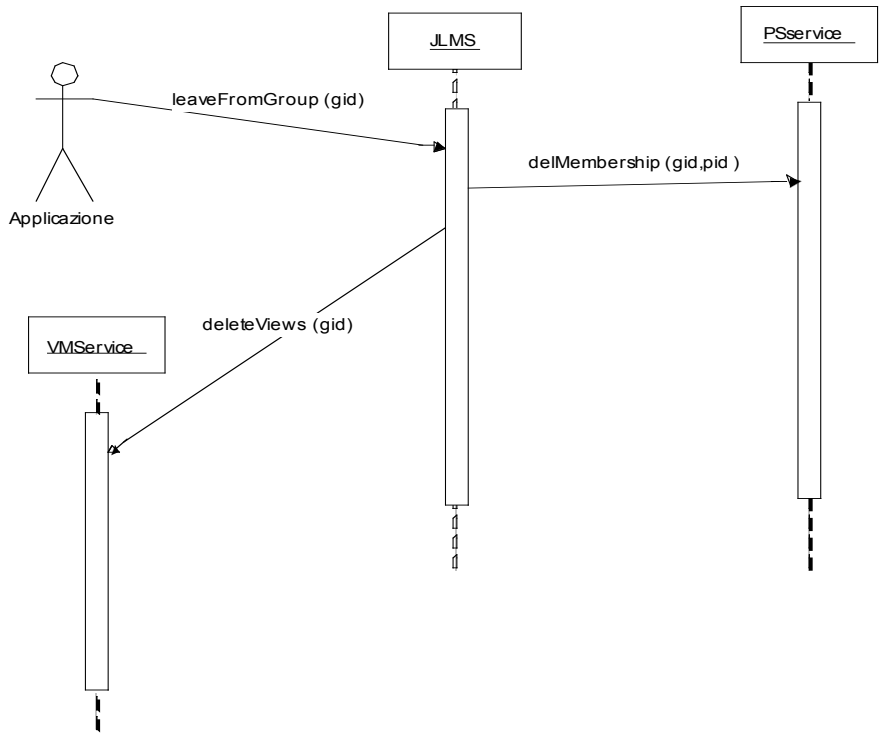

fig. 3.8 - Leave da un gruppo di un LME: diagramma di sequenza

Gli ME invece possono effettuare il leave solo dall'actual group, in quanto fanno parte di un gruppo per volta. Quando viene invocato il metodo leave di un ME, bisogna fornire come parametro il GID del gruppo a cui passare. Se tale gid non è presente nell'hashtable degli identificatori, non si passa ad alcun gruppo.

3.6 Discover dei gruppi

Questa operazione serve ad ottenere i profili dei gruppi presenti nella località.
Si può fare la discover di tutti i gruppi o solo di quelli il cui profilo combacia con quello dell'entità che esegue l'operazione.
L' ultimo caso può essere svolto in due modi:

- Richiedendo il profilo e memorizzandolo solo se supera il profile matching
- Inviando il profilo e memorizzando la risposta solo in caso di profile matching positivo.

Nel primo caso il profile matching viene effettuato sull' entità che fa la discover, nel secondo caso sull' LME che riceve la richiesta.
I messaggi in tutti i casi vengono inviati ad un LME per gruppo, il quale risponderà in base al messaggio ricevuto.
In caso della discover di tutti i gruppi, dal lato client viene generato il JLMSDiscoverThread che crea ed invia i JLMSDiscoverRequest ad un LME per gruppo e aspetta i JLMSDiscoverResponse. Quando arriva una response la memorizza nel JLMService invocando il metodo addD. La discover di tutti i gruppi è mostrata in figura 3.7.
Nel caso della discover dei gruppi il cui profilo combacia con il proprio, viene creato il thread JLMSDiscoverMatchingGroupsThread che agisce allo stesso modo del JLMSDiscoverThread ma memorizza solo i JLMSDiscoverResponse che contengono un profilo che rispetta i vincoli.

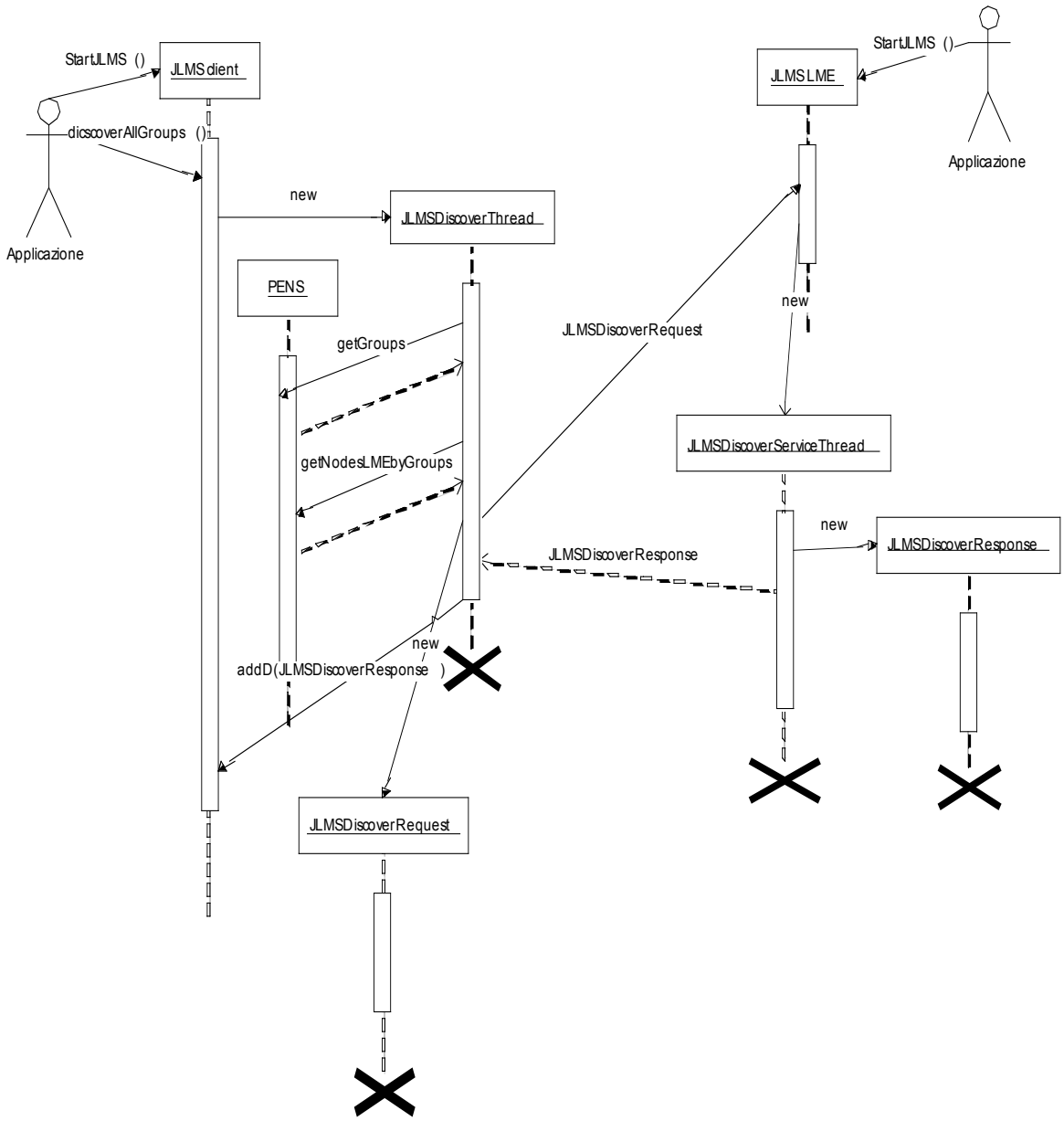

fig. 3.9 - Discover dei gruppi: diagramma di sequenza

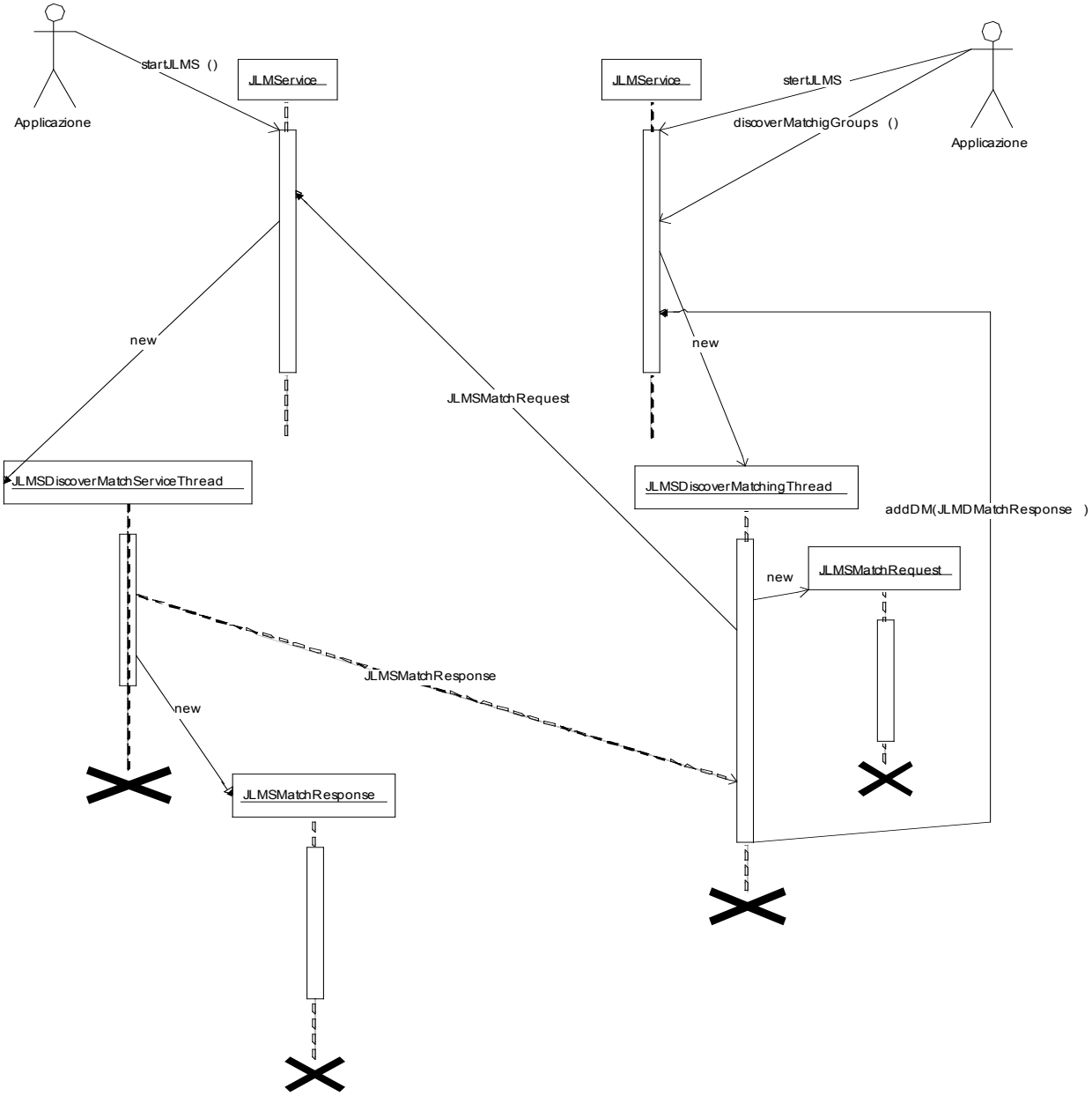

fig. 3.10 - Discover dei gruppi: diagramma di sequenza

Nel caso in cui il profile matching venga eseguito sull'LME che riceve la richiesta, il client invia una JLMSMatchRequest per mezzo del JLMSDiscoverMatchingThread.
Tale thread non fa altro che inviare i JLMSMatchRequest e attendere i JLMSMatchResponse.
Quando riceve un JLMSMatchResponse controlla se il campo di tipo booleano contenuto nel messaggio ha valore true o false.

Se ha valore true memorizza il messaggio invocando il metodo addDM della classe JLMService, altrimenti passa ad interrogare l'LME di un' altro gruppo.
Quando il JLMService riceve un JLMSMatchRequest crea il JLMSMatchServiceThread che effettua il profile matching e invia il JLMSMatchResponse.

CAPITOLO 4
VIEW MANAGER SERVICE (VMS)

Il View Manager Service (VMS) è il servizio di AGAPE che si occupa di creare, gestire e propagare a livello applicativo la lista di tutti i membro di uno stesso gruppo che sono correntemente nella stessa località, i loro attributi e caratteristiche. Chiamiamo questa lista context dependent view.
Le context dependent view o context view, sono delle tabelle in cui ogni entry contiene le seguenti informazioni: il PID, l'indirizzo IP e il profilo di ogni membro presente nella località di un certo gruppo.
Ogni vista è identificata dal GID del gruppo a cui appartengono le entry che la compongono.

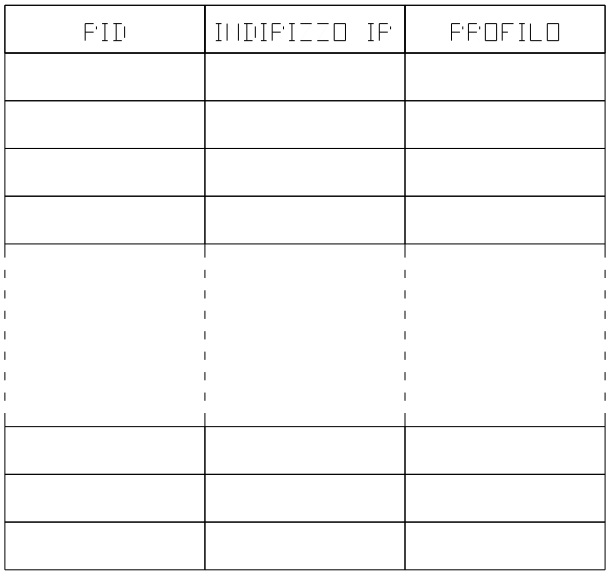

fig. 4.1 – Esempio di vista

In pratica si può immaginare una vista come una tabella contenente le informazioni riguardanti i membri di un gruppo.
Le context view vengono aggiornate e inviate periodicamente a tutti i membri contenuti nella vista stessa.
Ogni entità memorizza le viste dei gruppi di cui fa parte. Ogni volta che viene ricevuta una nuova context view tali viste vengono aggiornate inserendo i membri che non erano presenti.
AGAPE riconosce le caratteristiche eterogenee dei dispositivi quindi cerca di ottimizzare le prestazioni del sistema assegnando ai dispositivi con migliori prestazioni in termini di durata delle batterie e di capacità di calcolo i compiti di gestione.
Per questo motivo sono state sviluppate due versioni del VMS: una per gli LME e una per gli ME.

La versione sviluppata per le managed entities è più leggera rispetto a quella sviluppata per gli LME. Infatti solo gli LME gestiscono le context view.
In particolare solo un LME può creare una nuova vista e gestisce contemporaneamente le viste di tutti i gruppi di cui fa parte. Inoltre solo gli LME inviano le context view ai membri del gruppo presenti nella località.
Gli ME, potendo far parte attivamente di un gruppo per volta, ricevono solo le viste di quel gruppo.
Per gestire le context dependent view, il VMS nella versione LME deve monitorare i membri dei gruppi per controllare se i dati presenti nelle context view corrispondono alla realtà. Il VMS controlla periodicamente il PENS il modo da confrontare i dati contenuti nelle sue tabelle con quelli delle context view, in modo da aggiornare tali viste aggiungendo o eliminando dei membri.
Il VMS dissemina le viste, inviandole ad intervalli regolari a tutti i membri della località sfruttando un protocollo di gossipping.
Il controllo del PENS risulta utile anche per svolgere la funzione di roaming, in cui vanno gestiti gli ingressi e le uscite di membri dalla località.
Con il termine roaming si indica il problema delle stazioni mobili che migrano da una località ad un'altra.
L'uscita e l'ingresso delle stazioni da una località deve essere rilevata e gestita aggiornando le context view relative alla stazione.
Il rilevamento viene effettuato controllando periodicamente il PENS.
Il PENS tiene una tabella aggiornata di tutte le entità presenti nella località, quindi controllando periodicamente tale tabella il VMS si accorge se un'entità non è più visibile all'interno della località e quindi la elimina dalla context view.
Può accadere anche il comportamento inverso: una stazione facente parte di un gruppo gestito da un LME della località entra in tale località.
L'LME interrogando il PENS si accorge della presenza di una nuova entità che però non è nella context view.
Per inserirla nella vista non bastano i dati contenuti nel PENS, ma serve anche il profilo dell'entità.
Per ottenere il profilo, il VMS invia un VMSMessageProfileRequest all'entità.
A questo punto l'entità risponde all'LME inviando il proprio profilo.
Una volta ricevuto il profilo l'LME aggiorna la context view.

Per poter gestire le viste, il VMS si interfaccia con altri servizi di AGAPE: il J/LMS, il PENS, il VCS e l'NMS.
In particolare il J/LMS invoca i metodi per creare una nuova vista e per aggiungere un nuovo elemento ad una vista. Il PENS, invece, viene interrogato dal VMS per scoprire le entità presenti nella località. Il VCS autorizza l'invio della context view. Per l'invio e la ricezione delle context view vengono usate le primitive di comunicazione del NMS.
Essendo la gestione dei gruppi e la gestione delle viste strettamente correlate loro lo sono anche il J/LMS e il VMS.
In particolare tutte le operazioni effettuate dal J/LMS che comportano l'aggiunta o la rimozione di un membro da un gruppo si riflettono inevitabilmente sulle viste di quel gruppo.
Tutte le volte che viene creato un nuovo gruppo o un'entità esegue il join o il leave da un gruppo il VMS deve creare una nuova vista o aggiornare la vista già esistente.
Quando un LME crea un nuovo gruppo invoca il metodo createNewGroup del J/LMS il quale, dopo aver creato il nuovo gruppo, invoca il metodo inizializeView del VMS passando come parametri il GID del gruppo appena creato, il proprio PID e il proprio profilo.
In questo modo il VMS crea una nuova vista associata al nuovo gruppo e la inizializza inserendo come primo elemento l'LME che ha creato il gruppo.

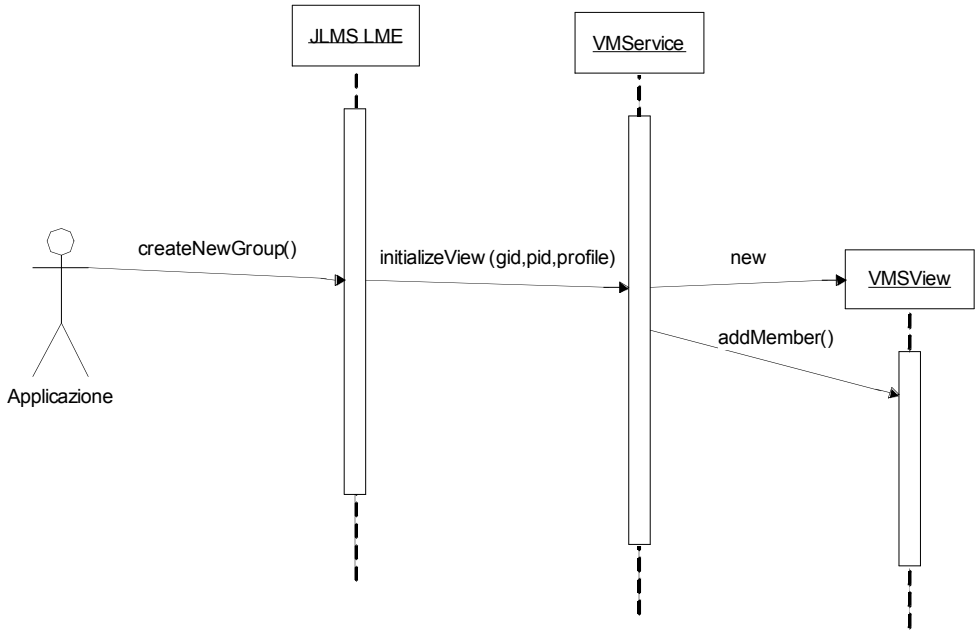

fig. 4.2 – J/LMS e VMS creazione di un gruppo

Quando un LME esegue il join ad un gruppo inizializza una nuova vista invocando il metodo del VMS initializaView e poi invoca il ReceiveViewThread che aggiunge alla nuova vista gli elementi della vista ricevuta con l'Acknoledge. Il thread inserisce solo gli elementi della località.
L'LME che ha ricevuto la richiesta di join non fa altro che aggiungere il nuovo membro del gruppo alla vista.

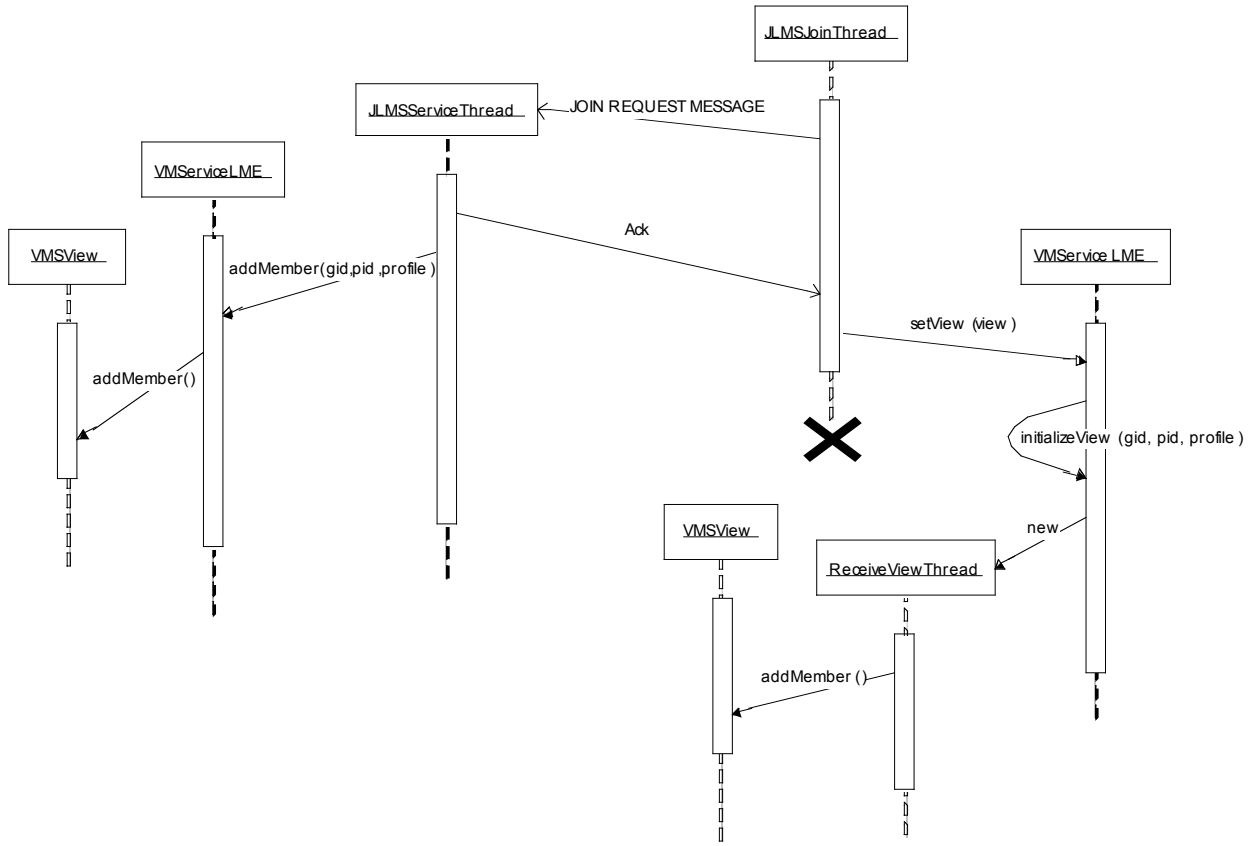

fig. 4.3 – J/LMS e VMS: join ad un gruppo

Per quanto riguarda l'operazione di leave da un gruppo il comportamento di un LME è diverso da quello di un ME.
Quando un LME lascia un gruppo non fa altro che invocare il metodo leaveFromGroup del J/LMS specificando come parametro il GID del gruppo da lasciare. Il J/LMS invoca il metodo deleteViews del VMS specificando il GID del gruppo.
Tale metodo cancella tutte le viste ricevute del gruppo GID e la vista corrispondente dal vettore delle viste gestite.

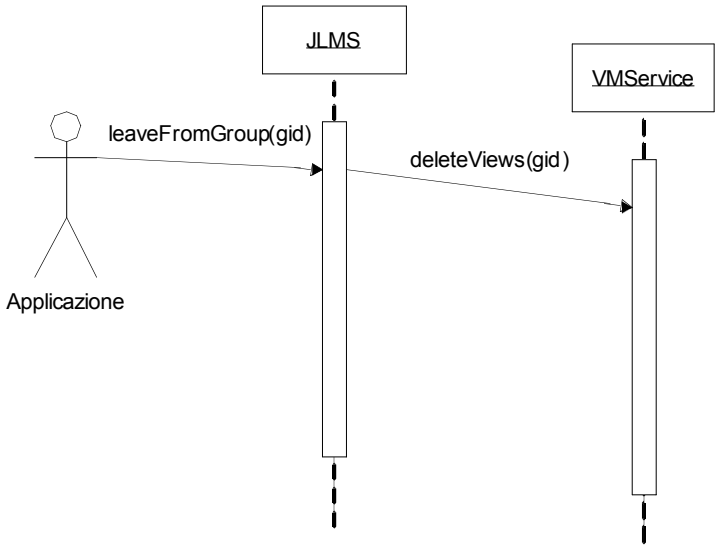

fig. 4.4 – J/LMS e VMS: leave da un gruppo di un LME

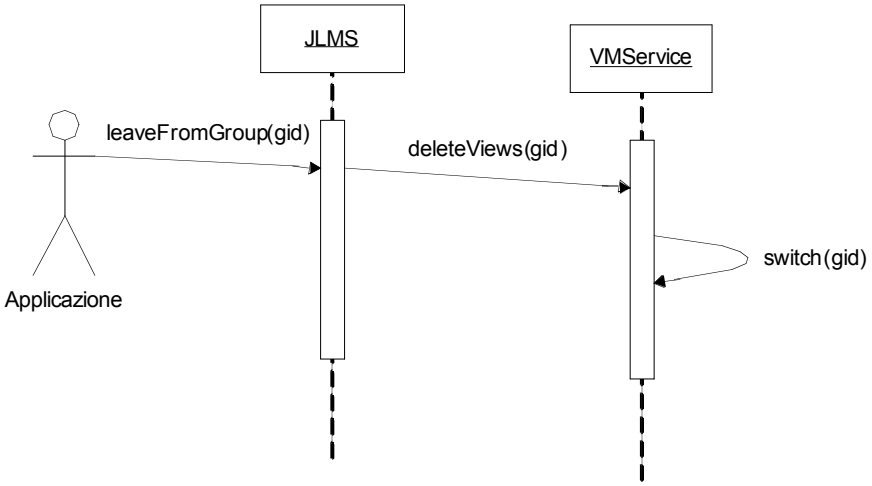

fig. 4.5 – J/LMS e VMS: leave da un gruppo di un ME

Quando un ME esegue il leave da un gruppo invoca il metodo leaveFromGroup del J/LMS specificando il GID del gruppo a cui passare una volta usciti da quello attuale.
Una volta eseguito il leave dal gruppo il J/LMS invoca il metodo deleteViews del VMS passando come parametro il GID del gruppo a cui fare lo switch.

4.1 Interrogazioni

Affinché le applicazioni e gli altri servizi possano interrogare il VMS sono stati sviluppati dei metodi che forniscono un interfaccia con l'esterno.
I metodi sviluppati permettono di ottenere molte informazioni riguardanti le viste e lo stato dell'LME.
Le differenze esistenti tra la versione LME e la versione ME del VMS si riflettono anche sull'interfaccia del servizio.
Infatti le due versioni differiscono tra loro per la presenza o meno di alcuni metodi in base alle differenze tra le due versioni.
Il metodo getBatteryLevel è presente solo negli LME e permette al VCS di autorizzare l'invio della context view leggendo il livello batteria.
Sono stati sviluppati due metodi che permettono di interrogare una delle viste per ottenere uno dei membri o un insieme di membri.
Il primo è getMember che viene invocato passando come parametri il GID che identifica la vista da interrogare e il PID del membro ricercato. Negli ME il GID non serve perché viene gestita una vista per volta.
Il secondo è getMatchingMembers che accetta come parametro un vector contenente una serie di vincoli e il GID che identifica la vista. Tale metodo esegue il profile matching in base ai vincoli passati del profilo d'ogni membro. Restituisce un vector contenente tutti i membri che hanno superato il profile matching. Negli ME non serve passare il GID.
I metodi getView(gid) e getViews() permettono di ottenere la vista identificata dal GID passato oppure tutte le viste.
Sono stati sviluppati dei metodi specifici per gli ME: getActualGid e getActualView.
Il primo restituisce il GID della vista corrente, il secondo restituisce la vista corrente.

4.2 Dettagli implementativi

Il servizio è stato sviluppato utilizzando il pattern singleton. La classe VMService è la classe principale del servizio su cui vengono invocati, da parte di applicazioni e degli altri servizi AGAPE, i metodi che implementano le funzionalità del servizio.
Tale classe è realizzata estendendo la classe ActiveComponent la quale estende AgapeComponent.

Gli ActiveComponent sono degli AgapeComponent attivi che hanno un thread che esegue il task implementato nel metodo task().
L'ActiveComponent viene attivato invocando il metodo start() e deattivato invocando il metodo stop(). Per controllare se l'ActiveComponent è attivo è stato implementato il metodo IsActive().

4.2.1 LME

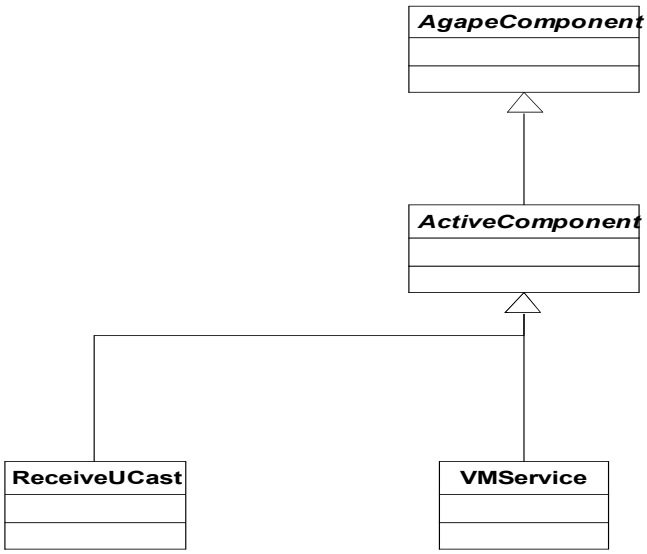

fig. 4.6 – Tassonomia del VMS

Oltre alla classe VMService è sempre attivo anche un altro ActiveComponent: ReceiveUCast.
Tale classe ha il compito di leggere continuamente sulla porta unicast per rilevare l'arrivo di viste, di richieste di profilo e di profili.
Tra le responsabilità degli LME compare anche il roaming. Per questo motivo la classe ReceiveUCast si occupa di gestire l'arrivo di richieste di profilo e l'arrivo di profili. Il roaming verrà descritto nel paragrafo 4.6.

4.2.1.1 Diagramma statico delle classi

La classe principale del servizio è VMService. E' la classe su cui vengono invocati i metodi che implementano le funzionalità del servizio.
Alcune di tali responsabilità vengono delegate ad altre classi. La classe ReceiveUCast è un ActiveComponent che viene attivato insieme al VMS e rimane in ascolto sulla porta unicast.

La classe ControlPensThread è un thread che viene creato dal VMS quando scopre che un membro presente nella vista non è presente tra le entry del PENS.
Questa situazione si può verificare in tre casi:

- Un'entità ha eseguito il leave dal gruppo;
- Un'entità è uscita dalla località;
- Un'entità ha appena fatto il join al gruppo.

Nei primi due casi l'entità và effettivamente cancellata dalla vista perché o non fa più parte del gruppo o non è più visibile nella località, mentre nell'ultimo caso non và cancellata perché si è appena unita al gruppo.
Quando un'entità effettua il join ad un gruppo si verifica una discrepanza tra i membri contenuti nella vista e le entry del PENS: la vista viene aggiornata appena l'LME conferma la corretta esecuzione del join, mentre il PENS viene aggiornato con qualche secondo di ritardo perché non riceve immediatamente i beacon con la membership del nuovo membro del gruppo.
Per questo motivo appena il ControlPensThread viene attivato, attende qualche secondo e verifica nuovamente se l'entità non è presente nel PENS. Solo se anche il secondo controllo dà esito positivo il membro viene effettivamente cancellato dalla vista.
La classe ManageViewThred è un thread che viene attivato da ReceiveUCast quando riceve una vista gestita dall'LME.
Il ManageViewThread aggiorna la lista delle viste ricevute e crea un nuovo oggetto ReceiveViewThread.

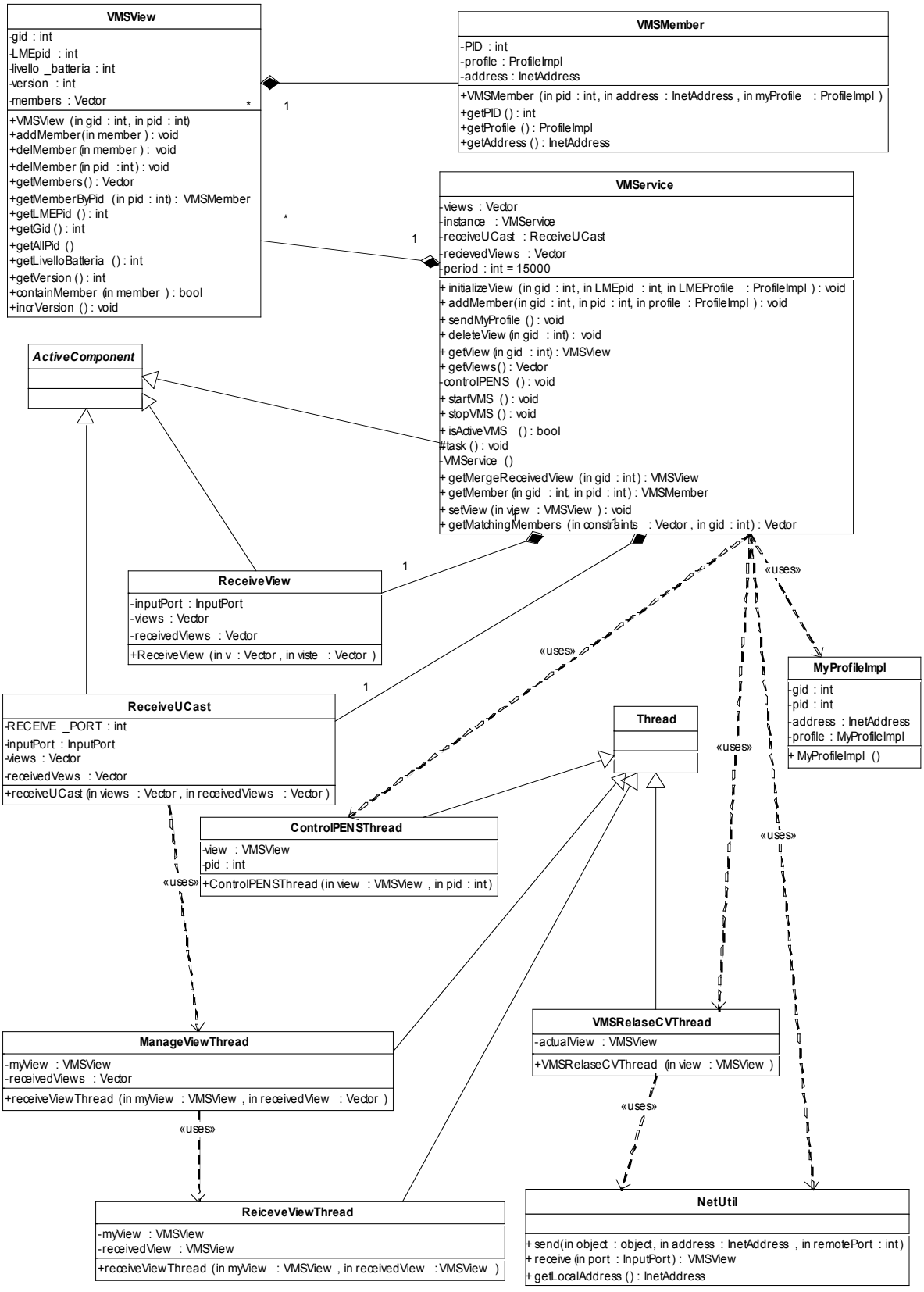

fig. 4.7 – LME: Diagramma statico delle classi

Il ReceiveViewThread si occupa di inserire nella vista i membri contenuti nella vista ricevuta che appartengono alla propria località.
I ReceiveViewThread vengono creati anche dal VMS quando un LME esegue il join ad un gruppo.
Nell'ultimo caso quando un LME fa il join ad un gruppo, riceve un messaggio di conferma che contiene anche la context view del gruppo a cui si è unito.
Appena riceve l'acknoledge l'LME crea una nuova vista in cui inserisce se stesso come primo elemento e successivamente tutti i membri della vista ricevuta con l'acknoledge.
Le classi VMSView e VMSMember astraggono rispettivamente i concetti di vista e di membro di una vista.
Un membro di una vista rappresenta un'entità presente nella località che fa parte del gruppo descritto dalla vista.
Per ogni membro vengono memorizzati il PID, il profilo e l'indirizzo IP.
Ogni vista ha un numero di versione che serve al VMS per memorizzare solo le viste più recenti, ha un campo che indica il livello della batteria dell'entità e un campo che indica il GID del gruppo.
Il campo livello batteria è utilizzato dal VCS per determinare quale LME della località deve trasmettere le context view.
La classe NetUtil viene usato per la comunicazione unicast.

4.2.2 ME

Come per la versione LME, anche per la versione ME è stato usato il pattern singleton.
La classe principale è VMService che eredita da ActiveComponent.
Gli ME gestiscono una vista per volta, semplicemente aggiornando la vista attuale con le context view che ricevono dagli LME della località.

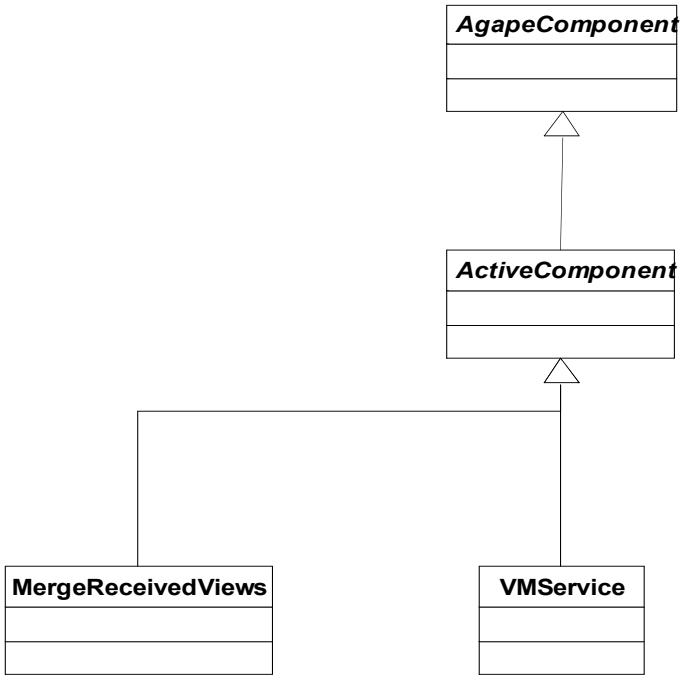

fig. 4.8 – Tassonomia del VMS

La classe MergeReceivedViews è un ActiveComponent che ogni quindici secondi effettua il merge delle viste ricevute, aggiornando la vista attuale.
Per il resto delle classi la descrizione è come quella dell'LME.

La gestione delle viste da parte di un ME avviene nel seguente modo: quando un ME riceve una vista del proprio gruppo la inserisce nel vettore delle viste ricevute, ogni quindici secondi interviene il thread MergeReceivedViews.
L'operazione di merge consiste nell'unire in un'unica vista tutte le informazioni contenute nelle viste ricevute evitando di inserire elementi ripetuti.

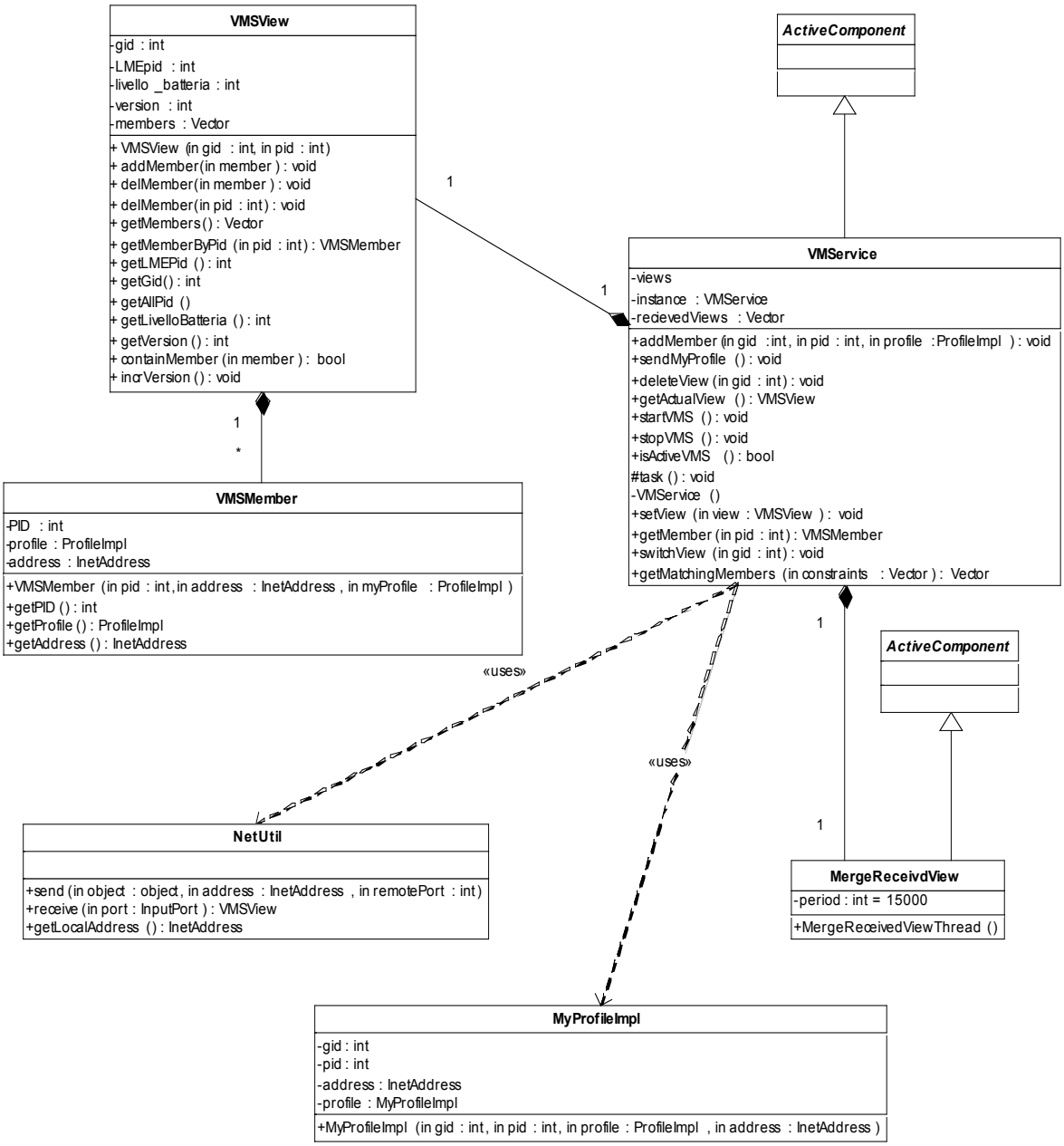

fig. 4.9 – VMS ME: diagramma statico delle classi

4.3 Invio e ricezione delle viste

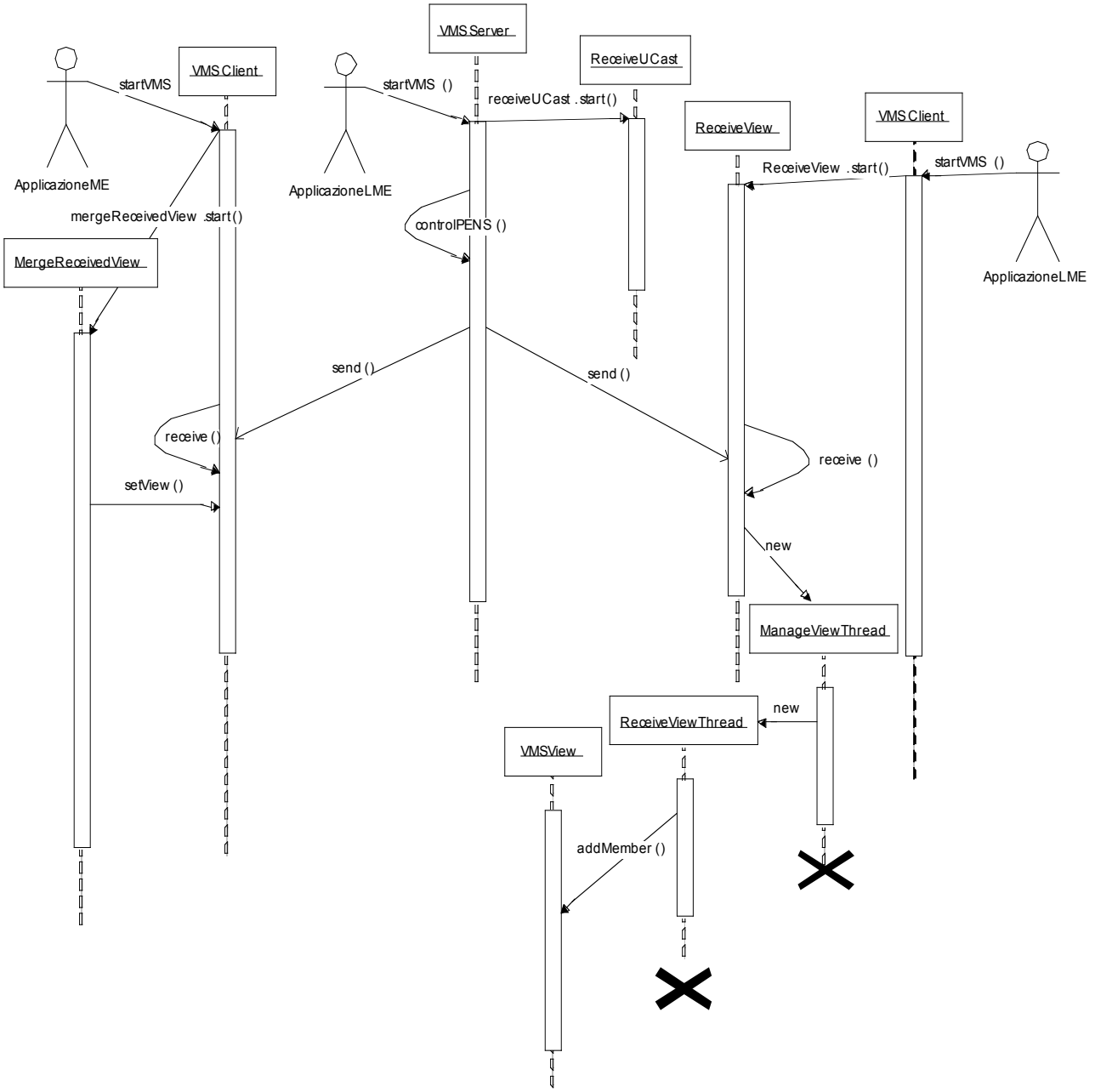

fig. 4.10 – Invio e ricezione delle viste: diagramma di sequenza

La diffusione delle viste viene effettuata periodicamente in broadcast dall'LME che è stato autorizzato dal VCS.
Gli LME ricevono le viste grazie al ReceiveView: un thread che è sempre in ascolto sulla porta di ricezione.
Una volta ricevuta una vista, il thread ReceiveView invoca ManageViewThread che controlla il numero di versione della vista e se è maggiore di quelli già memorizzati la inserisce nel vettore delle viste ricevute. A questo punto viene invocato il ReceiveViewThread il quale esegue un aggiornamento della vista del gruppo inserendo i membri

presenti nella context view appena ricevuta che non ci sono nella vista del gruppo.
Per quanto riguarda gli ME, invece, la ricezione della context view avviene in modo più semplice: la vista viene ricevuta direttamente dal VMS il quale non fa altro che memorizzare nel vettore delle viste ricevute quelle con il numero di versione maggiore.
Periodicamente interviene il MergeReceivedView, un thread sempre attivo, che esegue la fusione delle viste ricevute in un'unica vista.
La fusione avviene aggiungendo alla vista attuale i membri delle viste ricevute. Al termine della fusione viene aggiornata la vista attuale invocando il metodo setView e vengono cancellate le viste ricevute.

CAPITOLO 5
CASO DI STUDIO

Al fine di valutare l'applicabilità di AGAPE, supporto di applicazioni collaborative avanzate, abbiamo sviluppato il prototipo di un'applicazione nell'ambito della protezione civile che permette a operatori co-locati di collaborare in caso di una grave calamità senza la presenza di un'infrastruttura pianificata.
L'applicazione sfruttando il sistema AGAPE permette il coordinamento fra diversi operatori della protezione civile e consente loro di avere visibilità delle entità vicine e comunicare attraverso scambio di messaggi.
Il sistema AGAPE si adatta perfettamente alle esigenze dell'applicazione in quanto i differenti utenti del sistema hanno caratteristiche diverse in base al ruolo svolto all'interno della protezione civile, quindi sono caratterizzati da profili diversi, inoltre nel caso preso in esame non sono disponibili altre reti di comunicazione (GPRS, UMTS, Tetra) e gli utenti hanno la possibilità di interagire per mezzo di una MANET.
In particolare AGAPE è utile allo scopo dell'applicazione perché consente agli utenti di individuare tutti i vicini e tutti i gruppi della località e mantiene aggiornati tali dati per mezzo della diffusione delle context view e dei beacon del PS: l'applicazione può interrogare AGAPE per ottenere dati riguardanti i vicini utilizzando come parametri sia i profili di gruppo sia i profili utente.
Ogni gruppo creato è caratterizzato da un profilo che identifica le caratteristiche comuni a tutti gli appartenenti al gruppo.
Il profilo utente ha, oltre alle caratteristiche comini al gruppo cui appartiene, delle caratteristiche proprie.
I profili usati nell'applicazione sono composti da tre attributi tutti di tipo stringa: nome, cognome e professione. Quando viene creato il profilo di gruppo, viene inizializzato solo l'attributo professione con il valore che identifica il gruppo: ad esempio se il gruppo creato è quello dei vigili del fuoco tale attributo varrà Pompieri.
Inoltre il sistema AGAPE permette la creazione di gruppi, di unirsi ad un gruppo esistente, di cercare i gruppi presenti nella località e di rendere disponibili le informazioni relative ai gruppi a tutte le altre entità presenti nella località.
I test sono stati eseguiti su una rete MANET implementata utilizzando la rete IEEE802.11b in modalità ad hoc in cui gli indirizzi IP sono stati configurati staticamente.
I dispositivi usati per i test sono computer portatili che avevano il ruolo di LME e palmari con il ruolo di ME.

I laptop erano dotati di una scheda wireless Cisco e di sistema operativo Linux Fedora o Windows XP con JDK 1.4, mentre i palmari avevano il sistema operativo Windows CE con J2ME.

5.1 Creazione di un gruppo

Quando si verifica un'emergenza gli operatori della protezione civile che si trovano sul posto hanno la necessità di creare dei gruppi a cui unirsi.
Nel caso dell'applicazione sviluppata in questa tesi ogni gruppo rappresenta un tipo di operatore della protezione civile: ad esempio c'è il gruppo dei Pompieri, quello dei Paramedici, ecc.
Per creare un gruppo è necessario definire due classi di servizio che permettono di creare i profili di gruppo, i profili delle singole entità e i vincoli che verranno utilizzati per il profile matching.
Queste due classi sono la ProfileFactory e la ConstraintFactory.
La ProfileFactory permette di creare correttamente i profili sia per le singole entità sia per i gruppi, mentre la ConstraintFactory permette di creare l'insieme dei vincoli.

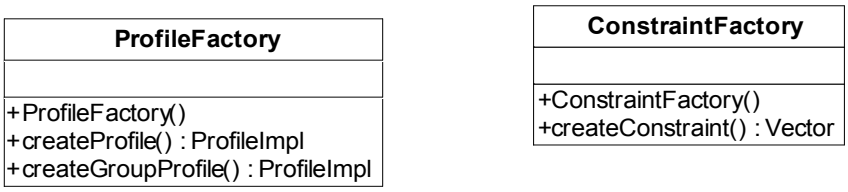

fig. 4.1 – Classi ProfileFactory e ConstraintFactory

Prima di invocare il metodo create() del J/LMS per creare un gruppo, vanno creati un profilo di gruppo e i vincoli usando le classi sopradescritte. Segue un esempio di codice per la creazione di un gruppo.

```
ProfileFactory pf=new ProfileFactory();
ProfileImpl groupProfile=pf.createGroupProfile();
Vector v=ConstraintFactory.createConstraint();
J/LMService.CreateNewGroup(groupProfile,v);
```

Durante la creazione dell'insieme dei vincoli vengono stabilite le regole usate per accettare un nuovo membro all'interno del gruppo: gli attributi nome e cognome possono non contenere alcun valore, mentre l'attributo professione deve essere presente e deve contenere un valore specifico (ad

esempio considerando il gruppo dei vigili del fuoco deve valere Pompiere).

5.2 Join ad un gruppo

Il join ad un gruppo viene effettuato da un operatore che arriva sul luogo del disastro e trova già un gruppo che rispecchia le sue caratteristiche.
Quando un'entità decide di entrare a far parte di un gruppo a livello applicativo deve utilizzare il metodo Join(int gid) della classe JLMService.
Quando viene invocato tale metodo il J/LMS invia un messaggio ad un LME che appartiene al gruppo contenente il suo profilo.
L'LME effettua il profile matching controllando che l'attributo professione sia presente ed abbia lo stesso valore contenuto nel profilo di gruppo.
Se il profile matching va a buon fine allora l'LME invia un messaggio all'entità che contiene il GID, il PID e la context view. A questo punto l'entità entra a far parte del gruppo.
Per facilitare la gestione di diversi gruppi, è stato implementato un metodo che, dato il valore del campo professione del profilo trova il gid del gruppo.

```
int gid=trovaGid(out,gruppo);
JLMService.Join(gid));
```

Il metodo trovaGid per trovare il gid dato il campo per prima cosa estrae dal vettore dei gruppi scoperti i profili di gruppo:

```
for(int i=0;i<disGr.size();i++)
{
        response=(JLMSDiscoverResponse)disGr.get(i);
        v.add(response.getProfile());
}
```

Una volta estratti i profili cerca il primo gruppo che ha il campo professione contenente il valore specificato e ne memorizza il profilo:

```
for(int j=0;j<v.size();j++)
{
        p=(ProfileImpl)v.get(j);
        Set s=p.getComponents();
        Iterator i=s.iterator();
        while(i.hasNext())
        {
```

```
                    c=(Component)i.next();
                    if(c!=null)
                        a=c.getAttribute("professione");
                        if(a!=null)
                        {
                            trovato=true;
                        }
                }
                if(trovato)
                    break;
        }
```

A questo punto ripete la scansione del vettore dei gruppi scoperti cercando il messaggio con il profilo trovato in precedenza. Una volta trovato ne estrae il gid:

```
for(int j=0;j<v.size();j++)
    if(p.equals((ProfileImpl)v.get(j)))
    {
        JLMSDiscoverResponse
        response=(JLMSDiscoverResponse)disGr.get(j);
        gid=response.getGid();
        break;
    }
```

5.3 Leave da un gruppo

Il leave da un gruppo viene effettuato nel seguente modo:

```
JLMService.leaveFromGroup(gid);
```

Per ottenere il gid del gruppo bisogna invocare il metodo:

```
int gid=trovaGid(out,gruppo);
```

dove gruppo è una stringa con il valore del campo professione del profilo.
Nel caso di un ME, invece, il leave può essere fatto solo dall'actualGroup. Quindi il GID da passare al metodo non è quello del gruppo da cui uscire, ma è quello del gruppo a cui passare una volta usciti dall'actualGroup.
Tale GID va scelto fra quelli presenti nelle hashtable del servizio, cioè fra quelli a cui l'entità aveva fatto il join precedentemente.
Per avere l'insieme di tutti i GID bisogna usare il metodo `getAllGid()`, che restituisce un'enumeration contenente i GID.

Se il GID passato come parametro non esiste nell'hashtable, non si passa ad alcun gruppo.

5.4 Discover dei profili di gruppo

L'operazione di discover dei profili di gruppo, permette ad un operatore di scoprire i profili di gruppo all'interno della sua località.
Per effettuare la discover dei profili di gruppo, sono a disposizione delle applicazioni, tre metodi:

- discoverAllGroups(): viene usato per scoprire i profili di tutti i gruppi presenti nella località;
- discoverMatchingGroups(constraints): viene usato per trovare tutti i gruppi che hanno il cui profilo supera il profile matching. Il profile matching viene effettuato in base al vettore dei vincoli passato come parametro al metodo. La sua esecuzione avviene sull'entità che genera le richieste;
- discoverMatchingGroups(): ha lo stesso scopo del precedente, ma il profile matching viene eseguito sull'LME che riceve la richiesta.

Nel primo e nell'ultimo caso i metodi vanno semplicemente invocati:

```
JLMService.discoverAllGroups();
```

oppure

```
JLMService.discoverMatchingGroups();
```

Nel secondo caso, invece, bisogna prima creare il vettore dei vincoli:

```
Vector v=ConstraintFactory.createConstraint();
JLMService.discoverMatchingGroups(v);
```

I metodi descritti in precedenza memorizzano i messaggi ricevuti come risposta dagli LME interrogati, in due vector.
Utilizzando i metodi getDiscoveredGroups() e getDiscoveredMatchingGroups() l'applicazione può interrogare il J/LMS in modo da ottenere tutti i profili dei gruppi scoperti o solo quelli che superano il profile matching.

```
JLMSDiscoverResponse response=(JLMSDiscoverResponse)disGr.get(i);
int gid=response.getGid();
ProfileImpl profile=response.getProfile();
```

Poiché l'altro vector può contenere due tipi diversi di messaggi, l'accesso ai messaggi è leggermente più complesso:

```
Vector disMGr=JLMService.getDiscoveredMatchingGroups();
for(int i=0;i<disMGr.size();i++)
{
	JLMSMessage msg=(JLMSMessage)disMGr.get(i);
	if(msg.getType().equals("DiscoverResponse"))
	{
		JLMSDiscoverResponse
		response=(JLMSDiscoverResponse)msg;
		int gid=response.getGid();
		ProfileImpl profile=response.getProfile();
	}
	else if(msg.getType().equals("matchRes"))
	{
		JLMSMatchResponse
		response=(JLMSMatchResponse)msg;
		boolean result=response.getMatchResult();
		int gid=result.getGid();
	}
}
```

5.5 Viste

Una vista viene creata al momento della creazione del gruppo: in questo caso la vista contiene un solo elemento che è l'entità che ha creato il gruppo.
Ogni qualvolta che un'entità esegue il join o il leave da un gruppo viene aggiornata la context view relativa al gruppo: per il join viene inserito un nuovo membro nella vista, in caso di leave viene eliminato un membro.
Supponiamo che un LME voglia creare il gruppo dei vigili del fuoco.
Una volta creato il profilo e i vincoli di gruppo crea la context view che avrà un aspetto simile a quella riportata in figura:

GID: -657890		
PID	Indirizzo IP	Profilo
12345677	10.0.0.1	Pompiere

Appena un'entità esegue il join al gruppo deve essere inserita all'interno della context view:

GID: -657890		
PID	Indirizzo IP	Profilo
12345677	10.0.0.1	Pompiere
-3456987690	10.0.0.3	Pompiere

L'uscita di un membro dal gruppo non viene rilevata immediatamente ma con un certo ritardo: ciò accade perché, a differenza del join che prevede uno scambio di messaggi tra le entità di tipo richiesta/acknoledge, per il leave non c'è scambio di messaggi tra le entità, quindi l'uscita di un membro viene rilevata interrogando il PENS e confrontando i risultati con il contenuto della context view.
Il PENS non viene aggiornato istantaneamente ma dopo qualche secondo di conseguenza anche la vista si viene aggiornata con ritardo.
Supponiamo che l'entità che ha creato il gruppo decida di lasciarlo.
Si possono verificare due casi: se l'altro membro è un LME esso inizierà a gestire il gruppo all'interno della località accettando le richieste di join o di discover e diffondendo le context view, oppure se non ci sono altri LME all'interno del gruppo esso viene sciolto in quanto non ci sono entità in grado di gestirlo.
La figura riporta il primo caso.

GID: -657890		
PID	Indirizzo IP	Profilo
-3456987690	10.0.0.3	Pompiere

5.6 Roaming

Oltre nei casi esposti nel paragrafo precedente le viste vanno aggiornate anche nel caso in cui un'entità lascia la località o entra al suo interno.
Supponiamo di avere un gruppo formato da tre membri:

GID: 345219		
PID	Indirizzo IP	Profilo
-12345543	10.0.0.2	Paramedico
345632456	10.0.0.5	Paramedico
45776783	10.0.0.4	Paramedico

Quando un membro lascia la località, il VMS si accorge che nella context view c'è un entità appartenente al gruppo che non è presente nella tabella del PENS. Dopo eseguito nuovamente il controllo ed aver ottenuto lo stesso risultato il membro viene eliminato dalla context view.

GID: 345219		
PID	Indirizzo IP	Profilo
-12345543	10.0.0.2	Paramedico
45776783	10.0.0.4	Paramedico

Nel caso in cui un membro appartenente al gruppo entra nella località non basta interrogare il PENS per avere tutti i dati del membro: dal PENS il VMS si accorge della presenza di un membro che non c'è nella context view, estrae il suo IP ed invia una richiesta di profilo al membro che risponde al LME inviando il suo profilo.
Una volta ottenuto il profilo l'LME può aggiungere il membro all'interno della context view:

GID: 345219		
PID	Indirizzo IP	Profilo
-12345543	10.0.0.2	Paramedico
345632456	10.0.0.5	Paramedico
-7654789	10.0.0.10	Paramedico

5.7 Invio e ricezione dei messaggi

L'invio e la ricezione dei messaggi viene realizzata sfruttando le primitive del NMS.
Ogni entità crea una porta di input e una di output: una volta scelto il gruppo tra quelli disponibili a cui inviare il messaggio si scrive il messaggio il quale viene inviato in unicast.
In ricezione c'è un thread sempre in ascolto sulla porta d'ingresso: quando viene ricevuto un messaggio viene stampato a video il suo testo e memorizzato il mittente in modo da poter rispondere.
Una volta scelto il gruppo l'entità a cui inviare il messaggio viene scelta automaticamente dall'applicazione.

Le operazioni per scegliere il membro a cui inviare il messaggio sono diverse nel caso in cui il gruppo scelto sia diverso dal proprio o meno.
Se il gruppo è diverso dal proprio l'unico modo per avere una lista dei membri di quel gruppo è interrogare direttamente il PENS, mentre se il gruppo è lo stesso basta interrogare il VMS richiedendo l'elenco dei membri del proprio gruppo che hanno le caratteristiche desiderate invocando il metodo getMatchingMembers nel seguente modo:

```
Vector v= VMService.getMatchingMembers(constr,gid);
```

dove constr è un Vector contenente i vincoli che esprimono i criteri di ricerca e gid è il gid del gruppo.
Nel caso ME l'interrogazione è leggermente diversa:

```
Vector v= VMService.getMatchingMembers(constr);
```

bisogna passare come parametro solo il Vector contenente i vincoli.

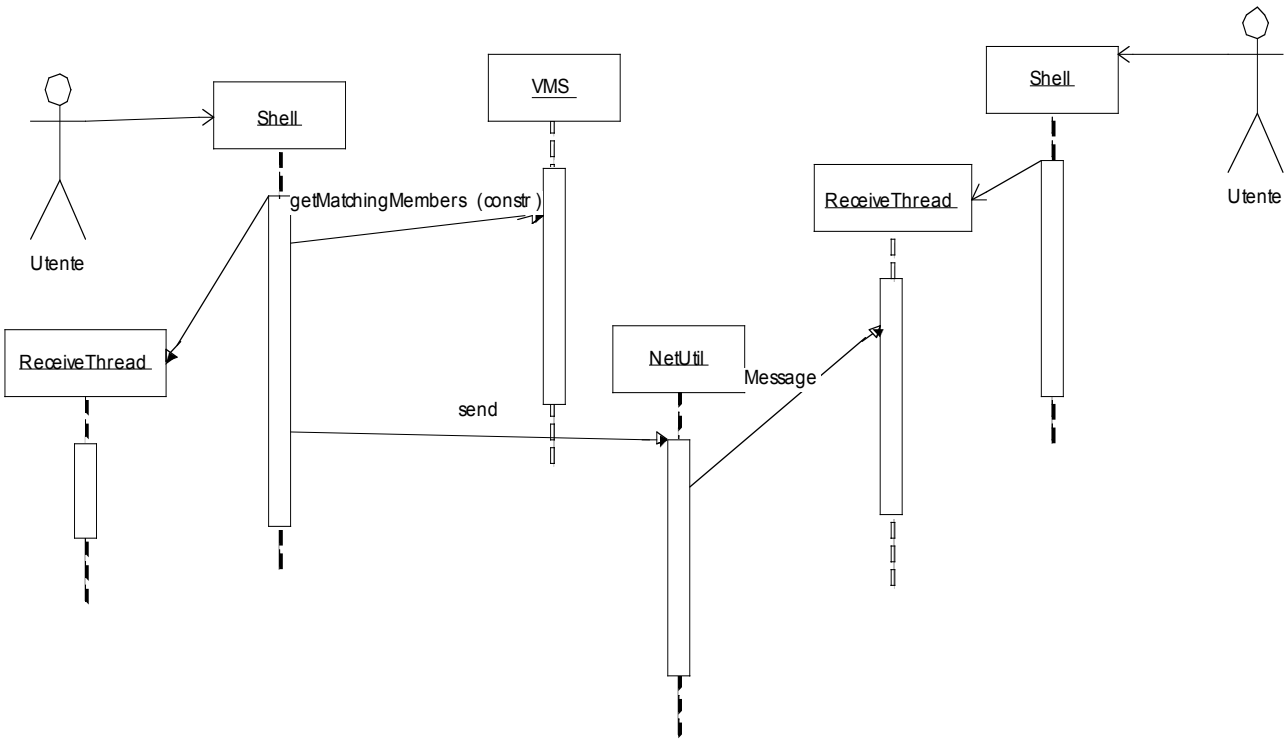

fig. 5.2 – Invio di un messaggio

Se, invece, si vogliono ottenere tutti i membri del gruppo bisogna usare il codice seguente:

```
VMSView vista=VMService.getView(gid);
Vector members=vista.getMembers();
```

dove gid è l'identificatore del gruppo descritto dalla vista.

5.8 Test

Essendo il problema della banda occupata, molto sentito in ambiente ad-hoc abbiamo valutato le dimensioni dei messaggi del J/LMS per controllare l'effettivo utilizzo della banda da parte di tale servizio.
Al fine di verificare l'utilizzo di memoria di AGAPE, le dimensioni di messaggi e viste sono state effettuate delle misurazioni sfruttando l'applicazione sviluppata.
Come prima cosa sono state misurate le dimensioni dei messaggi scambiati dal J/LMS per svolgere le sue funzioni:

TIPO MESSAGGIO	DIMENSIONE MEDIE(BYTE)
JoinRequestMessage	1650
AckMessage	2839
NackMessage	263
JLMSDiscoverRequest	292
JLMSDiscoverResponse	1654
JLMSMatchRequest	1654
JLMSMatchResponse	293

Molti dei dati riportati nella precedente tabella dipendono dalle dimensioni del profilo di gruppo o del profilo utente utilizzati nell'applicazione: in questo caso le dimensioni del profilo di gruppo sono 1339 byte e quelle del profilo utente sono 1351 byte.
All'interno dell'AckMessage è presente anche la vista aggiornata del gruppo, quindi le sue dimensioni varieranno al variare di quella della vista: nelle misurazioni effettuate la vista conteneva due membri e le sue dimensioni erano pari a 1902 byte, mentre ogni membro aveva dimensioni pari a 1569 byte.

Sono state effettuate anche delle misurazioni per verificare l'utilizzo di memoria di AGAPE sia a runtime sia in termini di memoria fisica occupata.
Le misurazioni fatte a runtime sono state compiute osservando le variazioni di dimensioni della java virtual machine.

Il sistema AGAPE con tutti i servizi attivi, senza alcun gruppo gestito e senza aver effettuato operazioni di discover dei gruppi occupa circa 3350KB. Tali dimensioni crescono all'aumentare dei gruppi gestiti o se si effettuano operazioni di discover dei gruppi, in quanto nel primo caso aumentano le viste da gestire, mente nel secondo caso vengono memorizzati dei dati relativi ai gruppi scoperti nella località e durante l'operazione di discover viene attivato un nuovo thread: ad esempio un LME che gestisce tre gruppi e che ha effettuato la discover di tutti i gruppi della località ha un occupazione di memoria di circa 3750KB (nella località sono presenti tre gruppi).
Le dimensioni nel caso ME non sono molto diverse dal caso LME: sono leggermente più piccole perché gli ME gestiscono le viste di un gruppo per volta e hanno meno thread attivi.
Per quanto riguarda l'occupazione di memoria fisica tutto il sistema AGAPE LME occupa 350KB, mentre la versione ME occupa 318KB.
Il J/LMS, invece, occupa nella versione LME 29,4KB e nella versione ME 27,7KB.
Le seguenti tabelle riportano i dettagli riguardanti lo spazio occupato dai vari servizi di AGAPE.

LME	
SERVIZIO	**SPAZIO OCCUPATO(KB)**
PS	4,89
PENS	7,37
NMS	84,5
VMS	26,4
J/LMS	29,4
Librerie di utilità	33,7

ME	
SERVIZIO	**SPAZIO OCCUPATO(KB)**
PS	4,89
PENS	7,37
NMS	84,5
VMS	14,0
J/LMS	27,7
Librerie di utilità	32,9

Conclusioni

Lo sviluppo di applicazioni collaborative in ambiente mobile ad-hoc network, solleva numerose problematiche riguardanti la gestione dei gruppi. In particolare le caratteristiche delle MANET e la mobilità dei terminali utente causano frequenti variazioni nella topologia della rete: connessioni, disconnessioni, partizioni e merge di rete sono eventi frequenti ed impredicibili che causano transitori nella collaborazione fra utenti nuovi e precedentemente sconosciuti.
In questa tesi è stato presentato AGAPE, un sistema context-aware per la gestione dei gruppi in ambienti MANET. AGAPE promuove e supporta lo sviluppo di applicazioni collaborative in scenari Mobile Ad-Hoc Network tramite la metafora di gruppo di entità. In AGAPE solo le entità di uno stesso gruppo possono collaborare fra di loro.
Il presente lavoro si è focalizzato sull'analisi, il design e l'implementazione del Join/Leave Manager Service (J/LMS) di AGAPE, che permette la creazione di nuovi gruppi on-demand, l'ingresso e l'uscita di nuovi membri dai gruppi e il discovery dei gruppi disponibili nella località. Inoltre è stato dettagliato il design del View Manager Service (VMS) che si occupa della gestione, e della disseminazione delle viste dei gruppi AGAPE. Infine, Il framework è stato testato tramite l'implementazione di un prototipo di applicazione nell'ambito della protezione civile. In particolare, il prototipo implementato permette la collaborazione fra i diversi operatori tramite scambio di messaggi.
I test effettuatati hanno dimostrato l'applicabilità di AGAPE al supporto di servizi collaborativi avanzati in ambienti MANET. I promettenti risultati conseguiti possono promuovere nuove attività di ricerca al fine di migliorare il supporto alla gestione dei gruppi esistente e di testarne l'applicabilità in diversi scenari applicativi.