SARAH
Presence Awareness, modalità disconnessa e dinamiche di update

Antonio Gaetani

Sommario

In tempi recenti lo sviluppo di applicazioni su reti ad hoc si sta orientando verso sistemi di pubblica utilità. In tale ambito si inserisce il nostro progetto: SARAH, uno shop assistant per disabili. Essendo frequenti in questo contesto partizionamenti di rete si è realizzato un meccanismo di Presence Awareness, mediante il quale è possibile individuare le entità con cui si interagisce, realizzare comportamenti differenti in relazione a chi ci è *vicino* e implementare dinamiche di update in caso di ricongiungimento di reti.

1 Scenario applicativo

1.1 Perchè una rete ad hoc

La grande diffusione di devices portatili e la crescita dell'interesse per le *Mobile Ad hoc NETworks* (MANET) hanno aperto nuovi orizzonti di comunicazione che prima non erano possibili. La collaborazione in reti manet pone all'attenzione del progettista una serie di problematiche riguardanti l'interazione e la gestione del concetto di gruppo. La vecchia idea di poter effettuare comunicazioni tra membri del gruppo che mantenessero proprietà di reliability, atomicity e synchronicity deve essere abbandonata per spostarsi verso sistemi maggiormente orientati a dinamiche best effort.

Non ci devono sorprendere quindi frequenti perdite di messaggi, connessioni e disconnessioni continue e problemi ulteriori (citiamo la questione dell'hidden terminal). Nonostante le suddette problematiche le reti ah hoc forniscono caratteristiche non ottenibili con reti "classiche", in quanto ad esempio non richiedono alcun tipo di server centrale, nè cablaggi di cavi o installazione di access point.

Tutto questo, per un utente non di alto profilo e senza particolari esigenze di sicurezza e affidabilità può essere visto come un grosso pregio, soprattutto se i meccanismi di creazione e gestione della manet vengono realizzati in modo trasparente.

1.2 SARAH

In questo contesto si inserisce il nostro progetto: SARAH, *Shop Assistant in Reti Ad Hoc*. Esso permette a persone anziane o disabili di richiedere un aiuto a soggetti al momento nell'ambiente ed a coordinare sia fruitori che fornitori dell'aiuto. Il concetto fondamentale che si trova dietro all'idea di coordinamento è quello di una lista condivisa e sincronizzata. Ovviamente il principio di sincronismo non è citato nel modo classico, ma inteso di tipo più lasco, nel quale non sempre è tutto perfettamente consistente.

Il caso d'uso principale del sistema consiste nell'arrivo di un disabile dotato di un dispositivo portatile all'interno di un ambiente (d'ora in poi supporremo che sia un supermercato) al cui interno è (o sarà) presente una rete ad hoc. Essendo affaticato o non fisicamente abile, e dovendo fare una spesa consistente, della quale abbiamo una rappresentazione all'interno della nostra applicazione, e con prodotti aventi locazioni nel negozio molto distanti tra loro, decide di richiedere aiuto. È predisposto quindi l'apposito bottone per mezzo del quale in modo trasparente all'utente viene creato un nuovo gruppo (individuato da un *gid*) e viene assegnato all'anziano un identificatore (il *pid* relativo). La richiesta di supporto (un particolare messaggio contente il *gid*) verrà inoltrata a tutti gli avventori dotati di dispositivi portatili connessi alla rete ad hoc sui quali è stata avviato SARAH e sui quali è stata espressa la volontà di essere un benefattore. Se i benefattori raggiunti dalla richiesta accettano di aderire al gruppo di aiuto, verrà loro inviata la lista relativa all'anziano che si sia deciso di aiutare. Da questo momento in poi quando qualcuno si troverà a poter prendere un prodotto per il disabile lo segnalerà checkando la corrispettiva entry nella propria lista locale. All'atto di questa azione dovrà essere riportato agli altri collaboratori ed al diretto interessato il cambiamento dello stato dell'elemento della lista (passando da "da prendere" a "preso").

Ovviamente non si vuole escludere la possibilità che un utente "benefattore" possa collaborare con più utenti in difficoltà. In questo caso bisognerà sapere oltre al "cosa" prendere, anche il "per chi" e a questo punto entrerà in gioco una feature di cui tratteremo in seguito: la *Presence Awareness*.

1.3 Problematiche di rete e scenario secondario

Le problematiche relative alla comunicazione in manet sono numerose e complesse. Basti citare il problema del routing in reti con una dinamicità molto elevata, nelle quali spesso avvengono partizioni e nodi cadono e ritornano attivi in breve tempo. Per poter lavorare ad un livello più elevato risulta molto utile l'utilizzo di un middleware che fornisca una serie di funzionalità sulle quali ci si possa appoggiare per la realizzazione dei propri obiettivi.

Considerando quanto detto bisogna prevedere una serie di meccanismi che ottengano un gestione quanto migliore possibile. Questo significa prevedere

dei meccanismi di update quando due entità si ritrovano nella stessa rete ma i loro dati non sono congruenti. Basandosi sul meccanismo della Presence Awareness è possibile determinare chi collabora con ciascuno, cosa che rende possibile predisporre dei comportamenti trasparenti all'utente che rendano le liste "sincronizzate". Con questo termine non intendiamo la completa congruenza, ma un tipo di consistenza che privilegi comunque la ridondanza anche a discapito dell'efficienza. Ad esempio prediligiamo una situazione in cui due benefattori prendano lo stesso item piuttosto del caso nel quale nessuno dei due lo faccia. Sicuramente non è una soluzione ottima, ma proprio per le problematiche intrinseche alle manet, a meno di costi proibitivi, qualsiasi tipo di interazione sarà best effort.

1.4 Scelta del middleware: AGAPE

Sono disponibili numerosi sistemi che lavorano su reti ad hoc, ciascuno dei quali è caratterizzato da alcune features peculiari. È responsabilità del progettista scegliere quello più adatto e che meglio fitta con i propri bisogni. Tra le varie opportunità disponibili la nostra scelta è caduta su AGAPE (*Allocation and Group Aware Pervasive Environment*), un sistema sviluppato presso il DEIS, che sebbene sia ancora in versione alfa fornisce un insieme di servizi che sono stati fondamentali nella realizzazione di SARAH.

Le motivazioni che ci hanno spinto ad utilizzare AGAPE sono state molteplici. In primis esso fornisce un meccanismo di routing basato sul gossiping (con probabilità a scelta) e gestisce il concetto di gruppo con primitive già implementate, del tipo *join/leave*. Quindi abbiamo un mezzo per creare gruppi e permettere a coloro che appartengono alla manet di effettuare operazioni di aggregazione o abbandono del gruppo.

La caratteristica per la quale, però, il middleware è unico è l'idea di vista. La vista non è altro che una tabella riassuntiva di coloro che sono nella mia prossimità, inviata a tutti i coloro che sono stati riconosciuti come mebri della rete con attiva una istanza di AGAPE. Il processo di creazione e distribuzione delle liste è gestita da entità centrali definite come LME (*Locality Manager Entity*), che consistono semplicemente in nodi della rete ad hoc con capacità computazionali maggiori. Lavorando sulla vista si può accedere ad informazioni riguardanti i *gid* dei vari gruppi, i *pid* e gli indirizzi dei vari membri e oltre a ciò ad un insieme di dati definito come profilo. Il profilo consiste in una collezione di dati riguardante un'entità ed è semplicemente caricabile da un file xml.

Come si può facilmente notare tutto questo scambio di dati causa un invio di un congruo numero di messaggi di dimensione anche rilevante, ma questo è un prezzo che vale la pena pagare per ottenere tutte le informazioni disponibili con le viste.

2 Presence Awareness

2.1 Aspetti generali

Numerose sono le motivazioni che ci spingono a focalizzare l' attenzione sul concetto di presenza e di individuazione dell' entità con cui si sta interagendo. Per poter realizzare efficacemente tutto ciò la prima cosa da definire è come identificare un attore del nostro sistema, che sia esso un disabile o un benefattore. Sfruttando la tecnologia messa a disposizione da AGAPE, possiamo utilizzare l' idea del profilo. Il profilo non è altro che una collezione di dati relativi ad un singolo utente, la quale viene distribuita insieme alla vista relativa ai gruppi cui esso fa parte. Poiché il middleware non forniva una metodo di configurazione da file, questo è stato realizzato mediante il parsing di un file xml con conseguente creazione del profilo definito prima del lancio effettivo dell' applicazione. Tramite ciò, quando ci si trova ad interagire con un altro membro del gruppo, sono disponibili una collezione di informazioni che lo riguardano, ad esempio possiamo semplicemente identificare il nome della persona bisognosa di aiuto. Non ci è però sufficiente in quanto, utilizzando il richiedente l' aiuto come entità centrale (che in effetti coordina le azioni dei suoi benefattori), è indispensabile sapere se esso sia attivo e reattivo ai nostri messaggi. Ricordando che siamo in reti ad hoc è fin troppo facile immaginare che in un qualche istante, anche per un semplice muro, non si possa più comunicare direttamente con l' ente centrale. Se non ci rendessimo mai conto di ciò che ci accade intorno il coordinamento tra gli attori del sistema risulterebbe quantomeno inefficacie ed al più scorretto.

2.2 Realizzazione

L' effettiva concretizzazione di un servizio di presenza si basa sul concetto di vista. Come si è già detto, mediante l' utilizzo di AGAPE, è disponibile su ogni macchina una collezione di informazioni inerenti le entità che sono nella mia prossimità e con cui quindi potrei comunicare direttamente. In realtà sono di interesse solo i dati di coloro che si sono assunti la responsabilità di prendere un *item* della lista del bisognoso e le informazioni relative al disabile. In questo ultimo caso in particolare ci interessa la sua presenza o meno nella nostra prossimità. Tutto ciò è realizzabile mediante un task che ad intervalli regolari controlla la vista disponibile e rileva i dati di interesse, quindi presenza/assenza del disabile e informazioni relativi ai propri vicini.

2.3 Modalità disconnessa e dinamiche di update

Risulta evidente come un benefattore si debba comportare diversamente in relazione alla presenza o meno del bisognoso. In caso di sua mancanza nella propria sottorete le modalità di interazione in caso di check di un item dovranno necessariamente cambiare, in modo tale che i nodi interagiscano

tra di loro per mantenere la più alta uniformità possibile. Inoltre all' atto di un eventuale ricongiungimento di due partizioni di rete, delle quali una contiene l' entità centrale, è indispensabile predisporre un protocollo di update mediante il quale le due parti possano assurgere ad uno stato congruente e aggiornato. Durante questa fase tornerà in gioco il disabile come entità centrale, in grado di garantire un elevato grado di consistenza e una risoluzione degli eventuali conflitti (se due benefattori hanno checkato un item a chi dei due deve essere effettivamente assegnato?). È importante notare come nella rete contenente il disabile tutte le dinamiche rimangano invariate e nessun cambiamento venga avvertito.

3 Presence Awareness in SARAH

3.1 Struttura dell'applicazione

Come si evince dalla figura 1 l'architettura di SARAH è di tipo 4-layer.

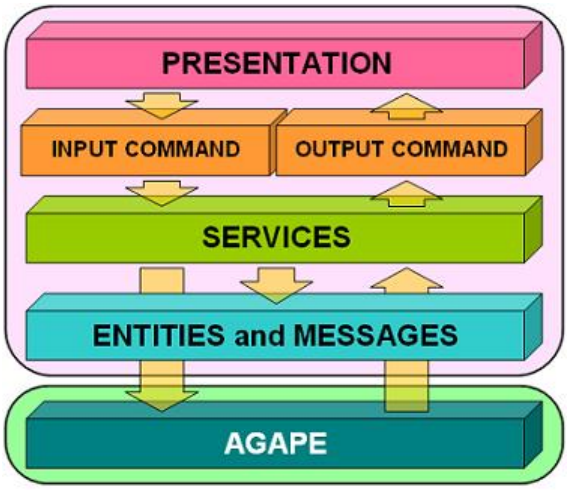

Figura 1: schema generale dell' architettura di SARAH

In realtà i livelli che sono interessati dalla gestione della presenza sono fondamentalmente due: quello dei servizi e quello relativo ad entità e messaggi (sebbene sia influenzato anche il layer dei comandi). Di seguito andremo nel dettaglio su come siano state realizzate la modalità disconnessa e le dinamiche di update e su quali entità ci siamo appoggiati.

3.2 Entities and Messages layer

A questo livello si avrà necessariamente bisogno di messaggi per il controllo del protocollo di aggiornamento e quindi di una gestione di questi ultimi.

I messaggi coinvolti dalle dinamiche introdotte sono:

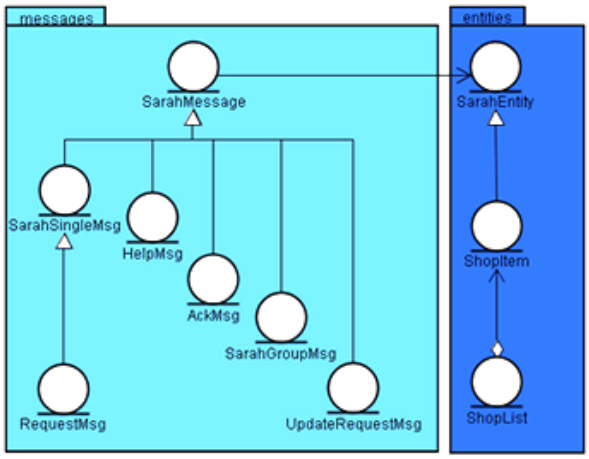

Figura 2: diagramma UML relativo al livello di entità e messaggi

- **SarahSingleMessage** Messaggio inviato per comunicare l' aggiornamento di un singolo elemento della lista. Solitamente di competenza dell' entità centrale, ma in modalità disconnessa è inviato anche dai benefattori;
- **UpdateRequestMessage** Messaggio che un benefattore usa per chiedere al disabile una reciproca sincronizzazione tra la sua lista e quella del disabile (è utilizzato quando un benefattore, rimasto temporaneamente in una partizione di rete in cui il disabile è assente, ottiene nuovamente la visibilità del disabile).

Al fine di distinguere i meccanismi di processing da attuare sulle singole tipologie di messaggi è stato realizzato un set di entità la cui responsabilità è la definizione delle azioni relative alla ricezione del messaggio. Queste sono utilizzate dai vari servizi, che in ricezione, richiedono a uno di tali processori di interpretarlo e reagire di conseguenza. Per rendere questo meccanismo quanto più flessibile e autocontenuto possibile abbiamo deciso di utilizzare un meccanismo di double dispatch. I processori coinvolti dalle dinamiche introdotte sono:

- **ProcessSingleMessage** Processore che deve modificare il modo di reazione, in quanto un *SarahSingleMessage* potrebbe provenire sia da un benefattore (con dati che andranno valutati prima di modificare i propri) o dal bisognoso (fonte di dati che non necessitano di alcun controllo);
- **ProcessUpdateRequest** Processore che si occupa della gestione di una richiesta di update fatta al disabile da un benefattore; contiene una lista di *item* che sono stati modificati.

3.3 Servizi

I servizi costituiscono, insieme ai processori dei messaggi, il cuore di SARAH. Vanno a formare infatti la parte dell' applicazione che si interfaccia con i servizi di AGAPE per usufruire delle primitive di comunicazione e dell' astrazione del concetto di gruppo.

È diventato fondamentale andare a interrogare periodicamente il middleware per capire di chi si ha visibilità e, in particolare, se si ha visibilità del disabile; queste informazioni sono necessarie separatamente per ogni gruppo al quale ci si è joinati (ovvero, per ogni disabile che si sta aiutando), oltre che per il gruppo che eventualmente si è creato.

A tale scopo è presente un servizio, il *PresenceService* (uno per ciascun gruppo cui apparteniamo). Ogni *PresenceService* è un componente attivo/passivo (un oggetto con un processo interno) che monitora la visibilità degli altri utenti in un certo gruppo, interrogando periodicamente il gestore delle viste di AGAPE e notificando i livelli superiori ogni qual volta la lista degli utenti visibili cambia (ovvero quando si perde o si riottiene visibilità di qualcuno). Inoltre, contiene l' informazione riguardante la presenza o meno del disabile che ha creato il gruppo (tale dato è fondamentale per cambiare protocollo di coordinamento in base al contesto). È il servizio stesso quindi che esegue una serie di azioni particolari nei momenti di transizione (disabile non visibile → disabile visibile e viceversa).

Per coordinare l' insieme dei servizi di presenza, è presente un *ServicePooler* (entità singleton), che restituisce, dato il *group identifier* o il *pid* del disabile, il corrispondente *PresenceService*.

3.4 Comandi

Il layer dei comandi ha il compito di separare completamente la logica dell' applicazione dalla sua interfaccia utente (si faccia riferimento allo schema 3. Il comando di presenza non dispone di un comando di input, poichè è svincolato dalle azioni dell' utente (la gestione è completamente a carico dei livelli inferiori). Serve invece un comando di output per comunicare all' utente chi è in quel momento visibile, e chi, nel caso di un item checkato, si è preso la responsabilità di prenderlo. Per rendere più flessibile il meccanismo di notifica, il *PresenceService* non chiama direttamente il comando, ma funge da *Observable* notificando, alla modifica dell' elenco dei "presenti", gli osservatori che si sono registrati. Tramite ciò è possibile esporre all' utente una lista di coloro che sono nelle nostre vicinanze rappresentati dai nomi che hanno deciso di inserire nel proprio profilo e di contrassegnare gli item già presi con il nome di colui che se ne è assunta la responsabilità.

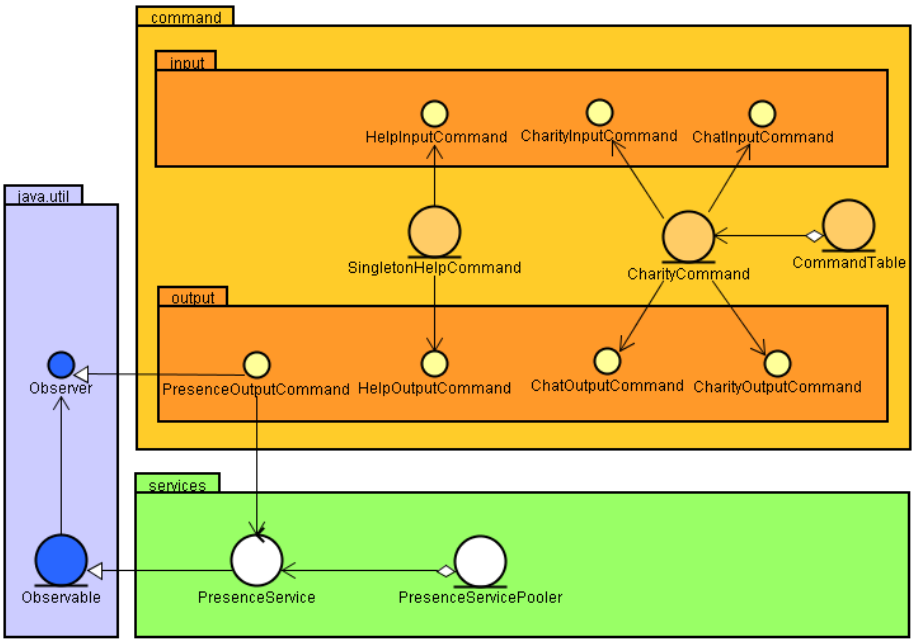

Figura 3: diagramma UML relativo al livello dei comandi

4 Protocollo di coordinamento

4.1 Obiettivi e proprietà

Risulta evidente come al contrario di tutti gli altri protocolli realizzati da SARAH, per quello di riallineamento delle liste non intervengono entità a livello comandi o presentazione. Infatti il protocollo del quale ci andiamo ora ad occupare è gestito interamente a livello di servizi e processori di messaggi. L' obiettivo ideale è quello di fare in modo che istante per istante, tutti i componenti del gruppo vedano la stessa lista. Ovviamente, tale proprietà è praticamente irrealizzabile, soprattutto ricordandoci che ci troviamo in ambiente ad hoc e che quindi si possono verificare eventi come quello rappresentato in figura Cioè un partizionamento del gruppo per cui alcuni componenti perdono visibilità di altri, con la conseguenza che l' insieme originario si ritrova frammentato.

Fenomeni di continue entrate/uscite dalla rete (come quando un dispositivo è ai limiti del raggio di copertura), vengono fortunatamente gestite dal middleware, che si preoccupa di eliminare un utente dalla vista solo dopo che manca da un certo tempo (tale utente, ovviamente, perderà comunque dei messaggi), ossia dopo che per due volte consecutive il suo Proximity Service non invia all' LME del gruppo il Beacon (inteso come testimonianza di presenza).

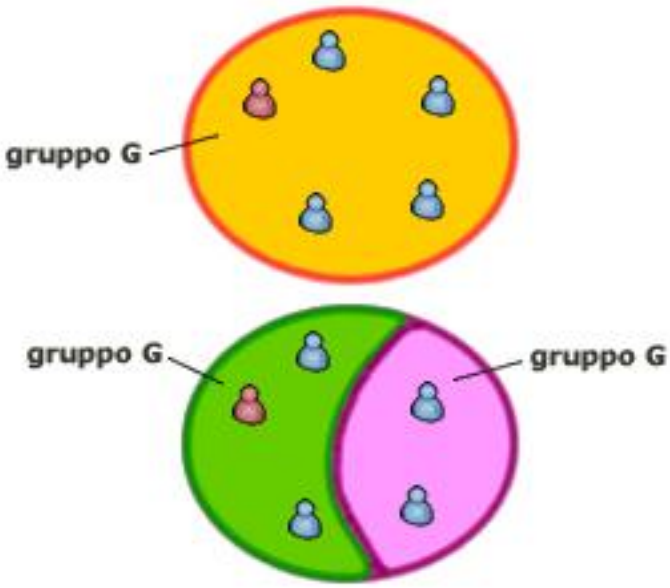

Figura 4: fenomeno di partizionamento del gruppo

Preso atto del fatto che le liste dei vari componenti del gruppo possono essere inconsistenti fra loro, diciamo che:

Proprietà 1 *Le liste dei partecipanti possono divergere in positivo, ovvero può capitare che più utenti prendano lo stesso elemento.*

Tale proprietà va comunque "limitata"; non possiamo ammettere la situazione in cui tutti i partecipanti prendono tutti gli elementi. Il nostro obiettivo diventa quindi quello di *minimizzare il numero di queste inconsistenze*! Il componente del gruppo che si accolla questo onere decisionale è il disabile. Se esso viene a mancare dalla prossimità di un gruppo di benefattori dobbiamo tener presente che meccanismi di elezione non sono applicabili nelle manet. Dobbiamo quindi rilassare ulteriormente i vincoli sull' inconsistenza fra le liste, permettendo quindi ai benefattori "isolati" di continuare a lavorare interagendo fra loro. In tal caso, come vedremo in seguito, si impone una fase di riconciliazione di liste quando il disabile torna visibile, fase in cui si cercano di rendere congruenti le liste di tutti. Si ricordi che il servizio di presenza si accorge delle situazioni in cui si passa da visibilità ad assenza del disabile e viceversa, permettendo di capire in che modalità si debba lavorare.

4.2 Descrizione del protocollo

Quando un benefattore si accorge dell' assenza del disabile, deve comunque continuare a lavorare cercando il più possibile di aggiornare i presenti sulle

proprie modifiche. La soluzione da noi implementata, che comunque evita di sovraccaricare la rete di messaggi (cosa particolarmente spiacevole in un ambiente ad-hoc), è semplicemente quella di spedire a tutti la propria modifica. In questo caso, non essendoci il disabile e non essendoci nessun protocollo di elezione, "ognuno è padrone di sè stesso", ovvero decide se accettare o meno la modifica ricevuta (la decisione riguarda sempre il fatto che se nella mia lista l' elemento per cui mi è arrivato un messaggio di check è già checkato, non modifico nulla). La parte centrale della figura 5 mostra infatti che le modifiche di ognuno, in tale contesto, possono portare le liste ad essere sensibilmente differenti. Ricordandoci comunque che tale fatto è spiacevole, poichè vorremmo che le liste fossero quanto più possibile congruenti, imponiamo che, quando il disabile ritorna visibile (prima o poi ciò dovrà accadere per forza, a meno di chiusura definitiva dell' applicazione), ci sia una fase di riconciliazione. Quando un benefattore perde visibilità del disabile, salva in locale lo stato corrente della lista; una volta ritrovatolo, può calcolare in cosa l' insieme corrente diverge dalla precedente "fotografia", e inviare alla ritornata entità centrale la lista degli item che differiscono. Il disabile risolve le incongruenze dando precedenza ai suoi dati (quindi se una sua entry risulta presa da A e gli arriva l' informazione che anche B se ne è assunta la responsabilità, ne risulterà alla fine della procedura ancora A il gestore) e invia nuovamente a tutti i benefattori gli item in modo che venga raggiunto un nuovo stato consistente. È necessario specificare come il flusso più ingente di dati si verifichi in seguito alla richiesta di update inoltrata dal primo nodo di una sottorete che ricomincia a sentire la presenza del bisognoso. Infatti, se la sottorete si fosse mantenuta in uno stato totalmente consistente, ci sarebbe un unico flusso di messaggi di aggiornamento da parte dell' ente centrale in quanto tutte le incongruenze sarebbero risolte al confronto con il primo messaggio di richiesta di refresh della lista. Proprio durante questa fase si risolvono situazioni del tipo evidenziato in figura, nelle quali due benefattori si ritengono responsabilità di un item. Viene effettivamente assegnata la gestione dell' item al primo che invia al disabile una richiesta di update.

Indice

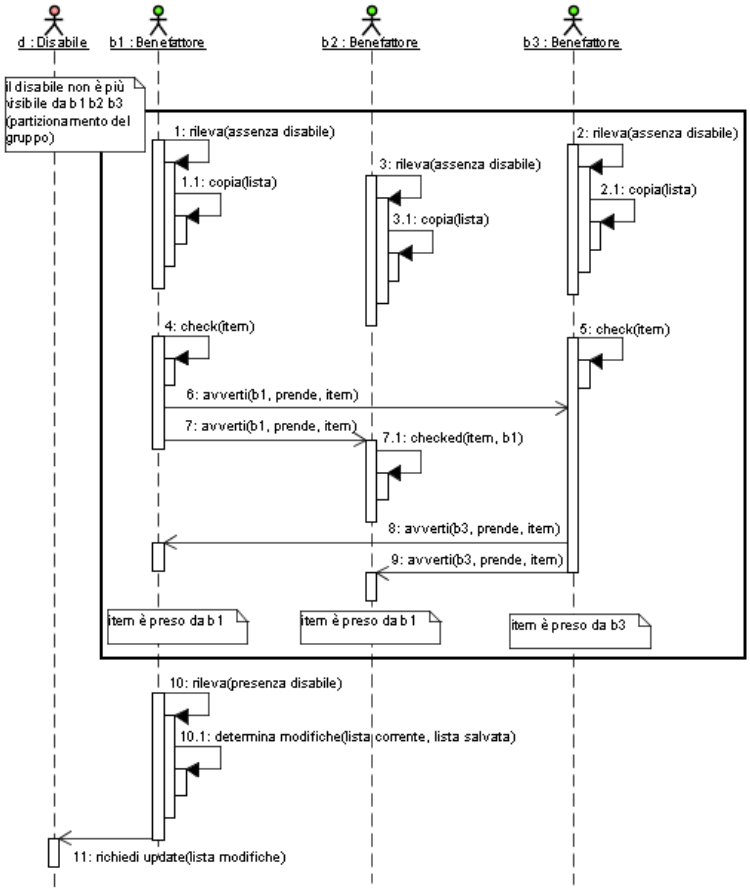

Figura 5: comportamento in assenza del disabile

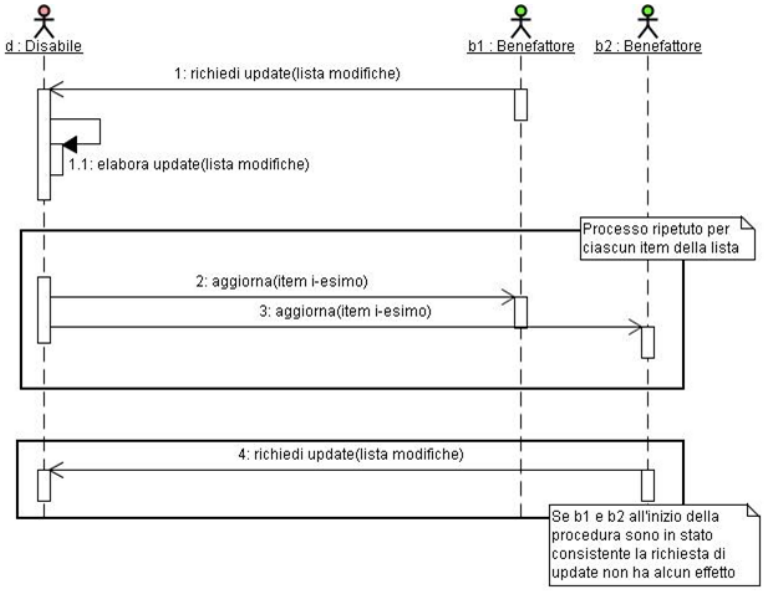

Figura 6: dinamiche di update