POLITECNICO DI MILANO

SCUOLA DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

A3Droid: un’implementazione dello stile architetturale A3 per dispositivi mobili

Relatore: Prof. Sam Guinea

Tesi di Laurea di: Francesco Castelli, matricola 765773

Anno Accademico 2013/2014

Indice

Indice 2

Indice delle figure 4

Indice dei grafici 5

Estratto – Abstract 9

1 Introduzione 11

2 Stato dell'arte 16

3 AllJoyn 30
3.1 Architettura 30
3.2 BusAttachment 35
3.3 Advertising 35
3.4 Discovery 36
3.5 Sessioni 38
3.6 BusInterface 40
3.7 BusMehod 41
3.8 BusProperty 41
3.9 BusSignal e sessionless signals 42
3.10 BusObject e object path 43
3.11 Comportamento di un'applicazione AllJoyn 44

4 A-3 46
4.1 A3JG 50
4.2 Perché A3Droid? 50

5 A3Droid 52
5.1 Il gruppo 54
5.2 La creazione di un gruppo 55
5.3 La distruzione di un gruppo 56
5.4 La classe A3Node 58
5.5 Le classi A3Role, A3SupervisorRole, A3FollowerRole e GroupDescriptor 66
5.6 Il gruppo "wait" 68
5.7 L'elezione del supervisore 69
5.8 La classe A3Message 71
5.9 Come creare un'applicazione A3Droid 72

6 Test 75
6.1 A3Test_1 76

6.2 A3Test_3 79
6.3 A3Test_5 97
6.4 A3Test_7 108
6.5 Discussione dei risultati ottenuti 113

7 Conclusioni **116**

Bibliografia **118**

Indice delle figure

Figura 1	Uso del framework in ambiente sanitario.	16
Figura 2	Struttura di un sistema autonomico.	19
Figura 3	Infrastruttura di controllo del framework Kinesthetics eXtreme.	21
Figura 4	Architettura del framework Aura.	22
Figura 5	Architettura del framework MobileSpaces.	23
Figura 6	Architettura del framework Ponder2.	25
Figura 7	Architettura autonomica affidabile.	26
Figura 8	Architettura di GridLite.	29
Figura 9	Applicazioni, router e loro topologie comuni.	31
Figura 10	Struttura di un'applicazione AllJoyn.	32
Figura 11	Versioni standard e thin di AllJoyn.	33
Figura 12	Struttura funzionale del router AllJoyn.	34
Figura 13	Comunicazione tra dispositivi in AllJoyn.	34
Figura 14	Instaurazione di una sessione AllJoyn.	40
Figura 15	Corrispondenza tra well-known name, nomi delle BusInterface e object path.	43
Figura 16	Workflow di un'applicazione AllJoyn.	45
Figura 17	Composizione di gruppi.	48
Figura 18	Corrispondenze tra A-3, AllJoyn e le API da me realizzate.	54
Figura 19	Le operazioni "stack" e "reverseStack".	60
Figura 20	Le operazioni "peers" e "reversePeers".	62
Figura 21	Le operazioni “hierarchy" e "reverseHierarchy".	63
Figura 22	L'operazione "merge".	64
Figura 23	Diagramma delle classi delle API da me realizzate.	73
Figura 24	L'interfaccia grafica dell'applicazione A3Test_1.	78
Figura 25	L'interfaccia grafica dell'applicazione A3Test_3.	81
Figura 26	L'interfaccia grafica dell'applicazione A3Test_5.	98
Figura 27	L'interfaccia grafica dell'applicazione A3Test_7.	109

Indice dei grafici

Grafico 1 Numero medio di messaggi da 62484 Byte (in ordinata) scambiati tra i nodi in funzione del numero di gruppi (in ascissa) e di telefoni presenti nel sistema. 82

Grafico 2 Durata media dell'esperimento (espressa in ns, in ordinata) in funzione del numero di gruppi (in ascissa) e di telefoni presenti nel sistema. La dimensione dei messaggi scambiati è 62484 Byte. 83

Grafico 3 Frequenza media di spedizione di messaggi da 62484 Byte (in ordinata, espressa in mex/s) in funzione del numero di gruppi (in ascissa) e di telefoni presenti nel sistema. 83

Grafico 4 Numero medio di messaggi da 32000 Byte (in ordinata) scambiati tra i nodi in funzione del numero di gruppi (in ascissa) e di telefoni presenti nel sistema. 84

Grafico 5 Durata media dell'esperimento (espressa in ns, in ordinata) in funzione del numero di gruppi (in ascissa) e di telefoni presenti nel sistema. La dimensione dei messaggi scambiati è 32000 Byte. 85

Grafico 6 Frequenza media di spedizione di messaggi da 32000 Byte (in ordinata, espressa in mex/s) in funzione del numero di gruppi (in ascissa) e di telefoni presenti nel sistema. 85

Grafico 7 Numero medio di messaggi da 5017 Byte (in ordinata) scambiati tra i nodi in funzione del numero di gruppi (in ascissa) e di telefoni presenti nel sistema. 86

Grafico 8 Durata media dell'esperimento (espressa in ns, in ordinata) in funzione del numero di gruppi (in ascissa) e di telefoni presenti nel sistema. La dimensione dei messaggi scambiati è 5017 Byte. 87

Grafico 9 Frequenza media di spedizione di messaggi da 5017 Byte (in ordinata, espressa in mex/s) in funzione del numero di gruppi (in ascissa) e di telefoni presenti nel sistema. 87

Grafico 10 Numero medio di messaggi da 1812 Byte (in ordinata) scambiati tra i nodi in funzione del numero di gruppi (in ascissa) e di telefoni presenti nel sistema. 88

Grafico 11 Durata media dell'esperimento (espressa in ns, in ordinata) in funzione del numero di gruppi (in ascissa) e di telefoni presenti nel sistema. La dimensione dei messaggi scambiati è 1812 Byte. 89

Grafico 12 Frequenza media di spedizione di messaggi da 1812 Byte (in ordinata, espressa in mex/s) in funzione del numero di gruppi (in ascissa) e di telefoni presenti nel sistema. 89

Grafico 13 Numero medio di messaggi spediti da 2 telefoni (in ordinata) in funzione del numero di gruppi (in ascissa) e della dimensione dei messaggi scambiati. 90

Grafico 14 Numero medio di messaggi spediti da 3 telefoni (in ordinata) in funzione del numero di gruppi (in ascissa) e della dimensione dei messaggi scambiati. 91

Grafico 15 Numero medio di messaggi spediti da 4 telefoni (in ordinata) in funzione del numero di gruppi (in ascissa) e della dimensione dei messaggi scambiati. 91

Grafico 16 Numero medio di messaggi spediti da 5 telefoni (in ordinata) in funzione del numero di gruppi (in ascissa) e della dimensione dei messaggi scambiati. 92

Grafico 17 Durata media dell'esperimento (espressa in ns, in ordinata) su 2 telefoni in funzione del numero di gruppi (in ascissa) e della dimensione dei messaggi scambiati. 93

Grafico 18 Durata media dell'esperimento (espressa in ns, in ordinata) su 3 telefoni in funzione del numero di gruppi (in ascissa) e della dimensione dei messaggi scambiati. 93

Grafico 19 Durata media dell'esperimento (espressa in ns, in ordinata) su 4 telefoni in funzione del numero di gruppi (in ascissa) e della dimensione dei messaggi scambiati. 94

Grafico 20 Durata media dell'esperimento (espressa in ns, in ordinata) su 5 telefoni in funzione del numero di gruppi (in ascissa) e della dimensione dei messaggi scambiati. 94

Grafico 21 Frequenza media di spedizione dei messaggi tra 2 telefoni (in ordinata, espressa in mex/s) in funzione del numero di gruppi (in ascissa) e della dimensione dei messaggi scambiati. 95

Grafico 22 Frequenza media di spedizione dei messaggi tra 3 telefoni (in ordinata, espressa in mex/s) in funzione del numero di gruppi (in ascissa) e della dimensione dei messaggi scambiati. 95

Grafico 23 Frequenza media di spedizione dei messaggi tra 4 telefoni (in ordinata, espressa in mex/s) in funzione del numero di gruppi (in ascissa) e della dimensione dei messaggi scambiati. 96

Grafico 24 Frequenza media di spedizione dei messaggi tra 5 telefoni (in ordinata, espressa in mex/s) in funzione del numero di gruppi (in ascissa) e della dimensione dei messaggi scambiati. 96

Grafico 25 Tempi medi (in ordinata, espressi in ns) di risposta dell'ultimo follower ad un messaggio di 62484 Byte inviato dal supervisore, in funzione del numero di telefoni e di gruppi presenti nel sistema. In ascissa è espresso il numero di follower destinatari del messaggio. 100

Grafico 26 Tempi medi (in ordinata, espressi in ns) di risposta dell'ultimo follower ad un messaggio di 32000 Byte inviato dal supervisore, in funzione del numero di telefoni e di gruppi presenti nel sistema. In ascissa è espresso il numero di follower destinatari del messaggio. 101

Grafico 27 Tempi medi (in ordinata, espressi in ns) di risposta dell'ultimo follower ad un messaggio di 5017 Byte inviato dal supervisore, in funzione del numero di telefoni e di gruppi presenti nel sistema. In ascissa è espresso il numero di follower destinatari del messaggio. 102

Grafico 28 Tempi medi (in ordinata, espressi in ns) di risposta dell'ultimo follower ad un messaggio di 1812 Byte inviato dal supervisore, in funzione del numero di telefoni e di gruppi presenti nel sistema. In ascissa è espresso il numero di follower destinatari del messaggio. 103

Grafico 29 Tempi medi (in ordinata, espressi in ns) di risposta dell'ultimo follower ad un messaggio inviato dal supervisore in un gruppo di 2 telefoni, in funzione del numero di gruppi presenti nel sistema (in ascissa) e della dimensione del messaggio. 104

Grafico 30 Tempi medi (in ordinata, espressi in ns) di risposta dell'ultimo follower ad un messaggio inviato dal supervisore in un gruppo di 3 telefoni, in funzione del numero di gruppi presenti nel sistema e della dimensione del messaggio. In ascissa è espresso il numero di follower destinatari del messaggio. 105

Grafico 31 Tempi medi (in ordinata, espressi in ns) di risposta dell'ultimo follower ad un messaggio inviato dal supervisore in un gruppo di 4 telefoni, in funzione del numero di gruppi presenti nel sistema e della dimensione del messaggio. In ascissa è espresso il numero di follower destinatari del messaggio. 106

Grafico 32 Tempi medi (in ordinata, espressi in ns) di risposta dell'ultimo follower ad un messaggio inviato dal supervisore in un gruppo di 5 telefoni, in funzione del numero di gruppi presenti nel sistema e della dimensione del messaggio. In ascissa è espresso il numero di follower destinatari del messaggio. 107

Grafico 33 Tempi medi di risposta (in ordinata, espressi in ns) di un telefono ad un messaggio di elezione del supervisore in un gruppo di 2 telefoni, in funzione del numero di gruppi presenti nel sistema. I numeri in ascissa identificano l'ordine delle risposte ricevute. 109

Grafico 34 Tempi medi di risposta (in ordinata, espressi in ns) di un telefono ad un messaggio di elezione del supervisore in un gruppo di 3 telefoni, in funzione del numero di gruppi presenti nel sistema. I numeri in ascissa identificano l'ordine delle risposte ricevute. 110

Grafico 35 Tempi medi di risposta (in ordinata, espressi in ns) di un telefono ad un messaggio di elezione del supervisore in un gruppo di 4 telefoni, in funzione del numero di gruppi presenti nel sistema. I numeri in ascissa identificano l'ordine delle risposte ricevute. 110

Grafico 36 Tempi medi di risposta (in ordinata, espressi in ns) di un telefono ad un messaggio di elezione del supervisore in un gruppo di 2 telefoni, in funzione del numero di gruppi presenti nel sistema. I numeri in ascissa identificano l'ordine delle risposte ricevute. 111

Grafico 37 Tempi medi di risposta (in ordinata, espressi in ns) di un telefono ad un messaggio di elezione del supervisore, in funzione del numero di telefoni presenti in un gruppo. I numeri in ascissa identificano l'ordine delle risposte ricevute. Nel sistema è presente un gruppo solo. 112

Grafico 38 Tempi medi di risposta (in ordinata, espressi in ns) di un telefono ad un messaggio di elezione del supervisore, in funzione del numero di telefoni presenti in un gruppo. I numeri in ascissa identificano l'ordine delle risposte ricevute. Nel sistema sono presenti 2 gruppi. 112

Grafico 39 Tempi medi di risposta (in ordinata, espressi in ns) di un telefono ad un messaggio di elezione del supervisore, in funzione del numero di telefoni presenti in un gruppo. I numeri in ascissa identificano l'ordine delle risposte ricevute. Nel sistema sono presenti 3 gruppi. 113

Estratto - Abstract

Sempre più persone sono dotate di smart phone e di apparecchiature in cui sono presenti sensori collegati tra loro in un qualche tipo di rete. La diffusione di massa di tali apparecchiature porta inevitabilmente alla creazione di sistemi sempre più complessi, più grandi e più difficili da gestire. A tutto ciò si aggiunga anche che la configurazione di tali sistemi cambia in continuazione e che i sistemi stessi devono mantenere una data qualità del servizio in tutte le possibili situazioni. Per garantire a questi sistemi la scalabilità, la robustezza, la tolleranza ai guasti e la velocità di reazione ai cambiamenti di cui essi hanno bisogno, è necessario che i nodi al loro interno siano coordinati in maniera decentralizzata. A-3 è uno stile architetturale per la progettazione, l'implementazione, la gestione ed il coordinamento di sistemi mobile distribuiti che presentino un elevato numero di nodi al loro interno e che siano altamente volatili. E' incentrato sulla scalabilità e l'autoadattamento di tali sistemi e fronteggia il problema introducendo i concetti di "gruppo" e di "composizione di gruppi": il primo consente di coordinare come un'unica entità i nodi facenti parte di uno stesso gruppo, mentre il secondo consente la gestione dell'intero sistema relazionando tra loro i diversi gruppi in modo gerarchico e cooperativo. A3Droid è un'implementazione di A-3, da me realizzata per piattaforma Android. Le sue API consentono di creare, distruggere e comporre tra loro i gruppi che si prevede faranno parte del sistema. In particolare, esse possono essere invocate automaticamente dall'applicazione per dividere un gruppo sovrappopolato o unire tra loro gruppi sottopopolati, al fine di garantire la scalabilità del sistema al variare del numero di nodi al suo interno. Scopo di questo lavoro è illustrare A3Droid: la sua struttura, le sue prestazioni ed i miglioramenti ad esso apportabili in futuro.

An increasing number of people owns a smart phone and devices with sensors linked to each other through some kind of network. The large diffusion of these devices leads to the creation of more complex, bigger and harder to manage systems. The configuration of such systems changes frequently and such systems must offer a certain quality of service in each possible situation. To guarantee these system the

scalability, the robustness, the fault tolerancee and the speed of reaction to changes they need, a decentralized coordination of their nodes is needed. A-3 is an architectural style for designing, implementation, management and coordination of distributed mobile systems which are highly dynamic and composed of a very large number of nodes. It is centered on scalability and self-adaptation of such systems and it copes with the problem by introducing the concepts of "group" and of "composition of groups": the former allows to coordinate all the nodes in the same group as a single entity, while the latter allows to manage the whole system by relating all groups in a hierarchical and cooperative manner.

A3Droid is an implementation of A-3, that I realized for Android platform. Its APIs allow to create, to destroy and to compose the groups in the system. In particular, they can be automatically called by the application to divide an oversized group or to merge undersized groups, in order to guarantee the scalability of the system while varying the number of nodes into it. The goal of this work is to present A3Droid: its structure, its performance and the possible improvements to it.

1 Introduzione

Oggetto di questo lavoro sono i sistemi distribuiti pervasivi, i quali sono impiegati in un'ampia varietà di applicazioni come monitoraggio ambientale, tracciamento della vita selvaggia, agricoltura intelligente, automazione domestica, trasporti intelligenti, sorveglianza, risposta alle emergenze, ottimizzazione dell'uso di risorse, socializzazione, gestione di eventi di massa. Si tratta di sistemi mobile distribuiti, altamente dinamici e costituiti da un elevato numero di nodi. Tale tipo di sistemi si sta diffondendo rapidamente, dal momento che sempre più persone sono dotate di smart phone e di apparecchiature in cui sono presenti sensori collegati tra loro in un qualche tipo di rete. La diffusione di massa di tali apparecchiature porta inevitabilmente alla creazione di sistemi sempre più complessi, più grandi e più difficili da gestire. A tutto ciò si aggiunga anche che la configurazione di tali sistemi cambia in continuazione e che i sistemi stessi devono mantenere una data qualità del servizio in tutte le possibili situazioni.

Per far sì che così tanti nodi si comportino in modo da raggiungere un obiettivo comune, è necessaria una coordinazione decentralizzata degli stessi. La decentralizzazione della coordinazione presenta una robustezza, una tolleranza ai guasti ed un adattamento ai cambiamenti dell'ambiente d'esecuzione che una coordinazione centralizzata invece non presenta. La coordinazione decentralizzata dei nodi, inoltre, tende ad evitare la formazione di colli di bottiglia, il cui fallimento determina un pesante rallentamento dell'intero sistema, se non il fallimento del sistema stesso. In sistemi come quelli in esame, la comunicazione è vitale e deve essere semplice e veloce: rallentamenti e fallimenti del sistema devono essere il più possibile evitati.

Un altro vantaggio della coordinazione decentralizzata dei nodi rispetto invece ad una coordinazione centralizzata degli stessi sta nella maggiore scalabilità che essa può offrire. I sistemi distribuiti pervasivi sono infatti altamente dinamici: in essi, i

nodi entrano ed escono in numero elevato e con elevata frequenza. Ognuno di tali nodi è anche soggetto al fallimento, e questo è inevitabile. Il sistema deve dunque essere in grado di reagire correttamente e velocemente ai cambiamenti della sua configurazione dovuta ai suddetti fattori.

Allo stato attuale, si veda il capitolo 2, gli approcci al problema sono diversi: alcuni mettono al centro i dati, puntando a distribuire meglio possibile il carico tra i nodi del sistema, costituito tipicamente da una griglia; altri si concentrano sui cicli di controllo globale e di ogni singolo elemento del sistema, mirando a migliorarne l'efficacia; altri ancora si focalizzano sulle politiche di gestione dei nodi del sistema; altri mirano a fornire all'utente i servizi da lui richiesti, recuperando le risorse opportune in ambienti diversi, come ad esempio casa ed ufficio; altri sono sistemi mobile agent, costituiti cioè da pezzi di codice che migrano tra i vari nodi a seconda delle proprie esigenze. Ognuno di essi indirizza un aspetto importante nella realizzazione di sistemi autoadattativi, ma nessuno di essi è incentrato sul garantire la scalabilità del sistema a fronte delle variazioni del numero di nodi in esso presenti.

Lo stile architetturale A-3 sopperisce a questa mancanza, in quanto appositamente ideato in modo da essere incentrato sulla scalabilità e l'autoadattamento di sistemi mobile distribuiti, altamente dinamici e composti da un elevato numero di nodi. A-3 introduce due concetti innovativi, assenti in tutti gli altri approcci: quelli di "gruppo" e di "composizione di gruppi". Un gruppo consente al progettista del sistema di coordinare più componenti come se fossero un'entità unica e di definire in maniera più semplice come essi debbano collaborare e come debbano essere gestiti. I componenti di un gruppo possono memorizzare all'interno del gruppo stesso informazioni rilevanti per l'applicazione ed il coordinamento in modo sicuro e robusto, al fine di recuperarle quando necessario. All'interno di un gruppo, un nodo può assumere due ruoli, tra essi mutuamente esclusivi: quello di supervisore o quello di follower. Il supervisore è il nodo incaricato della gestione del gruppo, mentre un follower si unisce al gruppo per capire come comportarsi nella situazione corrente. In ogni gruppo può esistere solo un supervisore, mentre possono esserci più follower. Ogni follower può comunicare solo col supervisore e solo in maniera unicast, mentre

il supervisore può scegliere di comunicare con i follower in modalità unicast, multicast o broadcast. La composizione dei gruppi consente la coordinazione decentralizzata dell'intero sistema e consiste in un insieme di relazioni gerarchiche e collaborative tra i gruppi del sistema. Essa è realizzata offrendo ai nodi del sistema la capacità di poter appartenere a più gruppi contemporaneamente e di poter avere ruoli diversi in gruppi diversi.

L'ideazione di A-3 è relativamente recente, dunque era necessario creare un'implementazione di riferimento. A tale scopo, ho creato A3Droid: un'implementazione di A-3 per piattaforma Android, il cui utilizzo è estendibile anche ad altre piattaforme grazie al framework AllJoyn, sul quale ne ho poggiato la struttura. Oltre a garantire l'interoperabilità tra piattaforme diverse, AllJoyn ha il vantaggio di astrarre diversi tipi di rete come un bus, al quale le applicazioni accedono mediante un oggetto detto BusAttachment. Inoltre, AllJoyn fornisce le API di comunicazione unicast e broadcast utilizzate in A3Droid. La comunicazione multicast in A3Droid è invece implementata come serie di comunicazioni unicast AllJoyn.

Le API di A3Droid consentono di creare, distruggere e comporre tra loro i gruppi che si prevede faranno parte del sistema. In particolare, esse possono essere invocate automaticamente dall’applicazione per dividere un gruppo sovrappopolato o unire tra loro gruppi sottopopolati, al fine di garantire la scalabilità del sistema al variare del numero di nodi al suo interno. La struttura di A3Droid consente di realizzare applicazioni in modo facile e veloce. Basta avere chiaro dall'inizio:

- quali gruppi possono essere presenti nel sistema;
- quali ruoli possono assumere i nodi in ogni gruppo, cioè il comportamento del supervisore e dei follower in ciascun gruppo.

Ogni gruppo è caratterizzato da un descrittore che ne contiene il nome ed i ruoli che i nodi possono assumere in esso. Un ruolo è una classe Java astratta, che deve essere estesa per implementare la logica del nodo e la reazione di quest'ultimo alla ricezione di un messaggio applicativo. Il nodo è rappresentato da un'apposita classe Java, alla

quale è sufficiente passare una lista di ruoli e di descrittori per poter usufruire di tutte le funzionalità offerte da A3Droid.

Dopo aver realizzato le API di A3Droid, ne ho poi valutato le prestazioni, realizzando applicazioni ottenute estendendo opportunamente le classi di A3Droid stesso rappresentanti ruoli e descrittori dei gruppi. Ho pensato che tali prestazioni potessero essere influenzate dai seguenti fattori:

- numero di telefoni presenti nel sistema;
- numero di gruppi presenti nel sistema;
- numero di telefoni in un gruppo;
- numero di gruppi a cui può essere connesso un telefono;
- dimensione dei messaggi scambiati tra i telefoni;

e che dunque fosse necessaria una misura della scalabilità di A-3. Ho quindi realizzato tre applicazioni diverse per valutare, in funzione dei suddetti fattori, rispettivamente:

- tempo necessario ai follower per ricevere risposta ad un messaggio da essi inviato al supervisore;
- tempo necessario ai follower per rispondere ai messaggi che il supervisore invia loro utilizzando i diversi tipi di comunicazione possibili in A-3 (unicast, multicast e broadcast);
- tempo necessario affinché tutti i follower rispondano ad un messaggio di inizio di elezione di un nuovo supervisore.

Riassumendo i risultati ottenuti dall'esecuzione di tali applicazioni, posso dire che:

- le prestazioni peggiorano all’aumentare del numero di telefoni,di gruppi e della dimensione dei messaggi scambiati;
- il passaggio da una configurazione a due telefoni ad una configurazione a tre telefoni determina un peggioramento delle prestazioni dovuto al passaggio di una sessione AllJoyn da singlepoint a multipoint;
- il tempo di risposta di ogni follower ai messaggi del supervisore aumenta all’aumentare del numero di telefoni e di gruppi presenti nel sistema e

l'aumentare del numero di telefoni ha un impatto maggiore sul tempo di risposta rispetto alla creazione di nuovi gruppi;

- in presenza di uno, due, o tre gruppi, la risposta ad un messaggio di inizio dell'elezione di un nuovo supervisore avviene entro 60 ms, indipendentemente dal numero di telefoni;
- in presenza di due telefoni e quattro o cinque gruppi, la risposta al messaggio d'inizio dell'elezione di un nuovo supervisore avviene entro 120 ms;
- le prestazioni di A3Droid peggiorano all 'aumentare della dimensione dei messaggi scambiati ed il peggioramento è più evidente per le dimensioni dei messaggi più piccole;
- si ottengono prestazioni ottimali di A3Droid se la distribuzione dei supervisori dei vari gruppi è uniforme sui telefoni;
- il numero di gruppi creabili nel sistema diminuisce all'aumentare del numero di telefoni in esso presenti ed è pari a 2 in presenza di 5 telefoni;
- il collo di bottiglia di un'applicazione A3Droid è il supervisore.

Il codice di A3Droid si è rivelato non essere adeguatamente scalabile per operare in sistemi di migliaia di nodi. Le prestazioni di A3Droid si sono rivelate scarse, a causa di un utilizzo non ottimale del framework AllJoyn, sul quale ho costruito la struttura di A3Droid. Le prove da me effettuate dimostrano che, per il futuro, è possibile ottenere sensibili miglioramenti riducendo il più possibile il numero di BusAttachment AllJoyn e facendo sì che su ogni dispositivo mobile sia presente al più un supervisore. E' inoltre possibile accelerare le comunicazioni unicast e multicast usando diversamente le API di AllJoyn: questa modifica porterebbe automaticamente anche alla riduzione del numero di BusAttachment AllJoyn. A causa dei suoi vantaggi presentati all'inizio del capitolo, AllJoyn risulta comunque essere il framework ideale per sviluppi futuri di A3Droid o per la realizzazione di nuove implementazioni dello stile architetturale A-3.

2 Stato dell'arte

La crescente complessità di sistemi, servizi ed applicazioni richiede che le architetture di sistema e del software siano adattativi in tutti i loro attributi e le loro funzionalità. Nel 2001, IBM ha pubblicato il manifesto di un paradigma denominato "autonomic computing", il cui scopo è fornire un approccio alla realizzazione di sistemi ed applicazioni che possano gestirsi autonomamente in accordo con le linee guida fornite dall'uomo. Questo significa che tali sistemi e tali applicazioni devono adattarsi, senza intervento umano, per soddisfare i loro requisiti di prestazioni, tolleranza ai guasti, affidabilità, sicurezza.

Un esempio di tale adattamento è descritto in [5]: si tratta di un framework basato su ruoli per l'approvvigionamento di risorse per una griglia mobile eterogenea che può essere usata per processare grandi quantità di dati in parallelo. Questo framework può essere usato sia per applicazioni in cui nodi diversi eseguono una stessa operazione su dati diversi, sia per applicazioni in cui nodi diversi eseguono operazioni diverse sugli stessi dati (oppure su dati diversi). La figura 1 mostra l'uso del framework in ambiente sanitario.

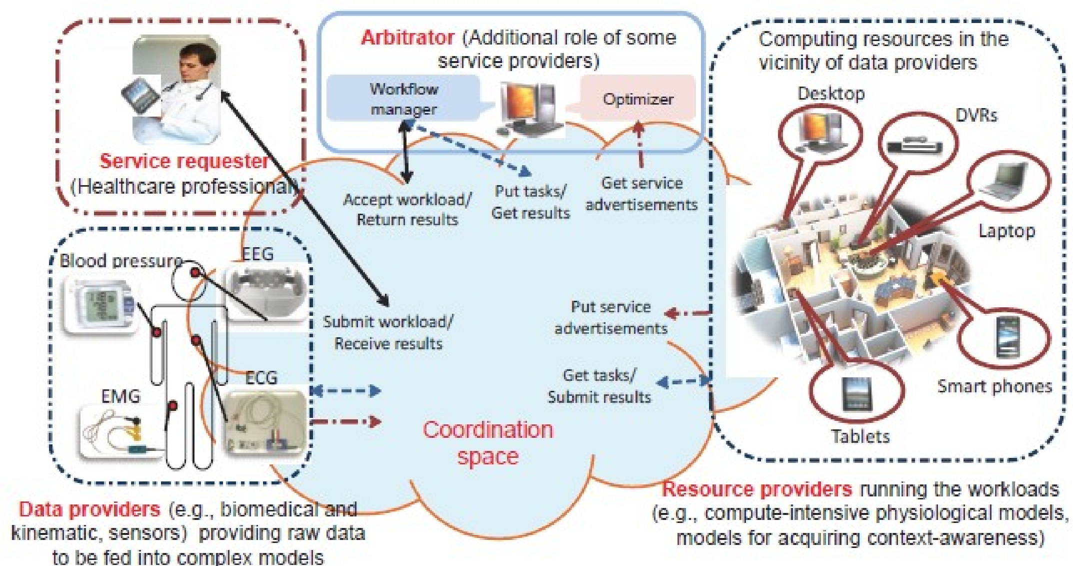

Figura 1 Uso del framework in ambiente sanitario.

Le applicazioni sono composte da workload: modelli matematici pesanti con diversi requisiti di calcolo, memoria e deadline. Questi workload sono composti da task, il cui ordine di esecuzione è specificato da un workflow.

Le entità della griglia possono assumere uno o più tra questi ruoli:

- service requester, che richiede workload ad altri dispositivi;
- service provider, che può fornire dati scalari o multimediali (allora è un "data provider") oppure può fornire risorse di calcolo, memorizzazione e comunicazione per processare dati (ed allora è un "resource provider");
- arbitrator (o "broker"), che processa le richieste dei richiedenti, determina l'insieme dei service provider che forniranno o processeranno i dati e distribuisce ottimamente i task tra essi. Per farlo, si appoggia ad un motore per allocazione delle risorse che massimizza il minimo residuo di carica della batteria su ogni service provider, massimizzando il loro tempo di vita e mantenendo l'eterogeneità dell'insieme di risorse per tempi più lunghi.

Ogni arbitrator è composto da due elementi:

- workload manager (o master), che traccia le richieste di workload, alloca i task ai service provider ed aggrega i risultati;
- scheduler/optimizer, che identifica il numero di service provider disponibili, determina la distribuzione ottimale dei task e distribuisce i dati forniti dai data provider ai service provider, basandosi su delle politiche.

Gli arbitrator scoprono i servizi basandosi sugli advertisement dei service provider. I service provider si auto eleggono arbitrator in base ad un algoritmo distribuito.

I sistemi e le applicazioni realizzate secondo il paradigma dell'autonomic computing possiedono otto proprietà:

- autocoscienza, cioè conoscenza del proprio stato e dei propri comportamenti;
- autoconfigurazione, cioè capacità di cambiare la propria configurazione nelle più varie ed imprevedibili condizioni, rispettando le regole di alto livello fornite da un sistema di politiche, che indicano cosa è desiderato, ma non

come ottenerlo;

- auto ottimizzazione, cioè individuazione dei comportamenti non ottimi e loro correzione, ma anche ricerca autonoma degli ultimi aggiornamenti disponibili ed impostazione automatica dei propri parametri;
- auto guarigione, cioè determinazione e risoluzione dei propri problemi, mantenendo nel frattempo un funzionamento più lineare possibile;
- auto protezione, cioè individuazione di attacchi interni ed esterni, di fallimenti a cascata, di input scorretti da parte dei sensori, e mantenimento della sicurezza e dell'integrità del sistema;
- portabilità, che deve essere garantita dall'uso di interfacce e protocolli standard;
- anticipazione, cioè capacità di prevedere in anticipo le necessità ed i comportamenti propri e del contesto.

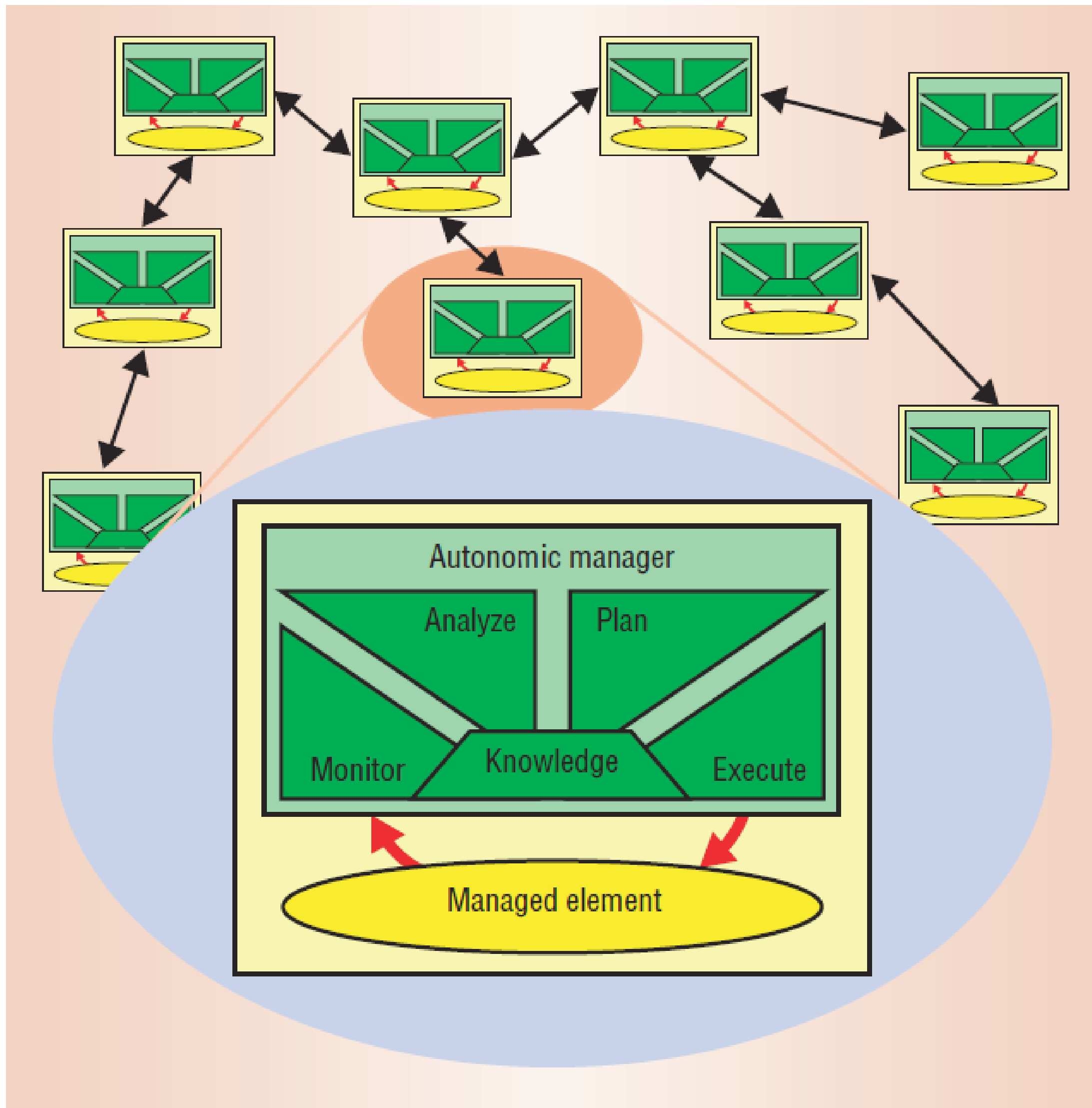

Figura 2 Struttura di un sistema autonomico.

Come si vede in figura 2, i sistemi autonomici e le applicazioni autonomiche sono composizioni dinamiche, opportunistiche o effimere, di elementi autonomici, ciascuno dei quali è composto da tre moduli:

- unità funzionale (o "elemento gestito"): è la più piccola unità funzionale dell'applicazione e contiene il codice sorgente. Esporta le sue interfacce funzionali, i suoi attributi e vincoli funzionali e comportamentali ed i suoi meccanismi di controllo.
- ambiente: tutti i fattori, interni ed esterni, che possono impattare sull'elemento gestito. L'ambiente interno consiste in cambiamenti all'interno dell'elemento gestito, che riflette lo stato dell'applicazione o del sistema,

mentre l'ambiente esterno riflette lo stato dell'ambiente di esecuzione. L'ambiente e l'elemento gestito possono essere visti come due sottosistemi che formano un sistema stabile. Ogni cambiamento nell'ambiente che porta il sistema in uno stato instabile è bilanciato da reazioni dell'elemento gestito che portano il sistema in uno stato stabile differente.

- unità di gestione e controllo: accetta i requisiti specificati dall'utente (prestazioni, tolleranza ai guasti, sicurezza, eccetera), interroga l'elemento gestito per caratterizzarne lo stato, determina lo stato dell'intera applicazione o dell'intero sistema, determina lo stato dell'ambiente ed usa queste informazioni per controllare le operazioni dell'elemento gestito, al fine di attuare i comportamenti specificati. Questo processo di controllo si ripete continuamente durante tutto il tempo di funzionamento dell'elemento autonomico.

Rappresentativo di questa architettura è il framework Kinesthetics eXtreme [8], che nasce con l'obiettivo di dotare sistemi legacy delle proprietà dell'autonomic computing, senza necessità di comprendere cosa faccia il loro codice e senza ricompilarlo. Il modello di controllo è illustrato in figura 3 e si compone delle seguenti entità:

- sistema target, cioè l'applicazione, le applicazioni o i loro componenti, che devono essere monitorati;
- probe, cioè codici installati intorno al sistema target che possono iniettarvi codice sorgente, modificare file binari o bytecode, sostituire librerie, ispezionare il traffico di rete, o fare altro per recuperare queste informazioni;
- gauge, che interpretano i dati provenienti dai probe (o da altri gauge se inseriti in una gerarchia) e generano eventi riguardo al comportamento dell'applicazione;
- attuatori, che riconfigurano o riparano il sistema;
- controllori, che ricevono le analisi effettuate dai gauge e decidono se e quando coordinare uno o più attuatori per tentare una riparazione.

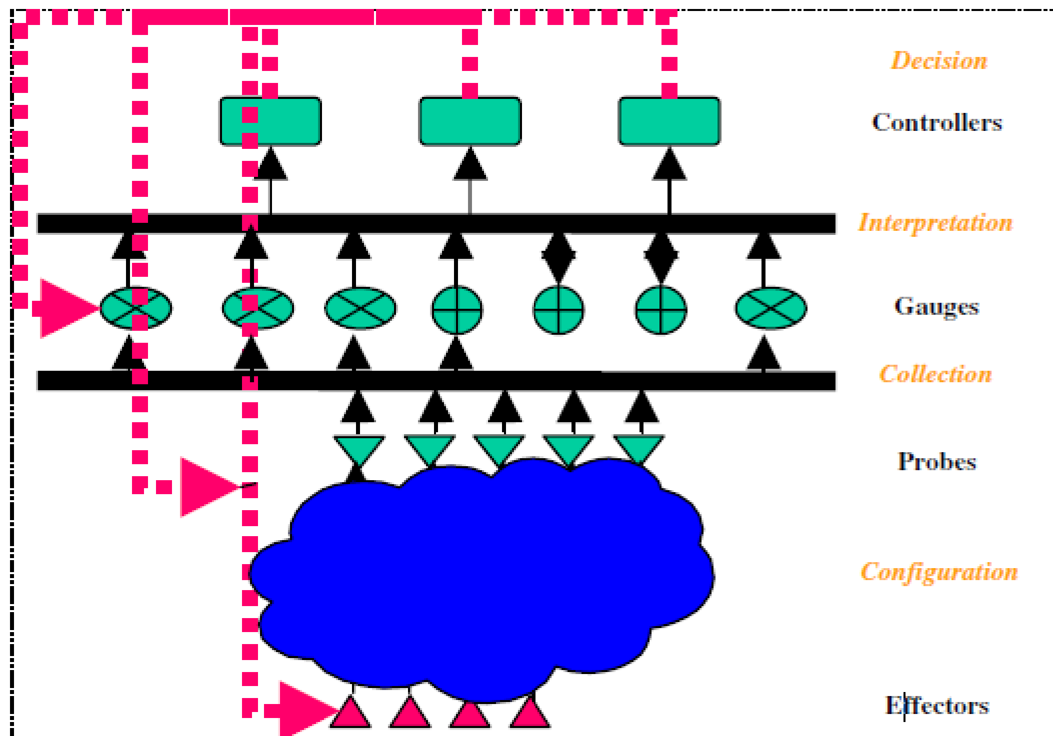

Figura 3 Infrastruttura di controllo del framework Kinesthetics eXtreme.

Tali entità sono basate su eventi e comunicano tra loro scambiandosi messaggi asincroni tramite un middleware ad eventi standardizzato.

Un approccio invece espressamente incentrato sui requisiti dell'utente è il framework "Aura" presentato in [4]. Aura è uno stile architetturale per applicazioni mobile di ubiquitous computing, cioè per quelle applicazioni che consentono agli utenti di continuare ad eseguire i loro task indipendentemente dall'ambiente in cui si trovano. Per "ambiente" si intende l'insieme dei dispositivi e delle applicazioni accessibili da un utente in un dato luogo. Per "task" si intende, invece, una coalizione di servizi astratti che contiene le intenzioni e le preferenze dell'utente. Esempi di task possono essere scrivere un articolo, preparare una presentazione o comprare qualcosa. I servizi astratti sono descritti in un linguaggio basato su XML e possono essere implementati da diverse applicazioni in diversi ambienti. Per esempio il servizio "text editing" può essere implementato da applicazioni come Emacs, Microsoft Word e Notepad. Il framework si basa sul concetto di "Aura personale" dell'utente mobile: un proxy che recupera le risorse appropriate per eseguire il task dell'utente. Questo

avviene ogniqualvolta l'utente entra in un nuovo ambiente. Nell'eseguire questa operazione, il sistema tiene conto dei vincoli imposti dal contesto fisico in cui l'utente si trova, dei requisiti e della qualità del servizio richiesti dall'utente. Rappresentando i task come un insieme di servizi, l'infrastruttura Aura può capire quando tutti i servizi sono istanziabili e quando no. Di conseguenza può cercare ambienti eterogenei che forniscano i servizi del task rispettando i requisiti e la qualità del servizio richiesti dall'utente. Qualora essi non fossero (più) rispettati, il sistema è in grado di cercare configurazioni alternative.

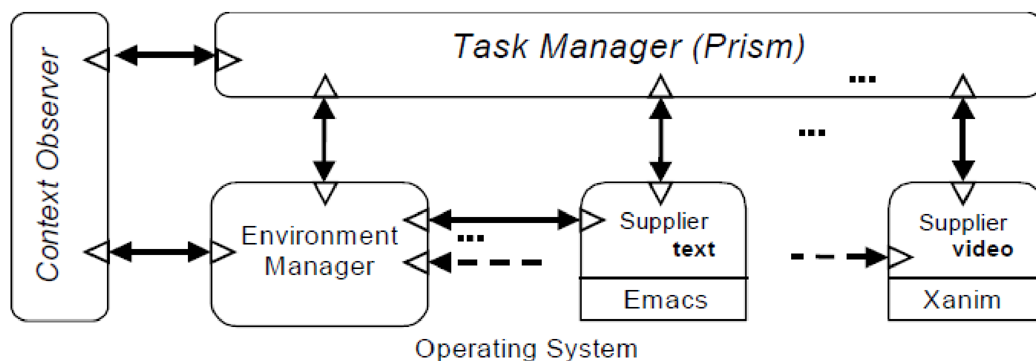

Figura 4 Architettura del framework Aura.

Come si vede in figura 4, il framework Aura è costituito dai quattro tipi di componenti:

- Task Manager (detto "Prism"), che implementa il concetto di "Aura personale", gestisce in modo trasparente all'utente i cambi di ambiente, di task e di contesto e realizza la migrazione di un task da un ambiente ad un altro, salvandone lo stato;
- Context Observer, che fornisce informazioni sul contesto fisico (ad esempio posizione, autenticazione, attività, altre persone nelle vicinanze) e notifica eventi importanti;
- Supplier, che forniscono i servizi astratti di cui i task sono composti e che sono implementati, in pratica, adattando applicazioni e servizi esistenti alle API di Aura;
- Environment Manager, che sa quali Supplier sono disponibili, quali servizi forniscono e dove si trovano. Concettualmente, ogni ambiente ha una sola

istanza di Task Manager, Context Observer e Environment Manager, ma in pratica potrebbero essercene di più per garantire una maggiore robustezza del sistema.

Il framework MobileSpaces [9] offre un approccio ancora diverso, basato sui mobile agent. I mobile agent sono programmi autonomi che possono viaggiare tra computer e computer sotto il loro proprio controllo ed offrono un valido framework per implementare applicazioni distribuite, anche mobile. MobileSpaces è un framework per costruire mobile agent, che, come illustrato in figura 5, si compone di due parti:

- un micro-kernel (o core system), che è visto come un mobile agent statico e che offre le funzioni di gestione della gerarchia degli agenti, della loro esecuzione, dei loro cicli di vita e di loro serializzazione e deserializzazione;
- un insieme di sottocomponenti, anche loro mobile agent, che realizzano la migrazione degli agenti e la memorizzazione dello stato degli agenti su disco.

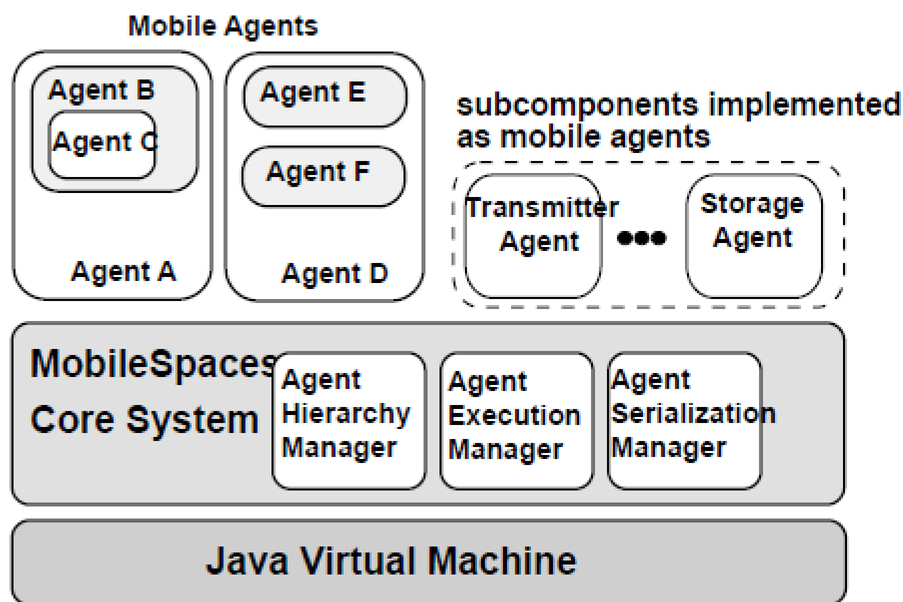

Figura 5 Architettura del framework MobileSpaces.

Ogni mobile agent è fornito come struttura ad albero, in cui ogni nodo contiene un mobile agent ed i suoi attributi. La radice di tale albero è il mobile agent stesso, mentre i nodi figli sono i mobile agent direttamente contenuti all'interno del mobile agent. L'intero sistema può essere visto come un albero la cui radice è il core system.

Ogni agente ha il controllo diretto dei suoi discendenti e può dunque serializzarli, distruggerli, inviarli presso un altro agente ed invocare i loro metodi direttamente. Ogni agente ha un insieme di metodi di servizio a cui i figli possono accedere solo sotto il controllo dell'agente stesso. Quando un agente si sposta da un computer ad un altro, lo fa con tutti i suoi discendenti al suo interno.

Nel paradigma dell'autonomic computing, la parte di controllo si compone di due cicli:

- un ciclo locale, che si basa sulla conoscenza posseduta dall'elemento, può manipolare solo stati noti dell'ambiente, non è a conoscenza del comportamento dell'intera applicazione o dell'intero sistema, e dunque non può raggiungere gli obiettivi globali.
- un ciclo globale, che può gestire stati del sistema sconosciuti, basandosi su tecniche di machine learning ed intelligenza artificiale e sull'intervento umano. Controlla ed analizza prestazioni, configurazioni, protezioni e sicurezza degli elementi. E costituisce una nuova conoscenza all'interno dell'elemento gestito, che abilita quest'ultimo all'adattamento dei suoi comportamenti per rispondere ai cambiamenti dell'ambiente.

Tipicamente, il controllo del sistema è basato su politiche. Questo approccio è particolarmente adatto per realizzare sistemi pervasivi autonomi perché costituiscono una tecnica semplice, dinamica e flessibile per implementare adattamento e controllo con feed-back. Un esempio è Ponder2 [11], che comprende un sistema di gestione di oggetti general-purpose, scambio di messaggi tra oggetti, eventi, politiche ed un motore di esecuzione delle politiche stesse. La struttura di Ponder 2 è illustrata in figura 6. In Ponder2, qualsiasi cosa è un Managed Object che include gestione delle politiche, sensori, allarmi, interruttori, ed adattamenti ad oggetti del mondo reale. Il sistema manipola tutti gli oggetti allo stesso modo. Tutti gli oggetti devono essere caricati automaticamente creando un factory managed object, a cui possono essere spediti messaggi per creare nuove istanze del Managed Object desiderato, che poi svolgeranno il lavoro del sistema.

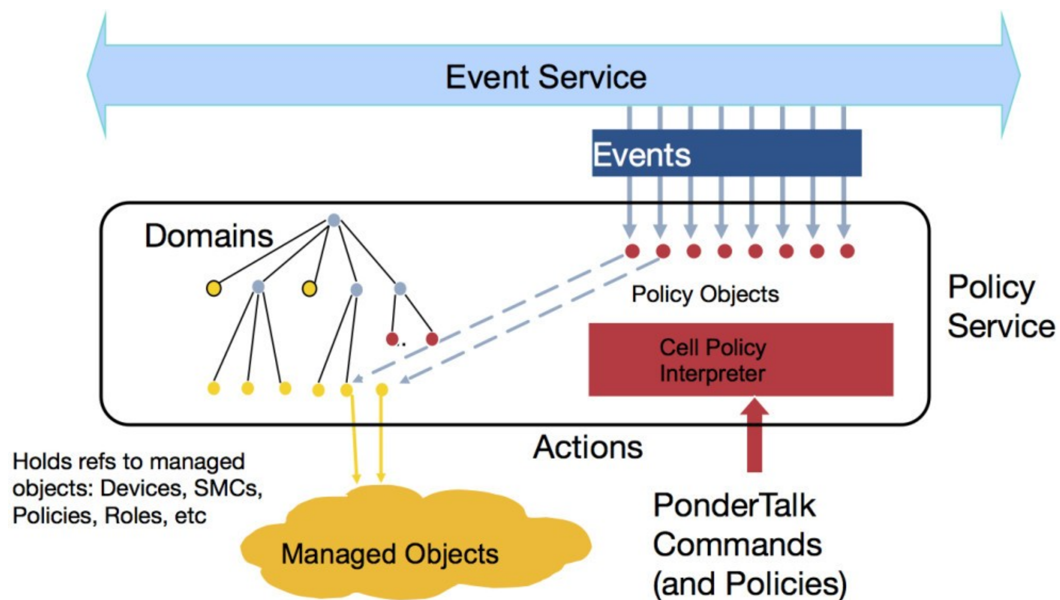

Figura 6 Architettura del framework Ponder2.

I tipi base di oggetti forniti da Ponder2 sono: politiche (Policies), eventi (Events) e domini (Domains).

Le politiche definiscono gli obiettivi della gestione del sistema e sono di due tipi:

- Obligation Policies, che definiscono le azioni che i gestori devono compiere quando si verificano certi eventi e forniscono al sistema la capacità di rispondere ai cambiamenti.
- Authorization Policies, che consentono od impediscono scambio di messaggi tra oggetti per proteggere le risorse ed i servizi da accessi indesiderati. Possono essere positive, e dunque definire le azioni che i chiamanti possono compiere sui chiamati, oppure negative, e dunque definire le azioni che i chiamanti non possono svolgere sui chiamati.

Gli eventi sono messaggi che contengono i valori da usare nella valutazione delle condizioni e nell'esecuzione delle azioni. I domini sono contenitori di Managed Object e sono strutturati come un file system gerarchico. Questa struttura gerarchica consente di offrire un evento solo alle politiche che lo richiedono ed è così possibile anche creare politiche che ricevano eventi solo da un insieme limitato di Managed

Object. Questo meccanismo è chiamato Domain Event Bus e, quando applicato a sistemi distribuiti e pervasivi, permette di far fronte agli eventi locali tramite politiche locali, pur avendo nel frattempo politiche di più alto livello che reagiscono allo stesso evento da più parti del sistema.

Tuttavia, un sistema di autogestione basato su politiche, per quanto ben costruito, non basta da solo a garantire l'affidabilità del sistema: i sensori potrebbero infatti fornire dei valori fuorvianti, che porterebbero il sistema a prendere decisioni che, sebbene legali, lo metterebbero in una condizione di instabilità o di inconsistenza. Un sistema è veramente affidabile quando le sue azioni sono continuamente validate a runtime per soddisfare i requisiti ed i risultati prodotti sono attendibili e non fuorvianti. Auto validazione, affidabilità ed attendibilità del sistema non possono essere correttamente aggiunte a sistemi già esistenti, dunque è necessario integrarle a livello architetturale [3].

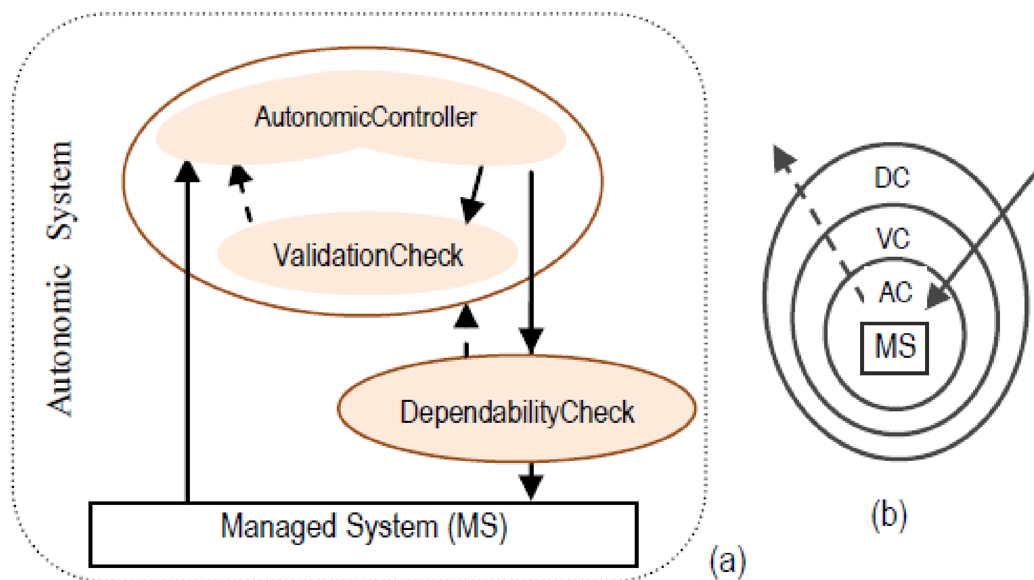

Figura 7 Architettura autonomica affidabile.

L'architettura proposta in [3] ed illustrata in figura 7 (a), sebbene diversa dalla soluzione tradizionale fornita dal paradigma dell'autonomic computing, è incentrata su un componente costruito proprio su tale soluzione, denominato

“AutonomicController”: è basato su logiche di controllo come MAPE o Intelligent Machine Design framework (IMD), e monitora il sistema gestito per ricavare informazioni sul contesto, in base alle quali decide quali azioni intraprendere. Prima di essere eseguite, tali azioni sono passate ad un altro componente denominato “ValidationCheck”: controlla che tali azioni siano conformi all’obiettivo descritto dalle politiche, cioè che tale obiettivo sia raggiunto solo seguendo le regole specificate. Inoltre mitiga le anomalie comportamentali e strutturali del sistema, come contraddizione tra politiche, distorsione degli obiettivi, strutture ed operazioni illegali, come una divisione per zero, e così via. Se la validazione fallisce, il ValidationCheck notifica l’AutonomicController, altrimenti chiama un componente denominato “DependabilityCheck”. Esso si assicura che una decisione non porti all’instabilità o ad un comportamento inconsistente del sistema, consentendo all’AutonomicController di cambiare decisione solo quando è necessario e sicuro farlo. Se la decisione supera anche il controllo del DependabilityCheck, allora è passata agli attuatori per essere eseguita, altrimenti è notificato l’AutonomicController. La figura 7 (b) mostra che l’architettura può essere considerata un ciclo di controllo annidato, dove il ciclo più interno è l’AutonomicController, quello intermedio è il ValidationCheck e quello più esterno è il DependabilityCheck.

Il paradigma dell’autonomic computing richiede che sia effettuata ricerca ad ogni livello. In particolare è necessario:

- definire astrazioni e modelli appropriati per specificare, capire, controllare ed implementare comportamenti autonomici;
- adattare a sistemi dinamici e multi agente le classiche teorie ed i classici modelli per il machine learning, l’ottimizzazione ed il controllo;
- fornire modelli per la negoziazione che gli elementi possano usare per stabilire relazioni tra loro;
- progettare modelli statistici di sistemi, affinché gli elementi possano rilevare o predire i problemi globali da un flusso di dati provenienti da sensori di singoli dispositivi;
- definire architetture software e di sistema in cui i comportamenti autonomici

possano essere specificati, implementati e controllati in maniera robusta e prevedibile;

- realizzare middleware sicuri, affidabili, robusti e scalabili che forniscano i servizi richiesti per realizzare i comportamenti autonomici in maniera altrettanto robusta ed affidabile;
- fornire modelli di programmazione, strutture e middleware che supportino la definizione di elementi autonomici, lo sviluppo di applicazioni autonomiche come composizione dinamica ed opportunistica di elementi autonomici, la gestione e l'esecuzione, basate su politiche, capacità e contesto, di tali applicazioni. Un esempio di piattaforma che offre queste funzionalità è GridLite [13].

GridLite è una piattaforma per griglia equipagiata con supporto per la progettazione, l'implementazione, la distribuzione e l'evoluzione di applicazioni per gli ambienti cosiddetti "DREAM" (Decentralized, Resource-constrained, Embedded, Autonomic, Mobile). E' strutturata su tre livelli, di seguito elencati in ordine crescente di astrazione:

- middleware (GLM), che astrae diversi protocolli di comunicazione e diversi sistemi operativi, supporta diversi linguaggi di programmazione ed in futuro dovrebbe anche supportare applicazioni leggere e fornire compatibilità con altre griglie;
- services (GLS), che offre servizi di discovery delle risorse nella griglia, controllo delle prestazioni della griglia e dei suoi nodi, accesso alle risorse in condizioni eterogenee, trasformazione dinamica delle architetture di sistema, installazione dei componenti della griglia su piattaforme di esecuzione diverse;
- architectural support (GLAS), che dovrebbe abilitare l'espressione di architetture di applicazione mediante primitive direttamente implementate nel livello di middleware.

L'architettura di GridLite è illustrata in figura 8 a pagina seguente.

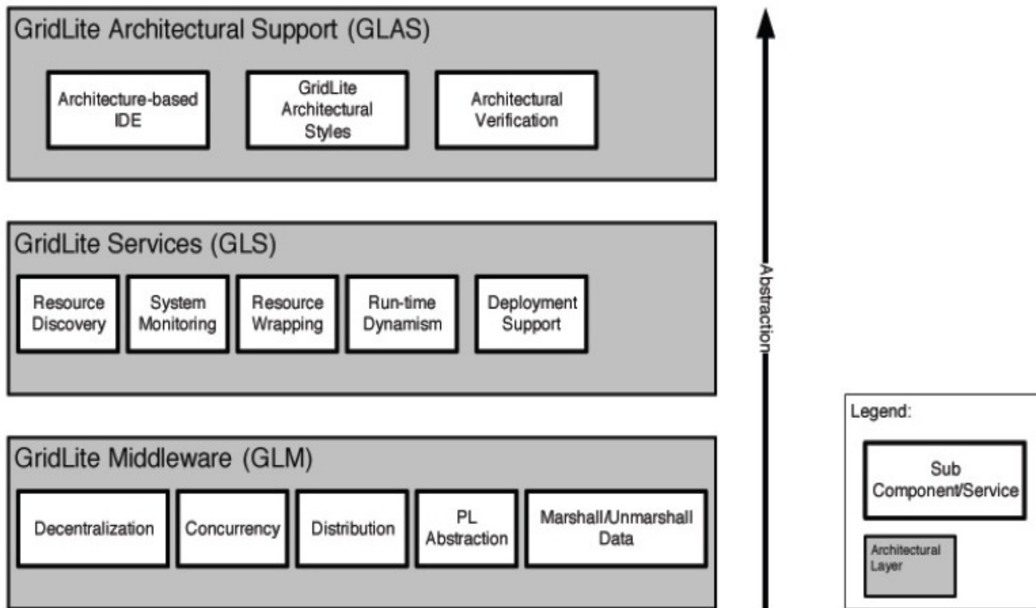

Figura 8 Architettura di GridLite.

3 AllJoyn

AllJoyn è un framework open source sviluppato da AllSeen Alliance: un gruppo di più di cento aziende che punta allo sviluppo, all'evoluzione ed alla diffusione su larga scala di un framework basato su AllJoyn per la connettività e la comunicazione tra dispositivi ed applicazioni nella Internet of Everything. AllJoyn incorpora più di venti tra i più importanti standard aperti ed è già usato in prodotti commerciali [2].

Al centro della progettazione di AllJoyn stanno sempre i concetti di prossimità, mobilità e configurazione dinamica: in un ambiente mobile, i dispositivi entrano continuamente nelle vicinanze di altri dispositivi, continuamente ne escono, e le capacità della rete sottostante può cambiare altrettanto in continuazione. AllJoyn permette alle applicazioni di parlare tra loro quando dispositivi che usano una stessa rete sono vicini e fornisce un approccio orientato agli oggetti per facilitare il peer-to-peer, evitando agli sviluppatori la necessità di fronteggiare i problemi che emergono in sistemi distribuiti eterogenei. Infatti:

- AllJoyn è sviluppato per girare su Microsoft Windows, Linux, Android, iOS, OS X, OpenWRT, e come Unity plug-in per l'ecosistema Unity di sviluppo di giochi;
- Il core di AllJoyn è scritto in C++, ma è possibile scrivere applicazioni AllJoyn, oltre che in C++, anche nei linguaggi Java, C#, JavaScript e Objective-C;
- Le tecnologie di rete gestite da AllJoyn, e da esso astratte come bus, sono Bluetooth, ICE, LAN, TCP, Wi-Fi Direct, wireless LAN, wireless WAN.

3.1 Architettura

Come si vede in figura 9 a pagina seguente, la struttura di AllJoyn comprende applicazioni e router (o demoni):

- i router parlano tra loro e con le applicazioni;
- le applicazioni parlano tra loro tramite i router.

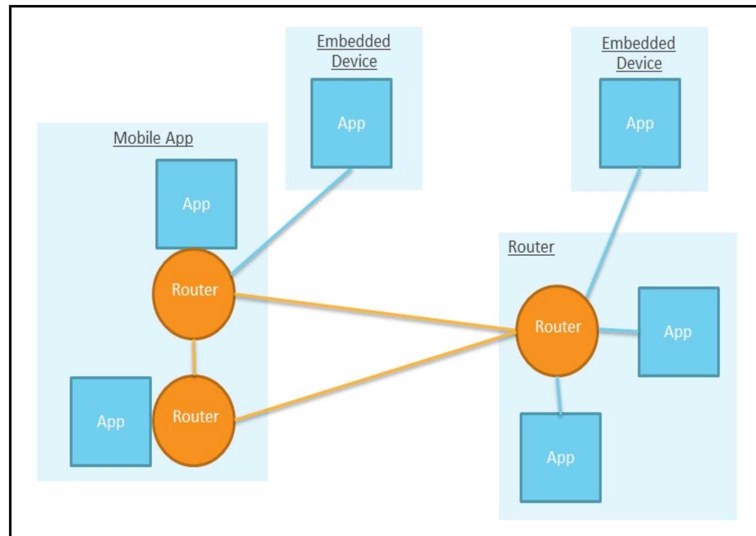

Figura 9 Applicazioni, router e loro topologie comuni.

Applicazioni e router possono trovarsi indifferentemente sullo stesso dispositivo o su dispositivi diversi. Sono possibili tre topologie:

- un'applicazione usa il proprio router, detto "bundled router" (tipico di applicazioni per Android, iOS, Mac OS X e Windows);
- più applicazioni sullo stesso dispositivo usano uno stesso router, che tipicamente gira in background ed è detto "stand alone router" (tipico di applicazioni per Linux);
- un'applicazione usa un router su un diverso dispositivo (tipico dei sistemi embedded, che non hanno abbastanza CPU e memoria per eseguire il router).

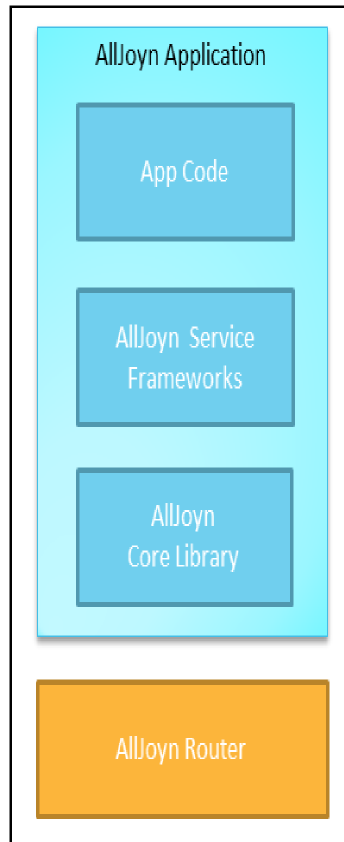

Figura 10 Struttura di un'applicazione AllJoyn.

Come si vede in figura 10, un’applicazione AllJoyn comprende:

- codice applicativo, che può essere programmato usando sia le librerie dell’AllJoyn Service Frameworks, sia la libreria Core di AllJoyn;
- librerie dell’AllJoyn Service Frameworks, che implementano un insieme di servizi comuni e che possono anche essere usate per sviluppare un’applicazione;
- libreria Core di AllJoyn, che fornisce le API di livello più basso per accedere alla rete AllJoyn, dando accesso diretto a adveritisement, discovery, creazione delle sessioni, interfaccia di definizione di metodi e segnali, creazione e manipolazione di oggetti. Possono essere usate sia per implementare l’AllJoyn Service Framewors, sia per implementare altre applicazioni.

Figura 11 Versioni standard e thin di AllJoyn.

Come si vede in figura 11, esistono due versioni di AllJoyn, perfettamente compatibili tra loro:

- standard, per sistemi non embedded;
- thin, per sistemi embedded con memoria limitata.

Il router fornisce una serie di funzionalità per abilitare le caratteristiche chiave di AllJoyn sopra diversi tipi di trasporto. Tali funzionalità sono advertisement e discovery di applicazioni, sessioni, scambio di dati, segnali sessionless e sicurezza. I segnali ed i dati provenienti dal livello applicativo sono incapsulati dal livello "Message and Signal Transport" per aderire al formato D_Bus. I tipi di trasporto ed i sistemi operativi sono astratti da due appositi livelli. Le funzionalità del router sono illustrate a pagina seguente in figura 12.

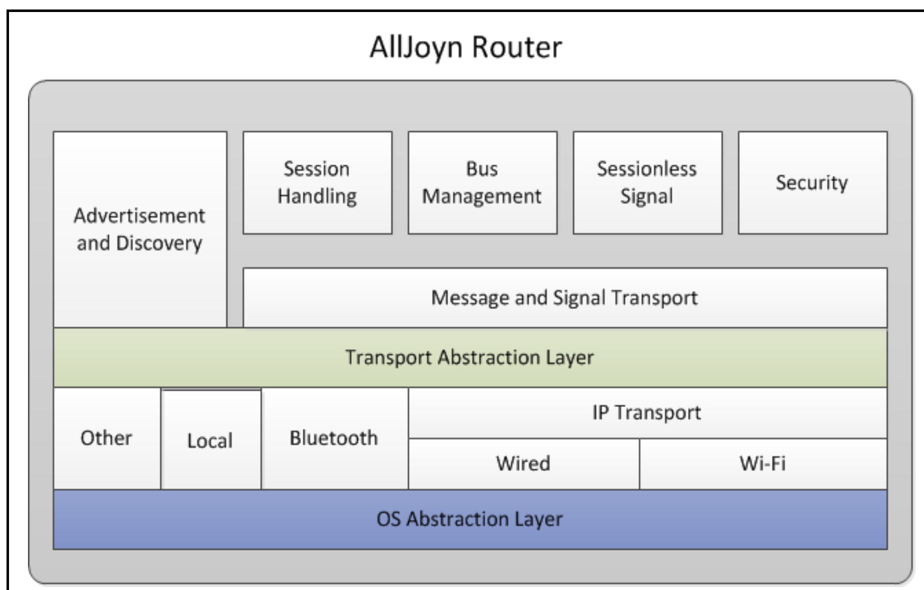

Figura 12 Struttura funzionale del router AllJoyn.

L'astrazione più fondamentale di AllJoyn è il bus, che consente alle applicazioni di spedire messaggi nel sistema distribuito, senza avere a che fare con i dettagli di basso livello. Come si vede in figura 13, il bus distribuito si estende tipicamente tra più dispositivi.

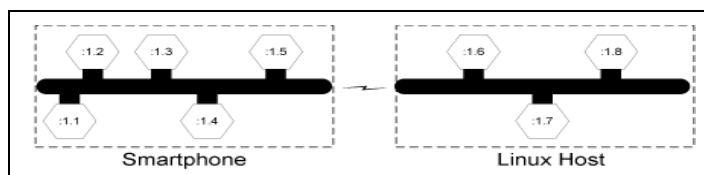

Figura 13 Comunicazione tra dispositivi in AllJoyn.

Un collegamento di comunicazione tra segmenti di bus distribuito che si trovano su dispositivi diversi è creato quando richiesto dai componenti. Il router gestisce tale collegamento e rende trasparente all'utente del bus la presenza di più dispositivi che lo ospitano. Un componente vede dunque sempre l'intero bus come se fosse locale.

3.2 BusAttachment

Ogni applicazione AllJoyn richiede un oggetto detto “BusAttachment” per collegarsi al bus. Il BusAttchment è l’interfaccia tra un’applicazione e le altre, che gestisce la rete di basso livello e rappresenta il bus distribuito alle applicazioni. Infatti, siccome ogni singola applicazione ed il router sono eseguiti in processi separati, deve esserci una rappresentanza del router in ogni processo. Ad ogni BusAttachment è assegnato un identificativo univoco detto “unique name”, che è generato automaticamente dal sistema quando un’applicazione si connette al bus. Le applicazioni usano un BusAttachment per compiere le seguenti operazioni: advertising, discovery e comunicazione con le altre applicazioni.

3.3 Advertising

Scopo dell’operazione di advertising è fornire informazioni riguardo all’applicazione e determinare a quale applicazione connettersi. Affinché un’applicazione possa connettersi e comunicare con un’altra applicazione, AllJoyn fornisce alle applicazioni un nome detto “well-known name”: un alias dello unique name che corrisponde al nome dell’applicazione. L’introduzione dei well-known name è necessaria perché lo unique name è assegnato da AllJoyn ogniqualvolta un’applicazione si connette al bus e non è dunque utilizzabile come identificatore persistente di un servizio. Il well-known name è definito dallo sviluppatore, è leggibile ed è molto importante perché fa capire la natura dell’applicazione alle altre applicazioni. Il well-known name è una stringa di lunghezza inferiore a 255 caratteri che deve avere il seguente formato:

- può contenere caratteri alfanumerici (a-z A-Z 0-9);
- può contenere i simboli “.” e “_”;
- deve contenere almeno un simbolo “.”;
- un numero non può mai seguire un simbolo “.”;
- non può contenere due simboli “.” consecutivi.

E' importante che il well-known name di ogni applicazione sia univoco: se più applicazioni usassero lo stesso well-known name, un'applicazione in fase di discovery non sarebbe in grado di determinare quante altre applicazioni le sarebbero vicine. Se facesse un tentativo di connessione, si connetterebbe ad un'applicazione a caso, che potrebbe non avere il comportamento desiderato. Se il well-known name fosse costruito solo con l'aggiunta di un input proveniente dall'utente, non ne sarebbe garantita l'unicità. La pratica raccomandata è quella di usare un prefisso con l'aggiunta di un GUID (Globally Unique ID) ed opzionalmente un valore inserito dall'utente. Il GUID è tipicamente recuperato invocando il metodo "getGlobalGUIDString" del BusAttachment. Per usare un well-known name, un'applicazione deve fare una richiesta al router: se il nome non è già in uso da parte di un'altra applicazione, allora all'applicazione ne è garantito l'uso esclusivo. In questo modo è garantito che il well-known name rappresenta un indirizzo univoco sul bus.

Non appena i dispositivi sono tra loro vicini, il sistema trasmette e riceve i nomi pubblicati sul bus attraverso i tipi di trasporto disponibili, notificando ad ogni dispositivo la disponibilità dei servizi corrispondenti. Per effettuare l'operazione di advertising in un'applicazione è necessario:

- creare un BusAttachment;
- invocare il metodo "RequestName" per fornire al framework il well-known name dell'applicazione;
- invocare il metodo "AdvertiseName" per notificare ogni dispositivo vicino in stato di discovery che l'applicazione esiste.

3.4 Discovery

Scopo dell'operazione di discovery è rilevare la presenza di altre applicazioni AllJoyn nelle vicinanze. Questo è fatto basandosi sul prefisso del well-known name, che è usato per identificare le applicazioni. AllJoyn notifica l'applicazione ogni volta

che scopre un nome che comincia con il prefisso fornito. AllJoyn fornisce una classe BusListener che contiene un insieme di metodi che possono essere implementati nell'applicazione. Tramite essa è possibile sapere quale applicazione è stata trovata, quale non si trova più nelle vicinanze ed il trasporto di comunicazione. Per effettuare l'operazione di discovery è necessario:

- estendere la classe BusListener implementando i metodi di quest'ultima;
- creare un'istanza della classe appena creata;
- registrarla nel BusAttachment;
- invocare il metodo "findAdvertiseName" del BusAttachment passando come parametro una stringa corrispondente al prefisso da cercare.

Se è trovato un nome di un'applicazione che comincia con il prefisso specificato, è chiamato il metodo "foundAdvertisedName" del BusListener registrato. Quando un'applicazione che sta facendo advertising cade, perde connettività o non è più nelle vicinanze, parte un timeout di 90 secondi dopo il quale, se l'applicazione non è tornata ad essere di nuovo disponibile, è chiamato il metodo "lostAdvertisedName" per il nome di tale applicazione. Il timeout non parte se l'applicazione termina volontariamente l'operazione di advertising chiamando il metodo "CancelAdvertiseName".

In un sistema peer-to-peer le operazioni di advertising e discovery sono effettuate da ogni applicazione. Ogni applicazione usa un solo BusAttachment per eseguirle entrambe. Nel dettaglio:

- il processo di advertising crea il BusAttachment e chiama i metodi "RequestName" e "AdvertiseName";
- il processo di discovery crea un oggetto BusListener che è registrato nel BusAttachment, ed invoca il metodo "findAdvertisedName". Se un'applicazione nelle vicinanze è scoperta, è invocato il metodo "foundAdvertisedName" per raccogliere le informazioni utili ad identificare tale applicazione.

3.5 Sessioni

Tramite le operazioni di advertising e di discovery, le applicazioni possono riunirsi in gruppi denominati “sessioni”, che permettono loro di comunicare e scambiarsi dati in maniera efficiente. Il concetto di sessione è introdotto perché AllJoyn utilizza ed astrae diverse tecnologie di rete, che in alcuni casi richiedono lavoro aggiuntivo a basso livello per creare un meccanismo di comunicazione. Un’applicazione che fornisce un servizio (provider) crea una sessione ed aspetta che un’altra applicazione (consumer) si unisca ad essa. L’applicazione che crea la sessione è il session owner, le altre applicazioni sono dette “joiner”. Siccome un’applicazione può appartenere anche a più sessioni contemporaneamente, è necessario identificare un session owner mediante un “session port number”, che è un numero che ha valore solo all’interno di un BusAttachment e non ha a che fare in nessun modo con le porte di rete. Tale numero è tipicamente definito dallo sviluppatore e deve essere noto ai joiner affinché possano creare una sessione con il session owner. Lato provider, l’applicazione occupa una porta con la libreria core AllJoyn, specificando una lista di opzioni per la sessione (trasporto, tipo di sessione, eccetera) e si mette in ascolto di consumer che si vogliono unire alla sessione. Il terminale è identificato univocamente con la combinazione di well-known name (o unique name) e session port. Lato consumer, l’applicazione richiede al bus di unirsi alla sessione con una data applicazione provider, specificando la porta, il nome del servizio (well-known name o unique name) e le opzioni della sessione. Il router AllJoyn lato consumer crea una connessione fisica con il provider, basandosi sulle informazioni ricavate in fase di discovery. Attualmente, questo consiste nel creare una connessione TCP attraverso Wi-Fi. Dopodiché, il router AllJoyn inizia l’instaurazione della sessione tra le due applicazioni. Dopo che il primo consumer si è unito alla sessione, il provider assegna ad essa un session ID univoco e crea una mappa in cui sono memorizzate le informazioni rilevanti per la seessione. Il session ID è poi spedito al consumer, il quale lo utilizzerà in seguito per la comunicazione con il provider. Anche il consumer crea una mappa per memorizzare le informazioni rilevanti riguardanti la sessione.

Le sessioni possono essere singlepoint (tra il session owner ed un solo joiner) o multipoint (tra il session owner e più joiner). Se cade il session owner di una sessione singlepoint, tale sessione non esiste più. Se invece cade il session owner di una sessione multipoint, tale sessione continua ad esistere, ma nessun altro joiner può più unirvisi, perché il corrispondente well-known name non è più presente sul bus e non può più essere scoperto.

Per creare una sessione è necessario:

- creare un BusAttachment;
- registrare i BusObject;
- creare un oggetto SessionOpts che definisce i mezzi di comunicazione consentiti, come i trasporti di comunicazione supportati e la prossimità;
- passare l'oggetto SessionOpts come parametro del metodo "bindSessionPort" del BusAttachment.

Un altro parametro del metodo "bindSessionPort" del BusAttachment è un oggetto SessionPortListener, che contiene metodi di callback che permettono ad un'applicazione di connettersi basandosi sull'oggetto SessionOpts e di essere notificata quando la connessione è stata stabilita. Una volta stabilita la connessione, è chiamato il metodo "sessionJoined" del SessionPortListener, dove l'ID della sessione ed il nome del joiner possono essere salvati per una futura interazione con il joiner.

Prima di creare una sessione, l'applicazione richiede:

- il well-known name del session owner;
- la porta che la sessione usa (solitamente un valore codificato uguale per tutte le applicazioni);
- un SessionListener che sarà registrato una volta che l'applicazione si è unita alla sessione;
- l'oggetto SessionOpts che specifica i trasporti, la prossimità e la natura del multipoint della sessione.

L'unione alla sessione è eseguita chiamando il metodo "joinSession" del

BusAttachment dopo che il metodo “foundAdvertisedName” è stato chiamato per notificare la scoperta del nome cercato. I callback “sessionMemberAdded”, “sessionMemberRemoved” e “sessionLost” notificano rispettivamente quando una nuova applicazione si unisce alla sessione, la lascia, e quando la sessione non è più valida. La figura seguente riassume la procedura di instaurazione di una sessione.

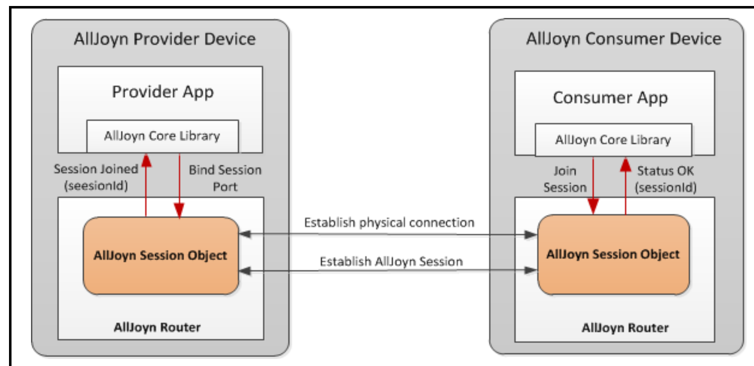

Figura 14 Instaurazione di una sessione AllJoyn.

3.6 BusInterface

La terza operazione che un’applicazione può effettuare usando un BusAttachment è comunicare con le altre applicazioni. La dimensione dei messaggi è limitata a 131072 byte. Affinché le applicazioni comunichino ed interagiscano tra loro in maniera ben definita, AllJoyn richiede la presenza di un’interfaccia detta “BusInterface”. Essa è una collezione di funzioni che hanno una qualche relazione tra loro e serve da contratto tra l’entità che la implementa ed il mondo esterno. Per questo motivo, le interfacce sono oggetto di standardizzazione attraverso strutture specifiche. Tale interfaccia è un insieme di almeno uno tra “BusMethod, “BusProperty” e “BusSignal”.

3.7 BusMethod

Un BusMethod è una chiamata sincrona ad un metodo remoto e funziona come un'interazione 1 a 1. Essendo sincrona, il metodo chiamante si blocca finché quello remoto non termina. Prima di chiamare un BusMethod su un altro dispositivo è necessario creare una sessione. In seguito bisogna creare un ProxyBusObject nell'applicazione chiamante. Un ProxyBusObject è una rappresentazione locale del BusObject che si trova nell'altra applicazione. Implementa la stessa interfaccia dell'oggetto remoto, ma attua i processi di marshalling e serializzazione dei parametri e di spedizione dei dati all'applicazione remota. Uno stub nell'applicazione remota effettua invece l'operazione inversa. Per gestire in maniera trasparente e corretta questo meccanismo, esiste una specifica delle signature dei metodi. Esse sono definite da stringhe di caratteri, le quali possono descrivere stringhe di caratteri, tipi numerici base e comuni a molti linguaggi di programmazione, tipi composti come array e strutture di tipi base. Per eseguire un BusMethod è necessario:

- creare un ProxyBusObject;
- ottenere la BusInterface di cui si vuole eseguire il BusMethod attraverso il ProxyBusObject;
- chiamare il BusMethod usando la BusInterface, come se si stesse chiamando un metodo locale.

L'applicazione remota esegue il codice del metodo e, se non è di tipo "void", ritorna dei valori.

3.8 BusProperty

Una BusProperty è usata in un'applicazione per facilitare la lettura e la scrittura di variabili ed è una scorciatoia per generare e creare un BusMethod volto a leggere o a scrivere una variabile. Il nome del metodo deve rispettare il formato "get<VariableName>" o "set<VariableName>". Non ha nulla di diverso da un

BusMethod, dunque è sincrona, bisogna creare una sessione prima di chiamarla ed è usata ed implementata come un BusMethod.

3.9 BusSignal e sessionless signals

I BusSignal sono usati per spedire dati a più applicazioni contemporaneamente e non c'è risposta. Il BusSignal sarà ricevuto da tutte le applicazioni nella sessione che si sono registrate per riceverlo. La registrazione avviene fornendo ad AllJoyn il nome del metodo che deve essere eseguito alla ricezione del segnale. I BusSignal spediti da una stessa applicazione arrivano a tutte le applicazioni nell'ordine in cui sono stati spediti. Non è invece detto che due BusSignal spediti da due applicazioni diverse siano ricevuti da tutte le applicazioni nello stesso ordine.

Siccome non sempre è conveniente creare una sessione, AllJoyn introduce i "sessionless signal": sono BusSignal con ID della sessione pari a 0, perché l'applicazione non fa parte di una sessione quando lo spedisce, e con un sessionless flag impostato a "true". Sono spediti come un normale BusSignal, ma per riceverli, oltre a registrare l'apposito metodo come per i normali BusSignal, bisogna aggiungere un'apposita regola tramite la funzione "addMatch("sessionless='t')". E' necessario capire le limitazioni seguenti alle prestazioni:

- un sessionless signal non sostituisce l'uso delle sessioni, ma è una scorciatoia per evitare di introdurre esplicitamente logica di creazione di una sessione per trasmettere piccole quantità di dati;
- tutti i sessionless signal hanno un overhead per creare e distruggere una connessione;
- i sessionless signal non devono dunque essere usati per spedire grandi quantità di dati o aggiornamenti frequenti;
- ogni sessionless signal rimpiazza il precedente, dunque un ricevente che non riesce a ricevere un sessionless signal lo perde se nel frattempo ne è stato emesso un altro.

3.10 BusObject e object path

La BusInterface fornisce un modo standard per definire un'interfaccia che funziona nel sistema distribuito. Il BusObject è l'oggetto nel quale deve essere implementata tale interfaccia. I BusObject vivono nel BusAttachment e servono come terminali di comunicazione. Dal momento che in un BusAttachment possono esserci più implementazioni di una stessa interfaccia, deve esistere un'ulteriore struttura per differenziarle, data dall'object path. Il namespace dell'object path è strutturato ad albero, come un albero di directory del file system. Dal momento che i BusObject sono implementazioni di BusInterface, gli object path devono seguire la convenzione del nome della corrispondente interfaccia. Lo stesso vale anche per i well-known name delle applicazioni. La corrispondenza tra well-known name delle applicazioni, nomi delle BusInterface e object path è illustrata in figura 15.

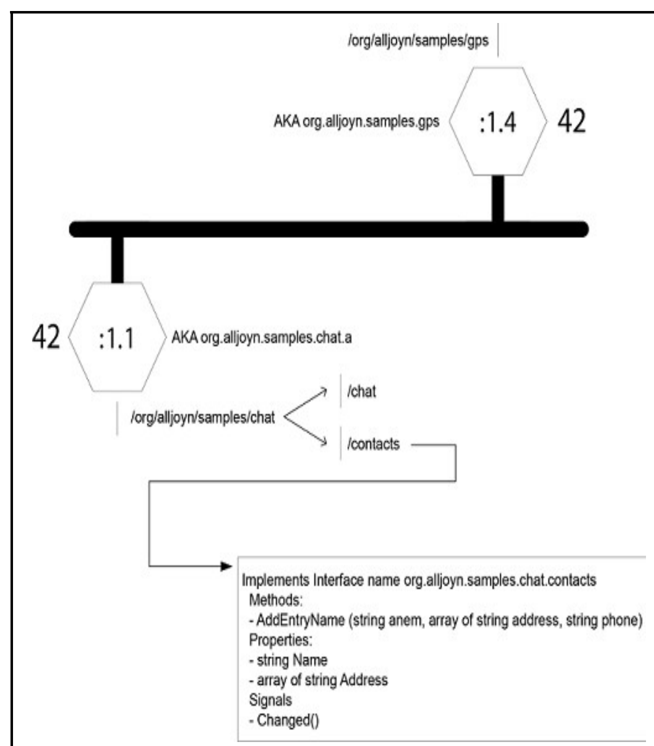

Figura 15 Corrispondenza tra well-known name, nomi delle BusInterface e object path.

Ad esempio, l'esistenza sul bus del well-known name "org.freedesktop.DeviceKit.Disks.sda1" implica l'esistenza di un BusObject di nome

“/org/freedesktop/DeviceKit/Disks/sda1” che implementa la BusInterface di nome “org.freedesktop.DeviceKit.Disks”. In questo modo è possibile risalire al nome dell’oggetto di interesse partendo dal nome scoperto sul bus e risalire alla sua interfaccia, che è quella che l’applicazione deve usare per comunicare con lui. Un well-known name è univoco, dunque l’unico modo per distinguere due applicazioni che implementano la stessa BusInterface è aggiungere un nome univoco come suffisso del nome di tale BusInterface. Per esempio, l’esistenza sul bus dei nomi “org.freedesktop.DeviceKit.Disks.sda1” e “org.freedesktop.DeviceKit.Disks.sda2” implica l’esistenza di due BusObject che implementano la BusInterface “org.freedesktop.DeviceKit.Disks”.

Il BusObject deve essere registrato nel BusAttachment tramite una chiamata al metodo “registerBusObject” del BusAttachment stesso. Tale metodo prende come parametro in ingresso il nome del BusObject. Tale nome è denominato “object path” e, per convenzione, deve rispecchiare la struttura di un file system, così come anche i well-known name delle applicazioni e i nomi delle BusInterface.

3.11 Comportamento di un’applicazione AllJoyn

In conclusione, la figura 16 a pagina seguente illustra il comportamento che un’applicazione deve tenere per poter usufruire di tutte le funzionalità di AllJoyn. Un’applicazione AllJoyn interagisce con il framework AllJoyn attraverso il BusAttachment. L’applicazione pubblicizza i suoi servizi mediante il well-known name. Quando un’applicazione remota scopre un’applicazione AllJoyn, può creare una sessione connettendosi ad una specifica porta. Sono supportate sia sessioni punto-punto sia sessioni multipoint. L’applicazione AllJoyn ha la possibilità di accettare o rifiutare richieste di connessione remota. Prima della creazione della sessione, l’applicazione può creare un numero qualsiasi di BusObject e metterli in uno specifico object path. Ogni BusObject può implementare un insieme di interfacce, definite da un’insieme di metodi, proprietà e segnali. Dopo che la sessione

è creata, l’applicazione remota comunicherà con l’applicazione creando un ProxyBusObject per interagire con il BusObject invocandone i metodi, leggendo e scrivendone i valori degli attributi e ricevendo segnali.

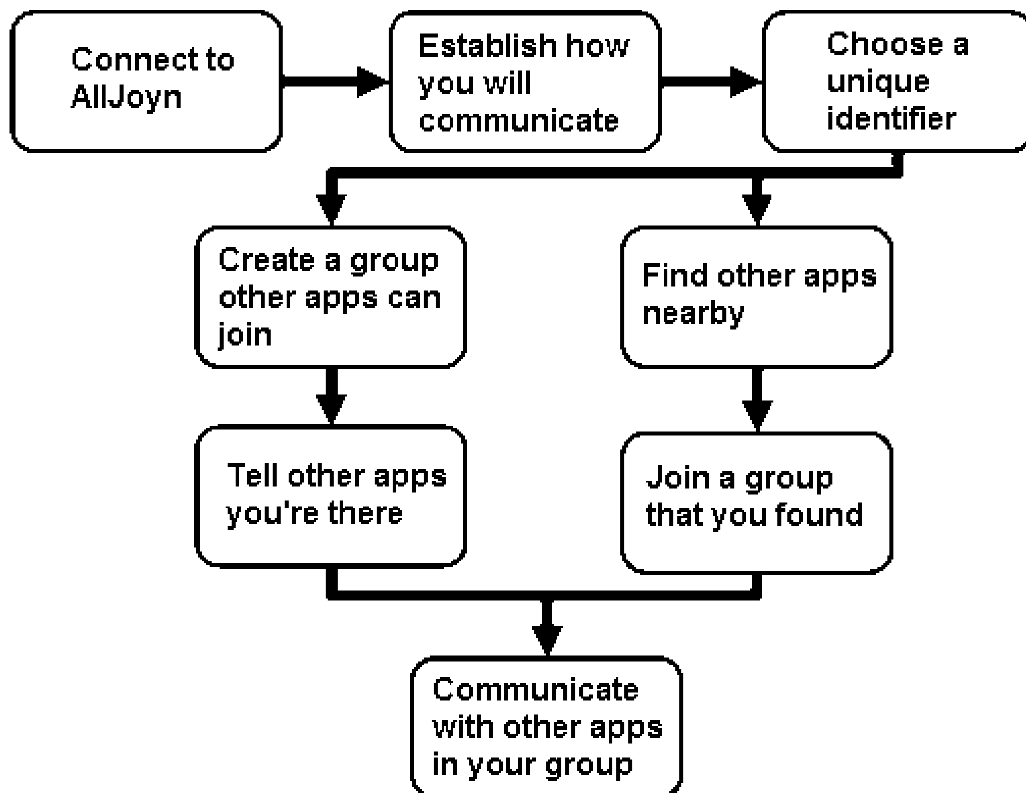

Figura 16 Workflow di un'applicazione AllJoyn.

4 A-3

A-3 è uno stile architetturale per la progettazione, l'implementazione, la gestione ed il coordinamento di sistemi distribuiti che presentino un elevato numero di nodi al loro interno e che siano altamente volatili. Esempi di applicazioni funzionanti su tale tipo di sistemi sono: monitoraggio ambientale e della vita degli animali selvaggi, agricoltura intelligente, automazione della casa e dei trasporti, sorveglianza, risposta alle emergenze, coordinamento della mobilità e di eventi sociali di massa [7]. La volatilità dei nodi del sistema richiede che quest'ultimo sia abbastanza flessibile per fronteggiare il continuo andirivieni di componenti, che determina variazioni nella disponibilità delle risorse del sistema nel tempo. Il sistema deve dunque essere gestito in modo tale che risponda a tali cambiamenti senza alterare né le proprie operazioni, né la qualità del servizio che esso offre. L'elevato numero di nodi che possono entrare a far parte del sistema impone che l'infrastruttura su cui è costruito il sistema stesso sia scalabile, in modo da far fronte ad un numero crescente di partecipanti. Con l'aumentare del numero di partecipanti aumenta sempre di più la rilevanza del problema di coordinare i componenti del sistema, cioè di organizzare i loro comportamenti al fine di raggiungere un obiettivo comune. Per evitare la creazione di colli di bottiglia è necessario che le capacità di coordinamento del sistema siano distribuite [1].

A-3 riduce il problema di coordinare un numero molto elevato ed estremamente variabile di componenti ad un problema più piccolo e meno dinamico: coordinare gruppi di componenti. L'astrazione fondamentale di A-3 è dunque quella di "gruppo". Un gruppo consente al progettista del sistema di coordinare più componenti come se fossero un'entità unica e di definire in maniera più semplice come essi debbano collaborare e come debbano essere gestiti. I componenti possono essere raggruppati per diversi motivi: perché sono concettualmente simili, perché sono fisicamente vicini o perché hanno uno stesso obiettivo. Spetta comunque al progettista del sistema decidere la politica di raggruppamento dei nodi [1]. I componenti di un gruppo possono memorizzare all'interno del gruppo stesso

informazioni rilevanti per l'applicazione ed il coordinamento in modo sicuro e robusto, al fine di recuperarle quando necessario.

A-3 classifica gli elementi di un gruppo in base al ruolo che hanno in esso. Un componente all'interno di un gruppo può assumere, in maniera mutualmente esclusiva, il ruolo di supervisore o quello di follower. Il supervisore coordina i follower mediante un ciclo di controllo MAPE, mentre questi ultimi si uniscono ad un gruppo per capire come si devono comportare nella situazione attuale [7]. Ogni gruppo è composto da uno ed un solo supervisore, da un numero teoricamente illimitato di follower, e da un connettore che li collega permettendo loro di scambiarsi messaggi asincroni. Se un connettore fallisce, il gruppo non esiste più ed ogni nodo del gruppo resta isolato: tutti i nodi provano ad istanziarne uno, ma solo uno vi riesce, mentre gli altri tentativi sono interpretati come richiesta di unirsi al nuovo gruppo. I follower possono comunicare solo con il supervisore del proprio gruppo, per fornirgli informazioni rilevanti per il coordinamento dell'intero sistema. Dunque la comunicazione tra follower deve avvenire fuori banda. Il supervisore può parlare solo con i follower del proprio gruppo, per fornire loro le direttive per il coordinamento, oppure i dati che essi possono utilizzare per cambiare autonomamente il loro comportamento. Il supervisore può spedire messaggi ai follower del proprio gruppo usando tre diversi paradigmi:

- messaggi broadcast, inviati a tutti i follower quando tutti loro devono agire in maniera coordinata;
- messaggi multicast, spediti solo ad alcuni follower individuati seguendo vari criteri, oppure scelti a caso;
- messaggi unicast, spediti ad un solo follower scelto secondo qualche criterio, oppure a caso [1].

A-3 richiede che la comunicazione avvenga con sincronia virtuale, cioè che i messaggi siano consegnati nello stesso ordine a tutti i destinatari. Inoltre A-3 fornisce la consegna ritardata dei messaggi: se un destinatario è disconnesso al momento della spedizione di un messaggio, ma si riconnette entro un tempo configurabile a priori, allora riceverà comunque il messaggio.

A-3 introduce anche il concetto di "view" del gruppo: supervisore e follower ricevono un aggiornamento ogni volta che un elemento si unisce al gruppo o lo lascia [7]. In questo modo, i follower possono sapere se e quando il proprio supervisore è caduto ed, in tal caso, iniziare un algoritmo di elezione di un nuovo supervisore. Inoltre, sempre in questo modo, il supervisore può implementare strategie di adattamento per rendere il sistema scalabile. Infatti, gruppi con troppo pochi nodi introducono astrazioni non necessarie, o perfino inadeguate, mentre gruppi con troppi nodi al loro interno portano ad un sovraccarico del supervisore. Ecco allora che un gruppo con troppo pochi nodi può essere fuso con un altro gruppo, oppure essere distrutto spostando altrove i suoi nodi. Ed ecco anche che un gruppo può essere diviso in più gruppi qualora il numero di nodi al suo interno aumenti troppo. I gruppi creati potrebbero essere indipendenti, oppure essere gestiti da un altro gruppo che li astrae come un'unica entità.

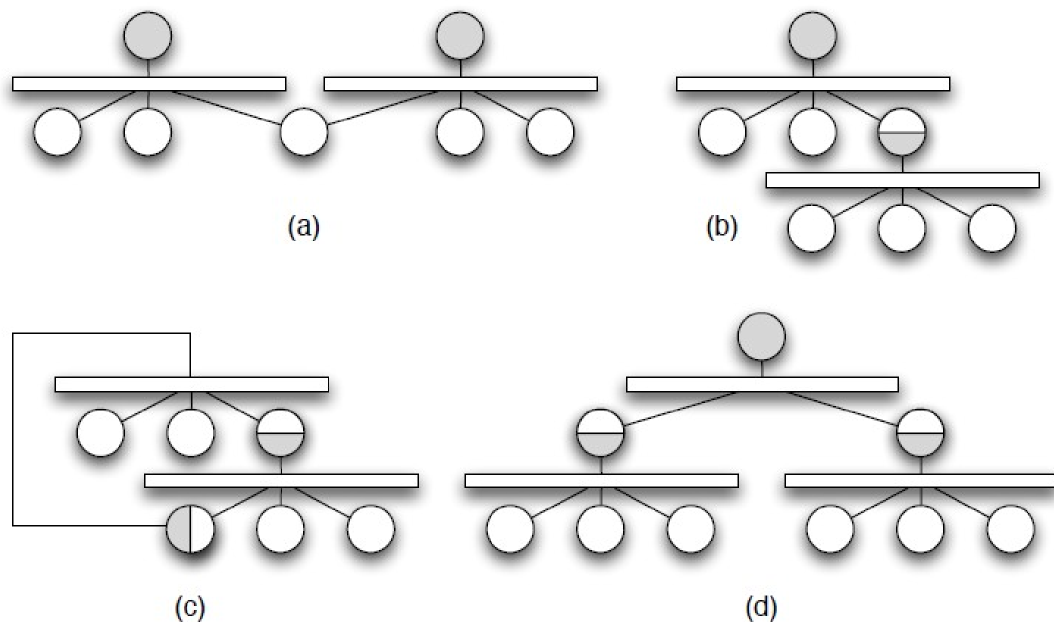

Figura 17 Composizione di gruppi.

Questa importante caratteristica di A-3 si chiama "composizione di gruppi" ed è realizzata offrendo ai nodi del sistema la capacità di poter appartenere a più gruppi e di poter avere ruoli diversi in gruppi diversi [7]. In figura 17 sono rappresentate

quattro tra le innumerevoli configurazioni ottenibili usando A-3. In particolare:

- in figura 17(a), un follower appartiene a più gruppi, dunque invia le sue informazioni a due supervisori contemporaneamente. Le direttive ricevute in risposta dai due supervisori potrebbero essere contrastanti: in tal caso è il follower stesso a dover risolvere il conflitto, dal momento che la configurazione non prevede una coordinazione esplicita tra i due supervisori.
- in figura 17(b), un supervisore controlla tre follower, di cui uno è il supervisore di un altro gruppo. L'esistenza di questo secondo gruppo è nascosta al supervisore del primo gruppo, ma il supervisore del secondo gruppo comunica la sua conoscenza al supervisore del primo gruppo, che così può conoscere lo stato dell'intero sistema senza parlare con tutti gli elementi del sistema stesso.
- in figura 17(c), la relazione di figura 17(b) è resa bidirezionale: entrambi i supervisori hanno una visione completa del sistema. Ogni supervisore spedisce le direttive solo ai suoi follower con messaggi multicast, evitando di spedirle agli altri supervisori per non creare cicli infiniti di spedizione.
- in figura 17(d), due gruppi di livello più basso sono coordinati da un gruppo di livello più alto. Questa configurazione pone il problema della sincronizzazione dei cicli di controllo MAPE dei tre supervisori, che probabilmente saranno in fasi diverse nello stesso momento. La soluzione al problema è lasciata al progettista del sistema. Il concetto fondamentale è comunque questo: i supervisori dei due gruppi di livello più basso contribuiscono alla fase di monitoraggio del supervisore del gruppo di livello più alto fornendo ad esso i dati rilevanti. Questi dati influiscono sulla decisione presa nel ciclo di controllo del supervisore del gruppo di livello più alto. Viceversa, le fasi di analisi, pianificazione ed esecuzione di tale ciclo influiscono sui cicli di controllo degli altri due supervisori, alterandone i risultati o addirittura il comportamento [1][7].

4.1 A3JG

Esiste già un'implementazione Java del framework A-3 denominata "A3JG", descritta nel dettaglio in [12]. Tale versione è basata su JGroups: un progetto open source che fornisce strumenti per gestire gruppi e scambiare messaggi all'interno di essi. Queste funzionalità sono fornite interamente dalla classe JChannel. Un canale JChannel può essere connesso solo ad un gruppo, dunque la connessione di un nodo a più gruppi è realizzata creando più canali in un nodo. Un nodo è implementato dalla classe A3JGNode. Le informazioni riguardanti lo stato di un gruppo sono memorizzate in una hashmap concorrente fornita da JGroups e denominata "ReplicatedHashMap". La classe A3JGroup contiene invece le informazioni riguardanti un gruppo che un nodo deve conoscere per potersi connettere ad esso. In particolare, in essa sono presenti due hashmap che contengono, rispettivamente, i possibili ruoli assumibili dal supervisore di un gruppo (classe A3JGSupervisorRole) e dai follower (classe A3JGFollowerRole). Di tutti i ruoli presenti, un solo A3JGSupervisorRole ed un solo A3JGFollowerRole sono scelti come ruoli di default per il gruppo. Un nodo può connettersi ad un gruppo solo se possiede il ruolo a lui richiesto. Il nodo che crea un gruppo è il primo a diventarne supervisore, mentre i supervisori che seguiranno saranno eletti tramite un meccanismo basato su fitness function. Tali fitness function devono essere definite dallo sviluppatore, mentre quest'ultimo non ha la necessità di riscrivere l'algoritmo di elezione. Allo sviluppatore è anche sufficiente stabilire il numero massimo di nodi che possono essere presenti in un gruppo, affinché il sistema divida automaticamente un gruppo in più sottogruppi qualora nel gruppo di partenza fossero presenti troppi nodi. Questo per rendere il sistema più scalabile.

4.2 Perché A3Droid?

A3Droid supera i limiti propri di A3JG, che sono sostanzialmente due:

- JGroups non è progettato per funzionare in sistemi mobile;

- A3JG non offre la possibilità allo sviluppatore di creare a proprio piacimento una composizione di gruppi e di mantenerla.

Ad alto livello, ho voluto mantenere in A3Droid la stessa struttura di A3JG, ma dovendo usare un framework diverso per supportare ambienti mobile, l'architettura di basso livello è cambiata. La differenza principale sta nella sparizione in A3Droid della ReplicatedHashMap presente in A3JG.

Descriverò A3Droid nel capitolo 5.

5 A3Droid

Mi è stato chiesto di implementare il framework A-3 descritto nel capitolo 4 e di realizzare delle API che consentissero di effettuare operazioni tra gruppi. Data la disponibilità di soli telefoni dotati di sistema operativo Android, mi è stato chiesto di scrivere queste API proprio per il sistema operativo Android, il quale richiede che esse siano scritte in linguaggio Java. Esse si poggiano sul framework AllJoyn descritto nel capitolo 3, che è stato scelto per la sua capacità di astrarre reti di tipo differente e di garantire interoperabilità tra dispositivi mobile funzionanti su piattaforme diverse. Le operazioni che mi è stato chiesto di implementare sono: creazione di un gruppo, distruzione di un gruppo, composizione tra gruppi. Quello di "gruppo" è un concetto fondamentale, alla cui descrizione dedicherò la prossima sezione (5.1). Come mostrerò, la struttura del gruppo ha complicato la realizzazione delle procedure di creazione e di distruzione di un gruppo, che descriverò rispettivamente nelle sezioni 5.2 e 5.3. Tuttavia, le classi che realizzano quanto appena detto sono trasparenti al progettista di applicazioni A-3, infatti tutte le API da me implementate sono offerte dalla classe A3Node, che rappresenta un dispositivo fisico facente parte del sistema e che è strutturata per realizzare la composizione di gruppi. Nella sezione 5.4 illustrerò dapprima la struttura della classe A3Node, ed in seguito l'implementazione da me fornita alle API di composizione dei gruppi.

L'operazione fondamentale da compiere quando si utilizzano le API da me realizzate è dunque creare un oggetto di classe A3Node. Nel realizzare queste API ho supposto che chi progetta un sistema A-3 abbia chiaro già in fase di progettazione:

- quali gruppi potranno essere creati dai nodi runtime;
- come i nodi si debbano comportare all'interno di tali gruppi.

Per definire i gruppi che possono caratterizzare un sistema A-3 ho realizzato la classe GroupDescriptor, mentre per rappresentare i comportamenti dei nodi all'interno di ogni gruppo ho realizzato la classe A3Role. Ho poi esteso quest'ultima con le classi A3SupervisorRole e A3FollowerRole per rappresentare con esse rispettivamente il

ruolo di supervisore e quello di follower. Descriverò le classi GroupDescriptor, A3Role, A3SupervisorRole e A3FollowerRole nella sezione 5.5, nella quale illustrerò anche come il sistema recuperi ed istanzi il ruolo corretto dato il nome di un gruppo, ed i vincoli che tale procedura impone sul nome dei gruppi.

Un oggetto di classe A3Node deve essere creato fornendo al suo costruttore una lista di descrittori GroupDescriptor, rappresentante tutti i gruppi che possono essere creati runtime, ed una lista di ruoli A3Role, rappresentante i ruoli che il dispositivo potrà assumere. Non è detto, infatti, che un dispositivo possa assumere tutti i ruoli possibili: dipenderà dal sistema realizzato e dalle caratteristiche del dispositivo stesso. Nel dettaglio:

- un nodo potrebbe diventare sia supervisore sia follower di un gruppo;
- un nodo potrebbe diventare supervisore di un gruppo senza poterne diventare follower;
- un nodo potrebbe diventare follower di un gruppo senza poterne diventare supervisore;
- un nodo non potrebbe diventare né supervisore né follower di un gruppo.

Nel primo caso, non ci sono problemi. Nell'ultimo caso, il nodo non entra a far parte del gruppo, dunque non ci sono problemi. Per gestire il secondo ed il terzo caso limitando il consumo di CPU e batteria, mi è stato chiesto di realizzare un gruppo in cui riunire tutti i nodi che rientrano in tali condizioni. Ho chiamato tale gruppo "wait": lo descriverò nella sezione 5.6. Il gruppo "wait" è coinvolto nella procedura di elezione di un nuovo supervisore, che illustrerò nella sezione 5.7. Tutte le API prevedono scambio di messaggi tra i nodi del sistema. Ho implementato tali messaggi con la classe A3Message, che descriverò nella sezione 5.8. In conclusione, per riassumere, elencherò nella sezione 5.9 i passi necessari per creare un'applicazione A-3 che utilizzi queste API.

5.1 Il gruppo

AllJoyn fornisce un concetto strettamente legato al concetto di gruppo proposto da A-3: quello di sessione. La sessione in AllJoyn, infatti, è definita proprio come un gruppo di applicazioni che permette a queste ultime di comunicare tra loro in modo efficiente [10]. A-3 definisce un gruppo come un insieme di componenti (di cui uno ed un solo supervisore e più follower) collegati da un connettore. Ho implementato il connettore di A-3 creando la classe Service, che corrisponde al concetto di session owner in AllJoyn. Ho invece implementato un componente di A-3 creando la classe A3Channel, che corrisponde al concetto di joiner in AllJoyn. Un gruppo A-3 corrisponde ad una ed una sola sessione multipoint in AllJoyn, che è formata da uno ed un solo Service e uno o più A3Channel nelle mie API. Le corrispondenze tra A-3, AllJoyn e le classi da me realizzate sono illustrate a pagina seguente in figura 18.

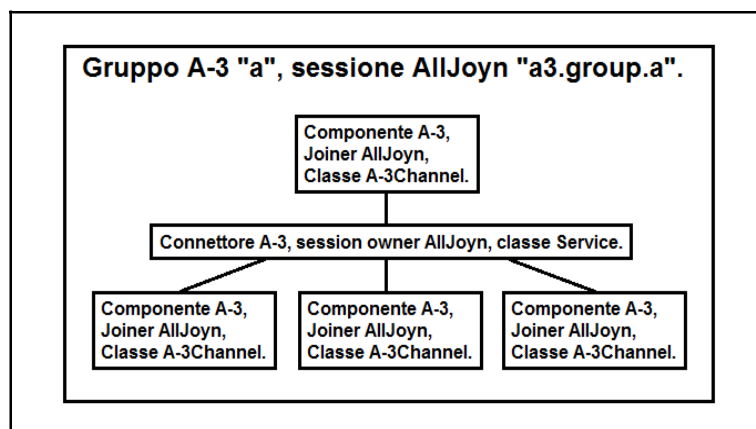

Figura 18 Corrispondenze tra A-3, AllJoyn e le API da me realizzate.

La classe Service offre al supervisore i metodi per la spedizione di messaggi broadcast, unicast e multicast, e ai follower un metodo per la spedizione di messaggi al supervisore. La spedizione di messaggi broadcast è implementata mediante la trasmissione di BusSignal AllJoyn da parte del Service. Per implementare la spedizione di messaggi unicast è invece necessario che ogni dispositivo all'interno del gruppo sia univocamente identificabile sul bus AllJoyn. L'unico modo per realizzare ciò è far sì che su ogni dispositivo sia presente un'applicazione session

owner AllJoyn, che ho chiamato A3UnicastReceiver e che implementa l'interfaccia A3UnicastInterface. Il nome da essa pubblicata sul bus è un'estensione del nome del gruppo cui il dispositivo appartiene, realizzata mediante l'aggiunta di un hash dell'identificativo che esso ottiene quando si unisce alla sessione AllJoyn con il Service. L'applicazione joiner per spedire messaggi unicast, che ho chiamato A3UnicastTransmitter, risiede sul Service. La spedizione di messaggi multicast è implementata come serie di spedizioni di messaggi unicast verso ogni dispositivo destinatario. Può avvenire sia indicando preventivamente ed arbitrariamente i dispositivi destinatari, sia tramite un meccanismo publish/subscribe, gestito sul Service dalla classe Subscription. Per risolvere problemi legati alla corretta comunicazione dell'indirizzo di un nuovo supervisore, ho implementato la spedizione di messaggi al supervisore come un BusSignal AllJoyn ricevuto solo dal supervisore del gruppo. L'interfaccia AllJoyn implementata dalla classe Service è l'interfaccia A3ServiceInterface.

La classe A3Channel è un'applicazione joiner AllJoyn che consente ad un dispositivo A3Node di comunicare con gli altri dispositivi membri del gruppo. In essa sono presenti tutti i metodi per l'invocazione da remoto dei metodi del Service: quelli per la spedizione di messaggi unicast, multicast e broadcast sono invocabili solo se il dispositivo è il supervisore del gruppo rappresentato dal canale A3Channel, mentre quello per la spedizione di messaggi al supervisore è invocabile solo se il dispositivo è un follower. Nella classe A3Channel sono presenti anche i ruoli che il dispositivo A3Node può assumere nel gruppo, di cui uno solo alla volta attivo. Nella classe A3Channel sono filtrati i messaggi di sistema del framework A-3, che risultano così essere trasparenti all'utilizzatore del framework A-3 stesso. I messaggi applicativi sono invece inviati al ruolo correntemente attivo.

5.2 La creazione di un gruppo

Un gruppo A-3 è creato quando un canale A3Channel forma una sessione multipoint

AllJoyn con il Service corrispondente al gruppo A-3 stesso. Ma se il Service non esiste ancora, allora bisogna crearlo. Quando un dispositivo A3Node vuole entrare a far parte di un gruppo crea un canale A3Channel, il quale fa partire il discovery del nome del gruppo A-3 sul bus AllJoyn. Se il nome è trovato entro un intervallo di tempo prefissato, allora il Service esiste ed il canale A3Channel può formare una sessione con esso. Altrimenti il canale A3Channel crea il Service, il quale pubblica il nome del gruppo A-3 sul bus, permettendo al canale A3Channel di potersi unire alla sessione.

5.3 La distruzione di un gruppo

Un gruppo è distrutto in ciascuno di questi due casi:

- non esistono più componenti collegati al connettore;
- non esiste più il connettore.

Per gestire il primo caso ho dotato la classe Service di un oggetto di classe View, la quale tiene traccia dei canali A3Channel connessi al Service stesso. Per fare ciò ho sfruttato i callback di AllJoyn che notificano la connessione e la disconnessione di un componente del gruppo. Se nessun A3Channel è più presente nell'oggetto di classe View associato ad un Service, allora tale Service è distrutto.

Il secondo caso introduce una situazione anomala, in cui si ha la presenza contemporanea di più Service per uno stesso gruppo. Il problema è dato dal funzionamento del discovery dei demoni AllJoyn e si presenta anche nel caso in cui gruppi con lo stesso nome sono creati da A3Channel che risiedono su dispositivi tra loro abbastanza lontani che uno dei gruppi già esistenti non sia visibile agli altri. Per fronteggiare queste situazioni ho introdotto la classe DiscoveryManager ed un algoritmo di rimozione dei gruppi duplicati. Un oggetto di classe DiscoveryManager è posto su ogni Service ed effettua il discovery sul bus AllJoyn del nome del gruppo a cui il Service corrisponde. Se il DiscoveryManager trova tale nome più di una

volta, allora notifica il Service, il quale spedisce un sessionless BusSignal AllJoyn denominato "GroupIsDuplicated". In esso sono riportati il nome del gruppo duplicato, lo unique name del supervisore, il numero di A3Channel nella sessione con il Service ed un numero casuale. Essendo sessionless, il BusSignal AllJoyn arriva anche ai Service dei gruppi che non devono processarlo, cioè a quelli con nome del gruppo diverso da quello contenuto nel BusSignal, oppure con nome del gruppo ed indirizzo del supervisore uguale a quello presente nel BusSignal. Un Service che processa il BusSignal confronta innanzitutto il numero di A3Channel nella propria sessione con quello contenuto nel BusSignal ricevuto:

- se il primo è maggiore, allora il Service è disconnesso, gli A3Channel effettuano di nuovo la procedura di connessione al gruppo e si uniscono ad uno qualsiasi dei Service rimasti;
- se il primo è minore, il Service spedisce un altro BusSignal GroupIsDuplicated.
- se sono uguali, il Service passa a confrontare il numero casuale contenuto nel BusSignal ricevuto con un numero casuale da lui generato: se il primo è maggiore allora il Service è disconnesso; se il primo è minore il Service spedisce un altro BusSignal GroupIsDuplicated; se sono uguali, allora è generato un nuovo numero casuale questo confronto si ripete.

Risultato di questo algoritmo è che sopravvive solo un Service, al quale si collegano tutti gli A3Channel prima sparsi in vari gruppi con lo stesso nome. La corrispondenza tra gruppo A-3 e sessione AllJoyn è così resa biunivoca.

Un altro problema legato al caso di caduta del Service è il recupero delle informazioni riguardanti il modo in cui i gruppi sono composti. In particolare, il nuovo supervisore deve essere in grado di ristabilire le relazioni tra il suo gruppo e gli altri gruppi cui era collegato il supervisore precedente, al fine di non alterare la composizione dei gruppi. Per risolvere questo problema, ho realizzato la classe Hierarchy, contenente i nomi dei gruppi di cui il supervisore deve essere follower. Al fine di recuperare correttamente le informazioni dopo la caduta del Service, ho inserito la classe Hierarchy all'interno della classe A3Channel, cosicché ogni

dispositivo A3Node connesso al gruppo ne abbia una copia identica.

5.4 La classe A3Node

La classe A3Node rappresenta un dispositivo fisico facente parte di un sistema A-3 e la sua struttura è tale da realizzare la composizione dei gruppi. La composizione dei gruppi consente ai nodi del sistema la capacità di poter appartenere a più gruppi e di poter avere ruoli diversi in gruppi diversi [7], cioè di essere contemporaneamente supervisore di alcuni gruppi e follower di altri. Nella struttura della classe A3Node ho dunque incluso:

- una lista di A3Channel, i quali le consentono di appartenere a più gruppi contemporaneamente;
- una lista di A3Role, rappresentante i ruoli che il dispositivo potrà assumere;
- una lista di descrittori GroupDescriptor, rappresentante tutti i gruppi che possono essere creati runtime, per il recupero dei ruoli corretti da passare ad un A3Channel all'atto della connessione ad un gruppo, secondo la procedura illustrata nella sezione 5.5.

Nella classe A3Node sono presenti gli stessi metodi per lo scambio di messaggi presenti nella classe A3Channel, che ho però reso parametrici anche rispetto al nome del gruppo: un dispositivo può così inviare messaggi in uno o più gruppi in reazione ad un messaggio ricevuto in un diverso gruppo o ad un evento esterno.

Il metodo "public boolean connect(String, boolean, boolean)" maschera le operazioni eseguite per connettersi ad un gruppo, che sono le seguenti:

- verifica che non esista già un canale A3Channel per il gruppo (altrimenti il metodo termina, garantendo che un dispositivo sia connesso ad un gruppo meditante uno ed un solo canale A3Channel);
- verifica che il descrittore del gruppo sia presente nella lista, secondo la procedura descritta nella sezione 5.5 (in caso contrario il metodo torna true se

il canale è connesso, false altrimenti);

- verifica che i ruoli contenuti nel descrittore siano presenti nella lista (il metodo torna false se nessuno dei due ruoli è presente);
- creazione e connessione del canale A3Channel, secondo la procedura descritta nella sezione 5.2;
- richiesta dell'indirizzo del supervisore da parte del canale A3Channel;
- disconnessione del canale A3Channel e suo ingresso nel gruppo "wait" qualora al canale sia richiesto di attivare il ruolo mancante (ciò è determinato dal confronto tra l'indirizzo del canale A3Channel e quello del supervisore).

Il metodo "disconnect(String, boolean)" provoca l'uscita del nodo dal gruppo specificato e dunque la rimozione del corrispondente canale A3Channel dalla lista. Ho voluto introdurre due flag per impedire all'applicazione di distruggere un canale in uso dal sistema e viceversa, quindi un canale A3Channel è rimosso dalla lista solo se né l'applicazione né il sistema usano più tale canale.
Gli altri metodi mascherano le operazioni di composizione tra gruppi, che sono stack, peers, hierarchy, split e le loro rispettive inverse: reverseStack, reversePeers, reverseHierarchy e merge. Ho implementato ciascuna di esse con un omonimo metodo della classe A3Node che prende come parametri in ingresso i nomi dei gruppi coinvolti. Per spiegare l'implementazione di ogni metodo userò qui di seguito tre gruppi denominati "a", "b" e "c".

stack: l'operazione stack("a", "b") può essere invocata solo dai supervisori dei gruppi "a" e "b" e consiste nel rendere follower del gruppo "a" il supervisore del gruppo "b", come mostrato in figura 19 a pagina seguente.

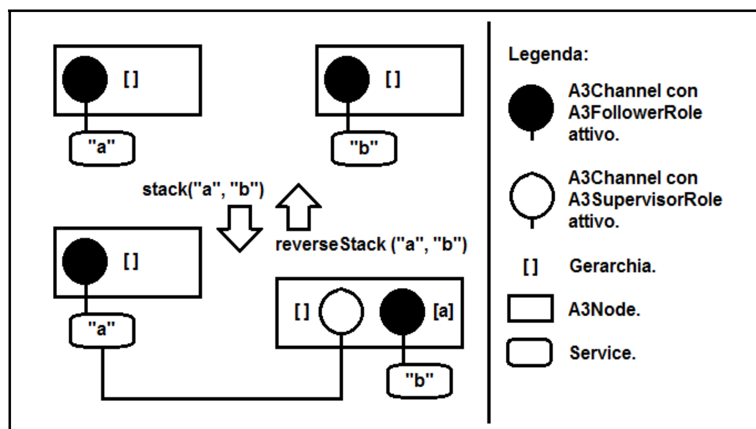

Figura 19 Le operazioni "stack" e "reverseStack".

Ciò significa che il supervisore del gruppo "b" deve creare un canale A3Channel verso il gruppo "a", se non già creato in precedenza, e deve comunicare a tutti i canali A3Channel nel proprio gruppo di aggiungere il nome del gruppo "a" alla propria gerarchia. Affinché ciò sia possibile, è necessario che il supervisore del gruppo "b" abbia il ruolo necessario per connettersi al gruppo "a" come follower. Se tale ruolo, per un motivo o per un altro, non è attivato, allora il canale A3Channel verso il gruppo "b" è posto in stato di attesa, poi distrutto, e l'operazione di stack fallisce. Altrimenti nella gerarchia di tutti i canali A3Channel nel gruppo "b" comparirà il nome del gruppo "a", causa spedizione broadcast di un opportuno messaggio da parte del Service del gruppo "b". Tutto questo avviene grazie all'invocazione del metodo actualStack(String, String) della classe A3Node, in particolare dell'istruzione actualStack("a", "b") sul supervisore del gruppo "b". Il metodo actualStack(String, String) può essere invocato sia localmente sia da remoto. In questo secondo caso, è creata una connessione temporanea tra i due supervisori, nella quale sono scambiati due messaggi: l'ordine di esecuzione di actualStack (String, String) e la comunicazione dell'esito di tale operazione. La connessione è temporanea perchè deve essere creata in senso inverso durante l'esecuzione di actualStack (String, String). Se il tentativo di connessione temporanea fallisce, allora l'operazione è annullata subito. L'invocazione di stack("a", "b") determina se l'istruzione actualStack("a", "b") deve essere eseguita in locale o remotamente. In entrambi i casi, l'esito dell'operazione stack("a", "b") è visualizzato dal metodo

stackReply(String, String, boolean) della classe A3Node, sul dispositivo sul quale essa è stata invocata. Tale metodo può tuttavia essere sovrascritto per eseguire altre operazioni.

reverseStack: l'operazione reverseStack("a", "b") può essere invocata solo dai supervisori dei gruppi "a" e "b" e consiste nel distruggere la relazione gerarchica creata invocando l'operazione stack("a", "b"), come mostrato in figura 19. Ciò significa che il supervisore del gruppo "b" deve distruggere il proprio canale A3Channel verso il gruppo "a", se esso non esisteva già prima dell'invocazione dell'operazione stack("a", "b"), e deve comunicare a tutti i canali A3Channel nel proprio gruppo di rimuovere il nome del gruppo "a" dalla propria gerarchia spedendo loro un opportuno messaggio broadcast. L'operazione è eseguita di fatto dal metodo actualReverseStack("a", "b") della classe A3Node, che è invocato localmente se l'operazione reverseStack("a", "b") è invocata dal supervisore del gruppo "b", oppure remotamente se l'operazione reverseStack("a", "b") è invocata dal supervisore del gruppo "a". In tal caso, l'operazione inizia e termina con la spedizione di opportuni messaggi tra i supervisori dei due gruppi. In entrambi i casi, l'esito dell'operazione è visualizzato sul dispositivo da cui è stata invocata l'operazione reverseStack("a", "b"), mediante il metodo reverseStackReply(String, String, boolean) della classe A3Node. Tale metodo può tuttavia essere sovrascritto per eseguire altre operazioni.

peers: l'operazione peers("a", "b") può essere invocata solo dai supervisori dei gruppi "a" e "b" e consiste nel rendere follower del gruppo "a" il supervisore del gruppo "b" e follower del gruppo "b" il supervisore del gruppo "a", come mostrato in figura 20 a pagina seguente.

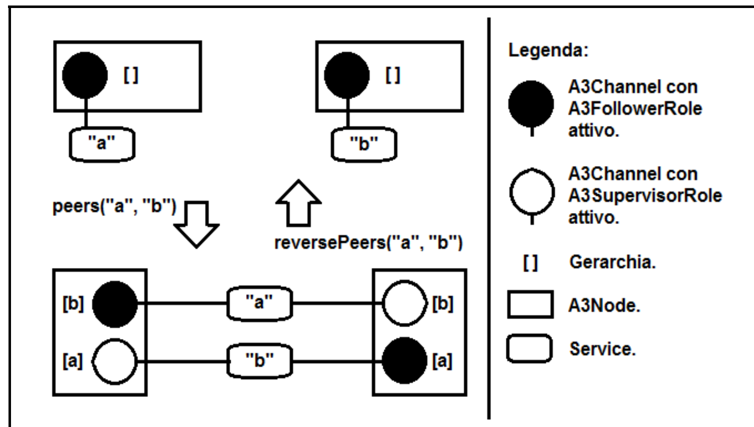

Figura 20 Le operazioni "peers" e "reversePeers".

Si tratta di un'esecuzione di actualStack("a", "b") sul supervisore del gruppo "b" e di actualStack("b", "a") sul supervisore del gruppo "a". Questo implica un'invocazione remota ed una locale del metodo actualStack(String, String), e l'eventuale annullamento dell'operazione non appena se ne scopre il fallimento. L'esito dell'operazione peers("a", "b") è visualizzato sul dispositivo da cui essa è stata invocata, mediante il metodo peersReply(String, String, boolean) della classe A3Node. Tale metodo può tuttavia essere sovrascritto per eseguire altre operazioni.

reversePeers: l'operazione reversePeers("a", "b") può essere invocata solo dai supervisori dei gruppi "a" e "b" e consiste nel distruggere la relazione gerarchica creata invocando l'operazione peers("a", "b"), come mostrato in figura 20. Ciò avviene mediante l'esecuzione dell'istruzione reverseStack("a", "b") sul supervisore del gruppo "b" e dell'istruzione reverseStack("b", "a") sul supervisore del gruppo "a". Questo implica un'invocazione remota ed una locale del metodo reverseStack(String, String). L'esito dell'operazione reversePeers("a", "b") è visualizzato sul dispositivo da cui essa è stata invocata, mediante il metodo reversePeersReply(String, String, boolean) della classe A3Node. Tale metodo può tuttavia essere sovrascritto per eseguire altre operazioni.

hierarchy: l'operazione hierarchy("c", "a", "b") può essere invocata solo dai supervisori dei gruppi "a" e "b" e consiste nel rendere follower del gruppo "c" i

supervisori dei gruppi "a" e "b", come illustrato in figura 21 a pagina seguente.

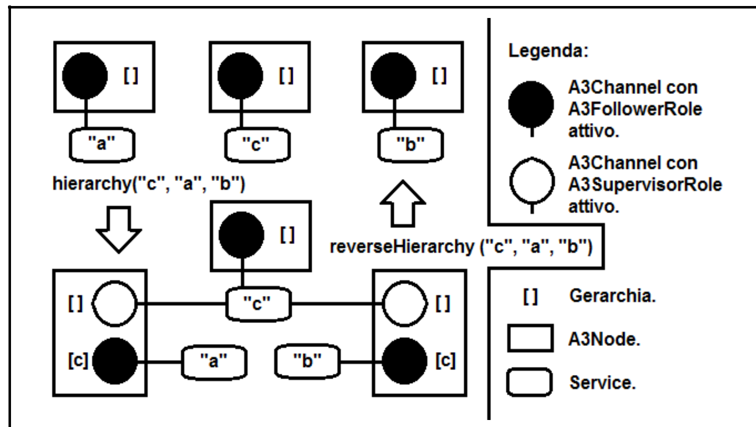

Figura 21 "Le operazioni hierarchy" e "reverseHierarchy".

Si tratta di un'esecuzione di actualStack("c", "b") sul supervisore del gruppo "b" e di actualStack("c", "a") sul supervisore del gruppo "a". Questo implica un'invocazione remota ed una locale del metodo actualStack(String, String), e l'eventuale annullamento dell'operazione non appena se ne scopre il fallimento. L'esito dell'operazione hierarchy("c", "a", "b") è visualizzato sul dispositivo da cui essa è stata invocata, mediante il metodo hierarchyReply(String, String, String, boolean) della classe A3Node. Tale metodo può tuttavia essere sovrascritto per eseguire altre operazioni.

reverseHierarchy: l'operazione reverseHierarchy("c", "a", "b") può essere invocata solo dai supervisori dei gruppi "a" e "b" e consiste nel distruggere la relazione gerarchica creata invocando l'operazione hierarchy("c", "a", "b"). Ciò avviene mediante l'esecuzione dell'istruzione reverseStack("c", "b") sul supervisore del gruppo "b" e dell'istruzione reverseStack ("c", "a") sul supervisore del gruppo "a". Questo implica un'invocazione remota ed una locale del metodo reverseStack(String, String). L'esito dell'operazione hierarchy("c", "a", "b") è visualizzato sul dispositivo da cui essa è stata invocata, mediante il metodo reverseHierarchyReply(String, String, String, boolean) della classe A3Node. Tale metodo può tuttavia essere sovrascritto per eseguire altre operazioni.

merge: l'operazione merge("a", "b") può essere invocata solo dai supervisori dei gruppi "a" e "b" e consiste nel trasferire tutti i canali A3Channel del gruppo "b" nel gruppo "a", distruggendo il gruppo "b", come si vede in figura 22.

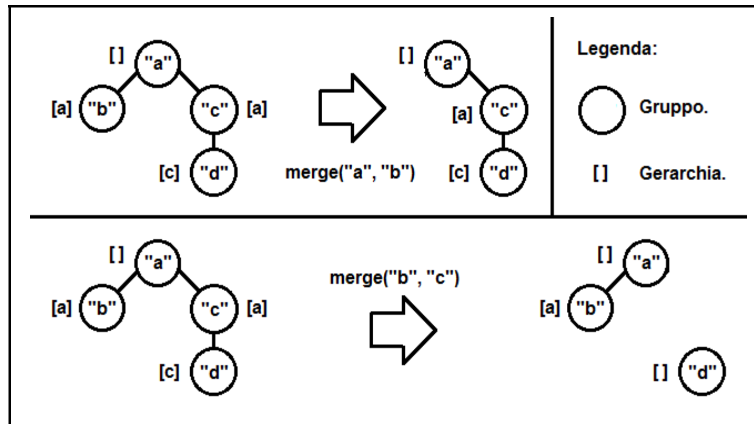

Figura 22 L'operazione "merge".

Sono trasferiti nel gruppo "a" anche i canali A3Channel nel gruppo "wait", in attesa per il gruppo "b". Un apposito messaggio broadcast è spedito ai canali A3Channel del gruppo "b", ed un messaggio analogo è spedito broadcast ai canali A3Channel del gruppo "wait". I canali A3Channel, ricevuto tale messaggio, invocano il metodo actualMerge(String, String) della classe A3Node, actualMerge("a", "b") nello specifico. Ogni canale A3Channel nel gruppo "b", o in attesa per il gruppo "b", è disconnesso da tale gruppo e connesso al gruppo "a". La connessione al gruppo "a" è possibile in entrambi i casi solo se il dispositivo A3Node possiede almeno uno dei due ruoli richiesti, altrimenti il dispositivo A3Node non risulterà connesso né al gruppo "a" né al gruppo "b". Nel caso il dispositivo A3Node disponesse di uno solo dei due ruoli richiesti per entrare nel gruppo "a", esso risulterà connesso a tale gruppo solo se il ruolo attivatosi sarà quello da lui posseduto, altrimenti sarà posto in stato di attesa. La disconnessione di tutti i canali A3Channel nel gruppo "b" comporta la disconnessione del Service del gruppo "b" e dunque la distruzione del gruppo "b".

split: l'operazione di split consiste nel creare un nuovo gruppo e nello spostarvi

alcuni canali A3Channel facenti parte di un altro gruppo. Il nuovo gruppo ha lo stesso descrittore del gruppo da cui è originato ed il suo nome è un estensione di quello di tale gruppo mediante l'aggiunta di un numero progressivo. Per esempio, i gruppi generati dal gruppo "a" si chiameranno "a_0", "a_1", "a_2", e così via. I gruppi generati a partire, ad esempio, dal gruppo "a_1" si chiameranno "a_1_0", "a_1_1", "a_1_2" e così via. Nel caso in cui esistano, ad esempio, i gruppi "a_0", "a_2", ma non esista il gruppo "a_1", il prossimo gruppo generato dal gruppo "a" si chiamerà "a_3". Il nuovo gruppo è indipendente da tutti gli altri, la sua gerarchia è vuota e può essere composto con gli altri gruppi usando le altre API. Mi è stato chiesto di implementare questa operazione in tre modi diversi, che ho denominato split casuale, split intera e split booleana. Di seguito spiegherò il significato e l'implementazione di ciascuno dei suddetti tre modi. In ogni caso, l'operazione di split è invocabile solo dal supervisore del gruppo ed è impossibile spostare nel nuovo gruppo il supervisore del gruppo da dividere.

split casuale: nel nuovo gruppo sono spostati n canali A3Channel del gruppo di partenza scelti a caso dal Service, con n arbitrario. E' implementata dal metodo split(String, int) della classe A3Node. Per esempio, l'esecuzione dell'istruzione split("a", 10) sposta nel nuovo gruppo dieci canali A3Channel a caso, tra quelli connessi al gruppo "a". Ciò avviene perché il dispositivo A3Node spedisce al Service del gruppo "a" l'ordine di effettuare l'operazione di split casuale. Allora il Service comunica a tutti i canali A3Channel nel gruppo "a" di incrementare un contatore contenuto nella classe Hierarchy. Tale contatore è usato solo per determinare il nome del nuovo gruppo, che risulterà essere "a_1" se lo stesso contatore vale 1. Tale contatore è replicato nell'oggetto Hierarchy di ogni canale A3Channel per poter recuperare l'informazione in caso di caduta del Service. Ad ogni canale A3Channel scelto a caso dal Service è inviato un messaggio unicast che ordina la disconnessione dal gruppo "a" e la connessione al nuovo gruppo.

split intera: nel nuovo gruppo sono spostati gli n canali A3Channel del gruppo di partenza che presentano i migliori valori di un'apposita fitness function intera. Il numero n è arbitrario. La collezione delle fitness function intere e la selezione degli n

canali A3Channel con i valori migliori delle stesse sono svolte da un oggetto di classe FitnessFunctionManager situata sul ruolo del supervisore del gruppo. L'operazione è implementata dal metodo splitWithIntegerFitnessFunction(String, int) della classe A3Node. Per esempio, l'esecuzione dell'istruzione splitWithIntegerFitnessFunction("a", 10) sposta nel nuovo gruppo i dieci canali A3Channel del gruppo "a" che presentano i valori di fitness function migliori. Questa operazione è effettuabile solo sovrascrivendo il metodo getIntegerSplitFitnessFunction() del descrittore del gruppo, estensione della classe GroupDescriptor.

split booleana: nel nuovo gruppo sono spostati tutti i canali A3Channel del gruppo "a" che soddisfano una certa condizione booleana determinata dal metodo getBooleanSplitFitnessFunction(), opportunamente sovrascritto, del descrittore del gruppo, estensione della classe GroupDescriptor. Questa operazione è effettuabile solo sovrascrivendo tale metodo ed è implementata dal metodo splitWithBooleanFitnessFunction(String) della classe A3Node. Per esempio, l'esecuzione dell'istruzione splitWithBooleanFitnessFunction("a") sposta nel nuovo gruppo tutti i canali A3Channel del gruppo “a” che soddisfano una condizione booleana. Il supervisore spedisce un messaggio broadcast, ordinando a tutti i canali A3Channel del proprio gruppo di valutare la loro fitness function booleana e di cambiare gruppo se è il caso.

5.5 Le classi A3Role, A3SupervisorRole, A3FollowerRole e GroupDescriptor

All'interno di ogni gruppo esistono due tipi di nodi: nodo supervisore e nodi follower. Il comportamento di ognuno di essi è detto “ruolo” e deve essere implementato da un'estensione della classe A3Role.
La classe A3Role offre un costruttore senza parametri che è invocato dal sistema al momento della connessione ad un nuovo gruppo. L’istruzione eseguita è

“Class.forName(roleId).getConstructor().newInstance();”, la quale recupera la classe di nome “roleId”, ne recupera il costruttore senza parametri e lo esegue. Per questo motivo, le estensioni della classe A3Role devono possedere un costruttore senza parametri. L’unica istruzione al suo interno deve essere l’istruzione “super();”. Bisogna considerare come costruttore effettivo il metodo astratto “public void onActivation()” della classe A3Role, il quale è invocato al momento dell’attivazione del ruolo. Tale attivazione consiste nell’avviare un ciclo che è eseguito finché il ruolo resta attivo. La logica eseguita all’interno di tale ciclo è definita nell’implementazione del metodo astratto “public void logic()” della classe A3Role. La classe A3Role consente anche di definire la reazione alla ricezione di un messaggio applicativo mediante l’implementazione del metodo astratto “public void receiveApplicationMessage(A3Message message)”.

Ho reso la classe A3Role trasparente al progettista di un sistema A-3, estendendola con le classi astratte A3SupervisorRole e A3FollowerRole: la prima implementa il comportamento di un supervisore ed è dunque in grado di ricevere e di gestire i messaggi di sistema provenienti dai follower; la seconda implementa il comportamento di un follower, è stata creata solo per distinguerla dalla prima e non aggiunge nulla alla classe “A3Role”. Lo scambio di messaggi applicativi deve essere gestito estendendo opportunamente le classi A3SupervisorRole e A3FollowerRole. Le estensioni di tali classi devono implementare i metodi “public void onActivation()”, “public void logic()” e “public void receiveApplicationMessage(A3Message message)” ereditati dalla classe A3Role.

Per definire un gruppo bisogna crearne il descrittore estendendo la classe GroupDescriptor. Un descrittore contiene innanzitutto una stringa che è il nome del gruppo. A causa di vincoli imposti da AllJoyn e da A-3, il nome di un gruppo deve avere il seguente formato:

- la sua dimensione deve essere minore di 246 caratteri;
- può contenere al suo interno solo caratteri alfanumerici (a-z A-Z 0-9) ed il simbolo ".";
- non può iniziare con “.” o con un numero (0-9);

- un numero (0-9) non può mai seguire il simbolo ".";
- non possono essere presenti al suo interno più simboli "." consecutivi;
- non può essere “wait”.

Si risale al descrittore corretto di un gruppo confrontando la parte iniziale del nome del gruppo stesso con il nome contenuto nel descrittore: se i due nomi coincidono, o se il primo è il secondo seguito dal simbolo “_”, allora si è trovato il descrittore corretto. Per esempio, un descrittore con il nome “a” caratterizza i gruppi “a”, “a_0”, “a_0_0”, ma non caratterizza i gruppi “ab” o “a.b”.
Il descrittore di un gruppo contiene anche gli identificativi dei due ruoli di supervisore e di follower, che corrispondono ai nomi estesi delle classi che estendono la classe A3Role.
La classe GroupDescriptor è astratta e deve essere estesa per implementare il calcolo dei valori delle fitness function usati nelle operazioni di elezione di un nuovo supervisore e di split.

5.6 Il gruppo “wait”

Il gruppo “wait” è il gruppo in cui sono posti i canali A3Channel nei seguenti due casi:

- il dispositivo A3Node possiede solo il ruolo di supervisore di un gruppo, ma il supervisore di quel gruppo esiste già;
- il dispositivo A3Node possiede solo il ruolo di follower di un gruppo, ma è stato eletto supervisore del gruppo stesso (in realtà questo succede solo se il dispositivo A3Node è l’unico a far parte di quel gruppo).

In entrambi i casi, i canali A3Channel sono risvegliati non appena si verificano le condizioni per cui il dispositivo può entrare a far parte del gruppo. Ho realizzato tale meccanismo creando le classi WaitGroupDescriptor, WaitSupervisorRole e WaitFollowerRole: la prima estende la classe GroupDescriptor ed è il descrittore del

gruppo "wait", mentre le altre due estendono rispettivamente le classi A3SupervisorRole e A3FollowerRole, e sono i ruoli che i dispositivi assumono nel gruppo "wait". In esse non è eseguita alcuna logica, in quanto i dispositivi nel gruppo "wait" devono solo attendere di essere risvegliati.
Quando un canale A3Channel scopre di non poter attivare il ruolo richiesto, è disconnesso e posto in stato d'attesa, ma rimane nella lista dei canali A3Channel presente sul dispositivo A3Node. E' poi chiamato il metodo "connect(String, boolean, boolean)" per la connessione al gruppo "wait" (si veda la sezione 5.6).

Quando un canale A3Channel è risvegliato, è nuovamente connesso e non è più in stato d'attesa. Tuttavia, il canale A3Channel verso il gruppo "wait" continua ad esistere e ad essere presente nella lista del dispositivo A3Node finché non è risvegliato l'ultimo canale in stato d'attesa.

5.7 L'elezione del supervisore

Siccome A-3 prevede che in un gruppo esista sempre un supervisore, un canale A3Channel è automaticamente eletto supervisore quando è l'unico canale A3Channel presente in un gruppo. Se il canale A3Channel in questione non può diventare supervisore del gruppo, esso è disconnesso (si veda la sezione 5.6). Non essendoci più canali A3Channel connessi al Service, quest'ultimo viene distrutto ed il gruppo non esiste più (si veda la sezione 5.3).

L'elezione di un nuovo supervisore può essere iniziata dall'applicazione mediante chiamata al metodo startSupervisorElection(String) della classe A3Node, dove il parametro è il nome del gruppo di cui bisogna eleggere il nuovo supervisore. L'elezione di un nuovo supervisore è anche iniziata dal Service quando il precedente supervisore ha lasciato il gruppo, volontariamente o meno. In entrambi i casi, il Service spedisce un messaggio broadcast di richiesta di valori di fitness function. Quando lo ricevono, i canali A3Channel calcolano la fitness function usando la

formula contenuta nel loro descrittore, estensione della classe GroupDescriptor, e la comunicano al Service. Su di esso è presente un oggetto di classe FitnessFunctionManager, che è utilizzato per raccogliere e riordinare dal migliore al peggiore i valori di fitness function ricevuti entro un intervallo di tempo prefissato. Allo scadere di tale intervallo di tempo, è determinato l'indirizzo del canale A3Channel col valore di fitness function migliore. Tale indirizzo è infine comunicato broadcast ai canali A3Channel, che, se è il caso, disattiveranno il loro ruolo attuale per attivare quello nuovo.

Messaggi analoghi, ma indicanti anche il nome del gruppo di cui si sta eleggendo il supervisore, sono scambiati in contemporanea anche con il gruppo "wait". I dispositivi A3Node connessi al gruppo “wait” determinano se c'è un canale A3Channel in stato di attesa che possa diventare supervisore del gruppo indicato nel messaggio ricevuto. In caso affermativo, determinano il valore della fitness function di tale canale A3Channel ricavandolo dal descrittore e lo comunicano al Service del gruppo in questione, dove sarà trattato come i valori provenienti da canali A3Channel che già fanno parte del gruppo stesso. Alla ricezione del messaggio contenente l'indirizzo del nuovo supervisore, i canali A3Channel connessi al gruppo "wait" confrontano tale indirizzo con il proprio, con il quale avevano risposto al Service:

- se gli indirizzi coincidono, allora il canale A3Channel verso l'altro gruppo è stato eletto supervisore, può diventare supervisore ed è quindi risvegliato.
- se gli indirizzi non coincidono, ma sul dispositivo A3Node è presente un canale A3Channel in attesa di diventare follower dell’altro gruppo, tale canale A3Channel è essere risvegliato.
- se gli indirizzi non coincidono, però sul dispositivo A3Node non è presente un canale A3Channel in attesa per il gruppo in questione, allora non succede nulla;
- non si può mai presentare il caso in cui gli indirizzi coincidono, sul dispositivo A3Node è presente un canale A3Channel in attesa per il gruppo in questione, ma esso non può diventarne supervisore: in tal caso, infatti, il valore della fitness function non è spedito al Service quando richiesto.

La comunicazione tra il Service di un qualsiasi gruppo e quello del gruppo "wait" avviene attraverso un oggetto di classe A3UnicastTransmitter che si connette al gruppo destinatario del messaggio.

Ho reso la classe FitnessFunctionManager disponibile all'uso da parte dei progettisti di applicazioni A-3. I valori di fitness function da essa gestiti devono essere però interi. I valori considerati migliori sono quelli più grandi. Se due valori sono uguali, quello considerato migliore è quello ricevuto prima. La classe FitnessFunctionManager deve essere usata in un oggetto che implementi l'interfaccia TimerInterface in modo tale che il proprio metodo timerFired(int) contenga una chiamata all'istruzione getBest(int), dove il parametro intero è il numero di identificativi dei canali A3Channel da recuperare. Il metodo timerFired(int) è chiamato automaticamente dopo che i risultati sono stati raccolti e posti in un vettore ordinato in cui il primo elemento è il migliore e l'ultimo è il peggiore. Nell'oggetto con interfaccia TimerInterface, la raccolta delle fitness function comincia con l'invocazione del metodo startCollectingFitnessFunctions(int) dell'oggetto di classe FitnessFunctionManager.

5.8 La classe A3Message

I messaggi scambiati tra i nodi di un sistema A-3 sono oggetti di classe A3Message. Si compongono, nell'ordine, di indirizzo del mittente, tipo di messaggio e contenuto del messaggio stesso. Il primo è una stringa corrispondente allo unique name assegnato al nodo da AllJoyn in fase di unione alla sessione. Il secondo è un intero. Se tale intero appartiene all'intervallo [0, 30], allora è definito nella classe Constants ed identifica il tipo di un messaggio di sistema, trasparente all'utilizzatore di A-3. In caso contrario, esso identifica un messaggio applicativo, non è filtrato da A-3 e giunge al ruolo correntemente attivo, dove il progettista avrà implementato la logica per processarlo opportunamente. Qualsiasi messaggio applicativo con tipo intero

compreso nell'intervallo [0, 30] sarà interpretato da A-3 come un messaggio di sistema generando malfunzionamenti. Il contenuto del messaggio è invece una stringa. Un messaggio è creato tramite il costruttore A3Message(int, String), dove il parametro intero è il tipo di messaggio e il parametro stringa è il contenuto del messaggio stesso. Per soddisfare le esigenze di AllJoyn è presente anche il costruttore A3Message(), che però non è mai invocato da A-3, né deve essere invocato dall'applicazione A-3, in quanto non esegue nulla. Altro vincolo imposto da AllJoyn è che gli attributi della classe da trasmettere sul bus siano pubblici, cosa che ha reso inutile l'introduzione di getter e setter per tali attributi. La classe A3Message può essere estesa per trasportare ulteriori informazioni definite dal progettista dell'applicazione A-3, purché egli si ricordi che la sua estensione sarà usata da AllJoyn durante il marshalling e l'unmarshalling della stessa. L'estensione della classe A3Message dovrà dunque contenere:

- un costruttore pubblico senza parametri, che richiami il costruttore super();
- attributi pubblici etichettati con l'etichetta @Position definita in AllJoyn, a partire da @Position(3).

5.9 Come creare un'applicazione A3Droid

Le mie API si compongono di una parte visibile al progettista di un sistema A-3 e di una parte che invece gli è trasparente. In figura 23 a pagina seguente si vede chiaramente questa distinzione. Per creare un'applicazione A-3 è dunque sufficiente conoscere le classi visibili.

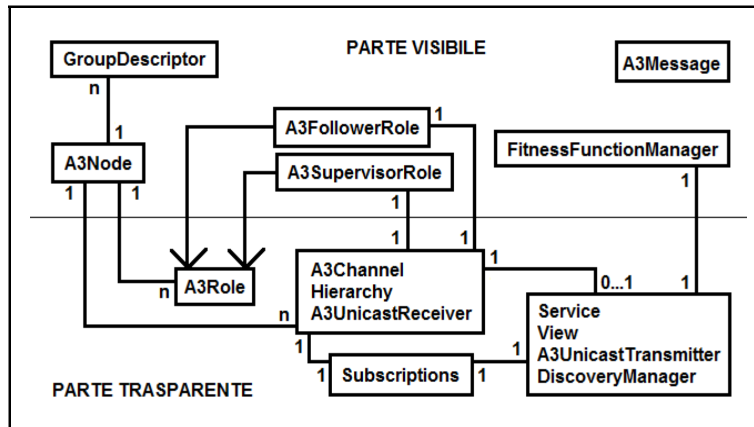

Figura 23 Diagramma delle classi delle API da me realizzate.

I passi necessari per realizzare un'applicazione A-3 utilizzando le API da me realizzate sono i seguenti:

1) Definire tutti i ruoli che possono essere presenti nel sistema.

2) Implementare ciascuno di essi con un'estensione della classe A3FollowerRole se il ruolo è un comportamento tenuto da un follower, o A3SupervisorRole se è un comportamento tenuto dal supervisore.

3) Definire tutti i gruppi che possono essere presenti nel sistema ed estendere la classe GroupDescriptor per crearne i descrittori.

Il costruttore dell'estensione della classe GroupDescriptor deve chiamare il costruttore "super(String, String, String)". Il primo parametro è il nome del gruppo, mentre gli altri sono i nomi estesi delle classi che implementano i ruoli del gruppo: supervisore il secondo, follower il terzo.

4) Definire tutti i possibili tipi di messaggi applicativi che i nodi potranno scambiarsi.

5) Estendere le classi A3Message e A3Node se necessario.

6) Creare un'Activity Android che estenda la classe A3DroidActivity. Quest'ultima implementa l'interfaccia UserInterface, carica la libreria nativa di AllJoyn con il metodo "static {System.loadLibrary("alljoyn_java");}" ed esegue l'istruzione "org.alljoyn.bus.alljoyn.DaemonInit.PrepareDaemon(getApplicationContext());" che lancia il demone AllJoyn.

7) Sovrascriverne il metodo onCreate(Bundle) in modo che:

- siano in esso lanciati due thread diversi per la gestione, rispettivamente, dei comandi provenienti dall'interfaccia grafica e dei messaggi diretti dall'applicazione all'interfaccia grafica;
- sia in esso creato un nodo mediante il costruttore A3Node(UserInterface, ArrayList<String>, ArrayList<GroupDescriptor>) o il costruttore equivalente di un’eventuale estensione della classe A3Node. Il primo parametro è la classe che deve alterare l'interfaccia grafica in reazione ad un messaggio. Il secondo è la lista dei nomi estesi delle classi che implementano i ruoli in possesso del dispositivo. Il terzo è la lista dei descrittori di tutti i gruppi che possono essere creati nel sistema.

6 Test

Dopo aver realizzato le API di A3Droid, ho controllato che esse realizzassero correttamente le funzioni richieste di creazione, distruzione, gestione e composizione di gruppi. Per farlo, ho realizzato l'applicazione A3Test_1, che non ho tuttavia progettato con lo scopo di effettuare misurazioni. Non ho quindi potuto ricavare da essa alcun dato circa le prestazioni di A3Droid.

Ho pensato che le prestazioni di A3Droid potessero essere influenzate dai seguenti fattori:

- numero di telefoni presenti nel sistema;
- numero di gruppi presenti nel sistema;
- numero di telefoni in un gruppo;
- numero di gruppi a cui può essere connesso un telefono;
- dimensione dei messaggi scambiati tra i telefoni;

e che dunque fosse necessaria una misura della scalabilità di A-3. Ho ricavato tale misura realizzando ed utilizzando l'applicazione A3Test_3.

Ho poi realizzato le applicazioni A3Test_5 ed A3Test_7 per valutare, rispettivamente:

- tempo necessario ai follower per rispondere ai messaggi che il supervisore invia loro utilizzando i diversi tipi di comunicazione possibili in A-3 (unicast, multicast e broadcast);
- tempo necessario affinché tutti i follower rispondano ad un messaggio di inizio di elezione di un nuovo supervisore.

Ho progettato le applicazioni A3Test_3, A3Test_5 ed A3Test_7 in modo tale che tutti i telefoni presenti nel sistema risultino connessi a tutti i gruppi creati dall'utente in numero arbitrario, e che l'utente non abbia possibilità di scegliere arbitrariamente il supervisore di ogni gruppo. Ho poi misurato le prestazioni di A3Droid eseguendo le suddette applicazioni su:

- un cellulare Samsung Galaxy Star, usato come access point Wi – Fi a cui ho connesso tutti gli altri telefoni;
- due cellulari Samsung Galaxy S4;
- due cellulari Google Nexus 5.

Ricavando poi i grafici dai dati raccolti, sono emerse apparenti anomalie, che hanno evidenziato come vi sia un altro fattore che influenza pesantemente le prestazioni di A3Droid: la distribuzione dei supervisori sui telefoni. Per averne le prove, ho dunque modificato l'applicazione A3Test_5 nel modo in cui la presenterò in seguito, dando la possibilità all'utente di scegliere arbitrariamente il supervisore di ogni gruppo.

Nella sezione 6.1 descriverò l'applicazione A3Test_1. Nelle sezioni 6.2, 6.3 e 6.4 illustrerò, rispettivamente, le applicazioni A3Test_3, A3Test_5 ed A3Test_7, insieme ai grafici ricavati dal loro utilizzo. Tali grafici risentono delle anomalie derivate dalla mancata considerazione della distribuzione dei supervisori sui telefoni al momento dell'esecuzione delle applicazioni. Nella sezione 6.5, infine, riassumerò i risultati ottenuti, mostrerò come la distribuzione dei supervisori influenza le prestazioni di A3Droid e presenterò possibili soluzioni per migliorare le prestazioni e la scalabilità di quest'ultimo.

6.1 A3Test_1

Scopo dell'applicazione A3Test_1 è verificare il corretto funzionamento delle API A3Droid da me realizzate.

Per far fronte allo scarso numero di telefoni inizialmente disponibili, ho realizzato l'applicazione in modo tale da simulare il comportamento di tre telefoni su un telefono solo. L'applicazione è composta dunque da tre nodi A3Node uguali ed indipendenti tra loro. E' possibile creare tre gruppi denominati "a", "b" e "c", dotati tutti dello stesso descrittore (classe a3.test1.AppGroupDescriptor) e quindi degli

stessi ruoli di supervisore (classe a3.test1.SupRole) e follower (classe a3.test1.FolRole). La loro logica consiste nella visualizzazione di un singolo messaggio sullo schermo, al fine di verificare la loro corretta attivazione e disattivazione. Tutti e tre i nodi possono connettersi a tutti e tre i gruppi. In caso ciò non andasse bene, è necessario agire sul codice per rimuovere alcuni ruoli da alcuni nodi: ho fatto così per verificare il corretto comportamento del gruppo "wait" e delle API di composizione dei gruppi in tale situazione. L'interfaccia grafica consiste in un'Activity Android principale (classe a3.test1.Test1Activity) che crea tre schede denominate "1OUT", "2OUT" e "3OUT". Il contenuto di ogni scheda è definito da un'altra Activity Android e consiste in un insieme di caselle di testo e di pulsanti tramite i quali è possibile invocare i metodi di A3Droid con i parametri corretti. Il contenuto di tutte le schede è uguale, ma per garantire il corretto funzionamento dell'applicazione ho dovuto creare tre Activity Android differenti (classi a3.test1.ActivityOne, a3.test1.ActivityTwo e a3.test1.ActivityThree). Al momento della loro creazione, ciascuna di esse crea un nodo A3Node che le risulterà univocamente legato. La figura 24 a pagina seguente mostra le due schermate caratteristiche dell'applicazione A3Test_1Sim, proprie di ciascuna scheda. In ognuna delle due schermate si notano le schede corrispondenti ai telefoni simulati ed il pulsante "Handle", che consente di passare da una schermata all'altra.

Figura 24 L'interfaccia grafica dell'applicazione A3Test_1.

La schermata di destra è una grande casella di testo, in cui vengono visualizzati i messaggi provenienti dal nodo A3Node associato alla scheda. La schermata di sinistra è invece quella in cui è possibile invocare le API di A3Droid. All'interno di ogni scheda, le caselle di testo sono denominate "par1", "par2" e "par3", come mostrato in figura 24. Inserendo opportunamente in esse i parametri e premendo i pulsanti, è possibile invocare i metodi forniti dalla classe A3Node di A3Droid. Nel dettaglio, le funzioni di ciascun pulsante sono le seguenti:

- pulsante "C/D" => connect(<par2>, false, true) se il nodo non è connesso al gruppo <par2>, disconnect(<par2>, true) se il nodo è invece connesso a tale gruppo;
- pulsante "I" => sendToSupervisor(new A3Message(50, <par1>), <par2>);
- pulsante "I50" => sendMulticast(new A3Message(50, <par1>), <par2>);
- pulsante "I51" => sendMulticast(new A3Message(51, <par1>), <par2>);
- pulsante "S50" => subscribe(50, <par2>);
- pulsante "S51" => subscribe(51, <par2>);
- pulsante "U50" => unsubscribe(50, <par2>);

- pulsante “U51” => unsubscribe(51, <par2>);
- pulsante “St” => stack(<par2>, <par3>);
- pulsante “P” => peers(<par2>, <par3>);
- pulsante “H” => hierarchy(<par1>, <par2>, <par3>);
- pulsante “M” => merge(<par2>, <par3>);
- pulsante “Rst” => reverseStack(<par2>, <par3>);
- pulsante “Rp” => reversePeers(<par2>, <par3>);
- pulsante “Rh” => reverseHierarchy(<par1>, <par2>, <par3>);
- pulsante “S” => split(<par2>, <par3>).

Dalle prove effettuate su un singolo telefono con questa applicazione emerge che A3Droid funziona correttamente. La correttezza del funzionamento è stata poi verificata con successo anche distribuendo l’applicazione su più telefoni.

6.2 A3Test_3

Scopo dell’applicazione A3Test_3 è studiare la variazione delle prestazioni di A3Droid al variare dei seguenti fattori:

- numero di telefoni presenti nel sistema;
- dimensione dei messaggi scambiati dai telefoni nel sistema;
- numero di gruppi presenti nel sistema.

Ho dunque progettato l’applicazione A3Test_3 in modo da variare manualmente questi fattori e da effettuare il numero di prove ritenuto opportuno per validare i risultati.

Un esperimento alla base dell’applicazione A3Test_3 consiste nel creare uno o più gruppi in cui i follower spediscono messaggi di ping al supervisore, il quale risponde a ciascuno di essi separatamente con messaggi unicast. Tramite differenza di timestamp, i follower riescono a misurare il tempo intercorso tra la spedizione del messaggio e la sua ricezione. Quando tale tempo supera la soglia pari ad 1s, allora l’esperimento termina e sono raccolti in un unico file:

- numero di messaggi spediti;
- durata dell'esperimento;
- frequenze medie di spedizione dei messaggi.

Tali dati sono ricavati su ogni follower in ogni gruppo, dunque lanciare un esperimento n volte equivale ad eseguire lo stesso esperimento n * numero_gruppi * (numero_telefoni – 1) volte.

All'avvio dell'applicazione A3Test_3, un telefono si connette automaticamente ad un gruppo denominato "control", che ha la funzione di contare i dispositivi che prendono parte all'esperimento e di sincronizzare inizio e fine dell'esperimento su tutti i telefoni. Nel frattempo, si presenta l'interfaccia grafica illustrata in figura 25. Nella casella di testo indicata con il simbolo "*" è possibile inserire il numero identificativo della dimensione dei messaggi che si vuole siano scambiati dai telefoni presenti nel sistema:

- numero 1: messaggi di 62484 Byte;
- numero 2: messaggi di 32000 Byte;
- numero 3: messaggi di 5017 Byte;
- numero 4: messaggi di 1812 Byte.

Tali dimensioni sono compatibili con il limite imposto da AllJoyn e corrispondono a dimensioni di foto dei formati più comuni nei social network. Dopo aver avviato l'applicazione su tutti i telefoni desiderati, premendo il pulsante "Create group" su uno qualsiasi di essi, è possibile connettere tutti i telefoni ad un gruppo denominato "A3Test3_i_n", dove "i" è il numero che identifica la dimensione dei messaggi ed "n" è un numero progressivo identificativo del gruppo stesso. Una volta creati tutti i gruppi desiderati, è possibile iniziare un esperimento premendo il pulsante "Start trial". L'esperimento termina automaticamente, oppure può essere terminato manualmente premendo il pulsante "Stop trial". E' possibile ripetere nuovamente lo stesso esperimento premendo nuovamente il pulsante "Start trial", senza dover nuovamente riconnettere i telefoni né ricreare nuovamente tutti i gruppi.

Figura 25 L'interfaccia grafica dell'applicazione A3Test_3.

L'interfaccia grafica dell'applicazione A3Test_3 è costituita dalla classe "a3.test3.Test3Activity". Il descrittore del gruppo "control" è rappresentato dalla classe "a3.test3.ControlDescriptor", che definisce come ruolo di supervisore la classe "a3.test3.ControlSupervisorRole" e come ruolo di follower la classe "a3.test3.ControlFollowerRole". I gruppi "A3Test3_i_n" hanno come descrittore la classe "a3.test3.ExperimentDescriptor", che definisce come ruolo di supervisore la classe "a3.test3.ExperimentSupervisorRole" e come ruolo di follower la classe "a3.test3.ExperimentFollowerRole".

Le misurazioni effettuate evidenziano un limite nel numero di gruppi effettivamente creabili all'interno del sistema: aumentare il numero di gruppi oltre questo limite peggiorerebbe le prestazioni del sistema impedendone l'usabilità. Tali limiti consistono in:

- 5 gruppi di 2 telefoni;
- 3 gruppi di 3 telefoni;
- 2 gruppi di 4 telefoni;
- 2 gruppi di 5 telefoni.

I grafici mostrano chiaramente come sia l'introduzione di un terzo telefono a causare un calo delle prestazioni più marcato: la sessione AllJoyn corrispondente al gruppo A-3 passa ad essere da singlepoint a multipoint, necessitando quindi di una gestione più complessa.

I grafici dall'1 al 12 mostrano i dati ottenuti al variare del numero di gruppi (in ascissa) e di telefoni, mantenendo ogni volta fissa la dimensione dei messaggi. Tali dati mostrano che il numero medio di messaggi spediti, la durata media dell'esperimento e la frequenza media di spedizione dei messaggi tendono a diminuire all'aumentare del numero di telefoni e di gruppi. Questo a causa del maggior traffico di messaggi presente nel sistema.

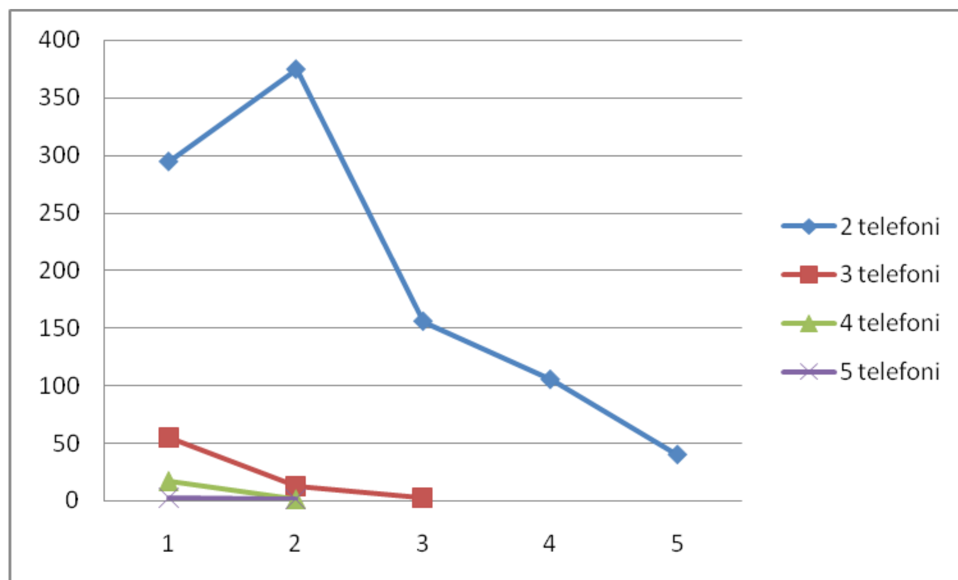

Grafico 1 Numero medio di messaggi da 62484 Byte (in ordinata) scambiati tra i nodi in funzione del numero di gruppi (in ascissa) e di telefoni presenti nel sistema.

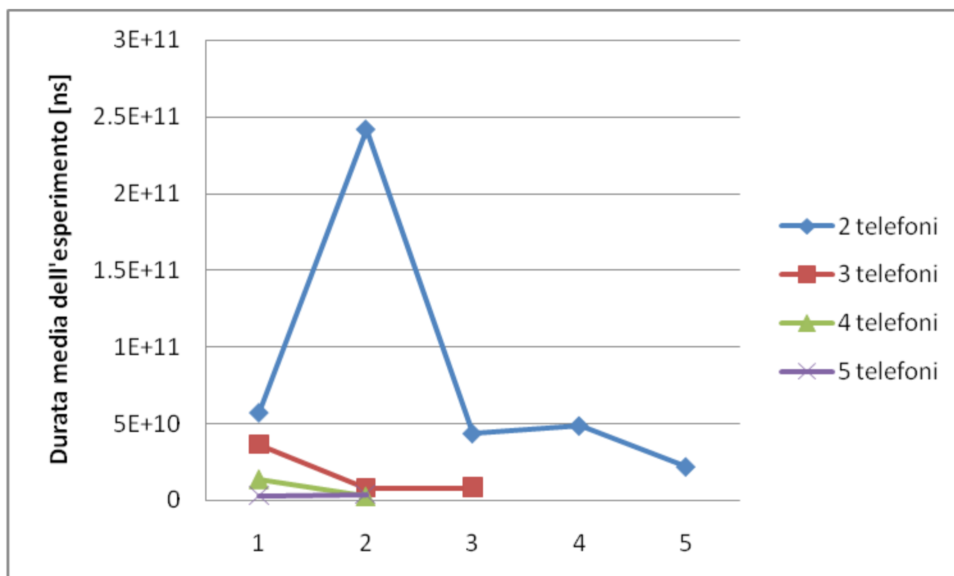

Grafico 2 Durata media dell'esperimento (espressa in ns, in ordinata) in funzione del numero di gruppi (in ascissa) e di telefoni presenti nel sistema. La dimensione dei messaggi scambiati è 62484 Byte.

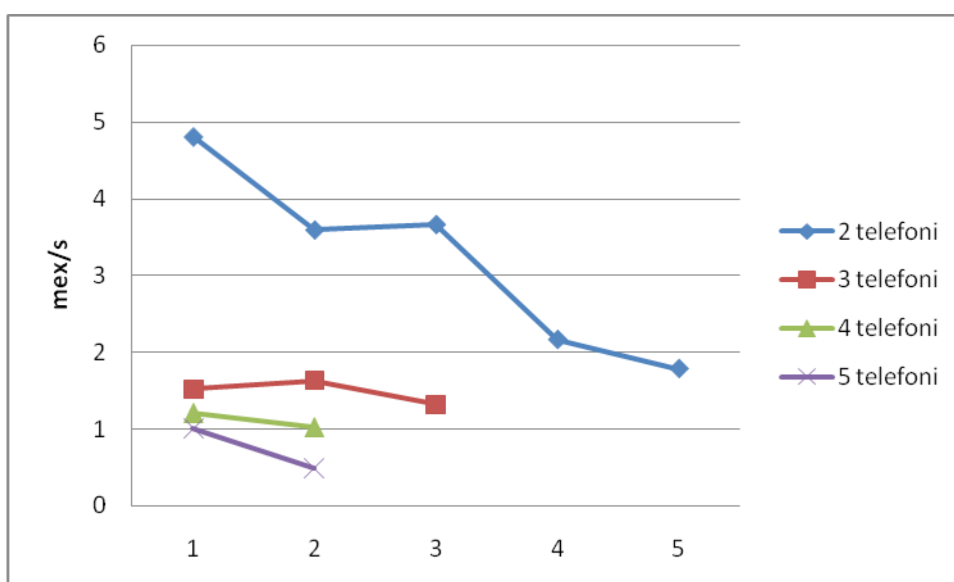

Grafico 3 Frequenza media di spedizione di messaggi da 62484 Byte (in ordinata, espressa in mex/s) in funzione del numero di gruppi (in ascissa) e di telefoni presenti nel sistema.

I grafici sul numero medio di messaggi spediti e sulla durata media dell'esperimento presentano un'anomalia nel caso in cui due telefoni sono connessi a due gruppi:

durando di più l'esperimento, risulta essere maggiore anche il numero di messaggi scambiati. La frequenza di spedizione dei messaggi tra due telefoni risulta comunque essere coerente, in quanto cala all'aumentare del numero di gruppi presenti nel sistema. Questo perché l'introduzione di un nuovo gruppo nel sistema provoca un maggiore traffico di messaggi in rete.

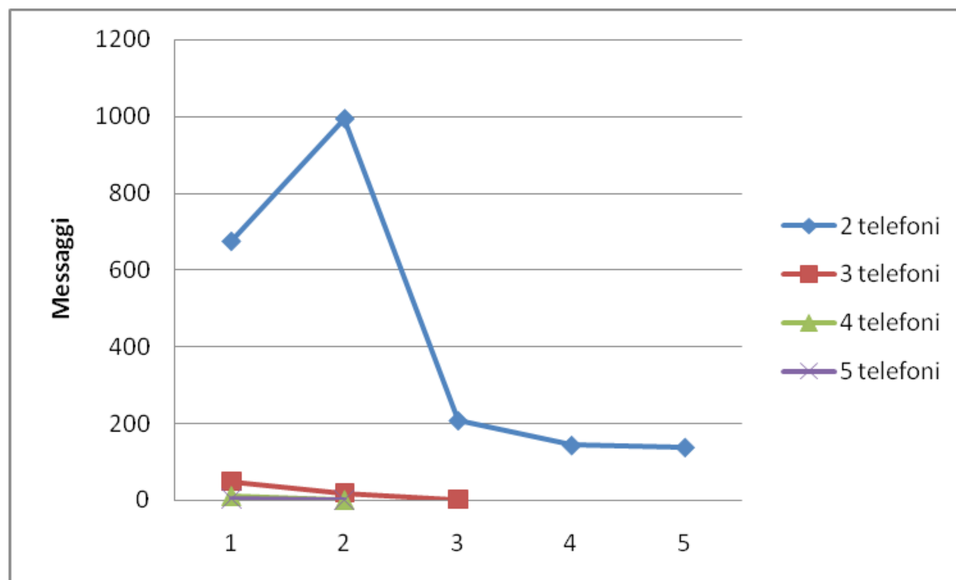

Grafico 4 Numero medio di messaggi da 32000 Byte (in ordinata) scambiati tra i nodi in funzione del numero di gruppi (in ascissa) e di telefoni presenti nel sistema.

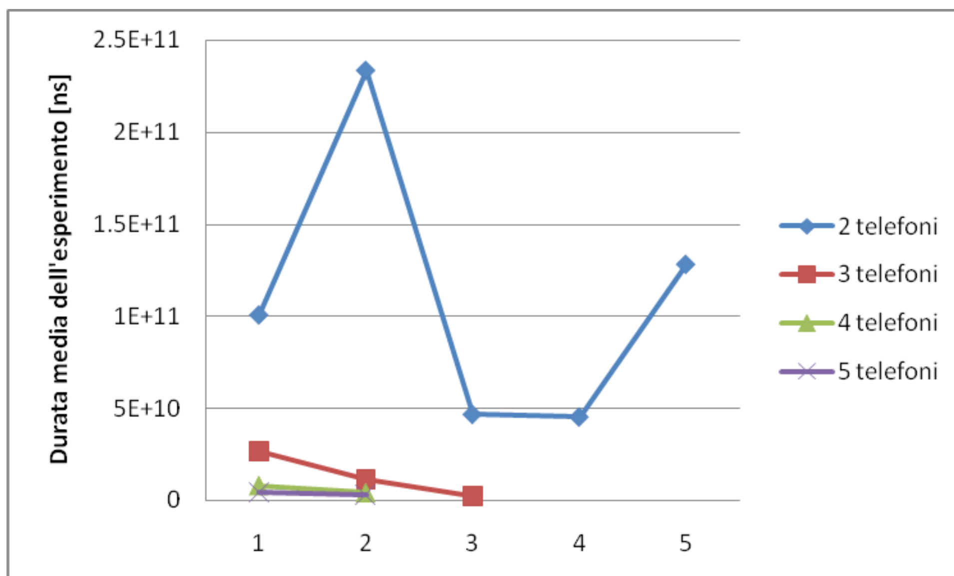

Grafico 5 Durata media dell'esperimento (espressa in ns, in ordinata) in funzione del numero di gruppi (in ascissa) e di telefoni presenti nel sistema. La dimensione dei messaggi scambiati è 32000 Byte.

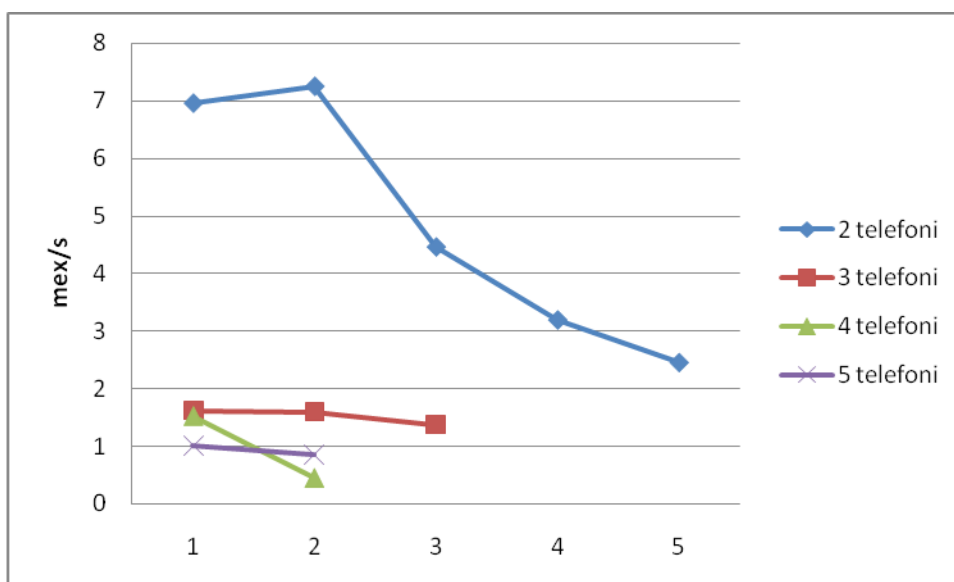

Grafico 6 Frequenza media di spedizione di messaggi da 32000 Byte (in ordinata, espressa in mex/s) in funzione del numero di gruppi (in ascissa) e di telefoni presenti nel sistema.

Come nel caso precedente, i grafici del numero medio di messaggi spediti e della durata media dell'esperimento risentono di un'anomalia nel caso della presenza di due telefoni e due gruppi: anche in questo caso, l'esperimento dura di più e dunque

sono spediti più messaggi. Un'altra anomalia è presente nel caso in cui due telefoni sono connessi a cinque gruppi: nonostante l'esperimento duri di più che nel caso della presenza di due telefoni e quattro gruppi, sono spediti meno messaggi e la frequenza di spedizione risulta correttamente essere minore. Un'altra apparente anomalia compare nel grafico delle frequenze, in cui le linee associate alla presenza di quattro e cinque telefoni si intersecano: non potendo spedire un numero di messaggi non intero, partendo entrambe le linee da sopra 1 mex/s e finendo entrambe sotto 1 mex/s, le due curve sono equiparabili.

Le durate medie degli esperimenti con messaggi di 32000 Byte risultano molto simili, anomalie a parte, a quelle ottenute con messaggi di 62484 Byte. Ne consegue che, passando da una di queste dimensioni dei messaggi ad un'altra, l'utente non percepisce particolari rallentamenti dell'applicazione. I messaggi da 32000 Byte sono però più piccoli, richiedono meno tempo per essere spediti e ricevuti, e dunque aumentano il numero dei messaggi spediti e la frequenza della loro spedizione.

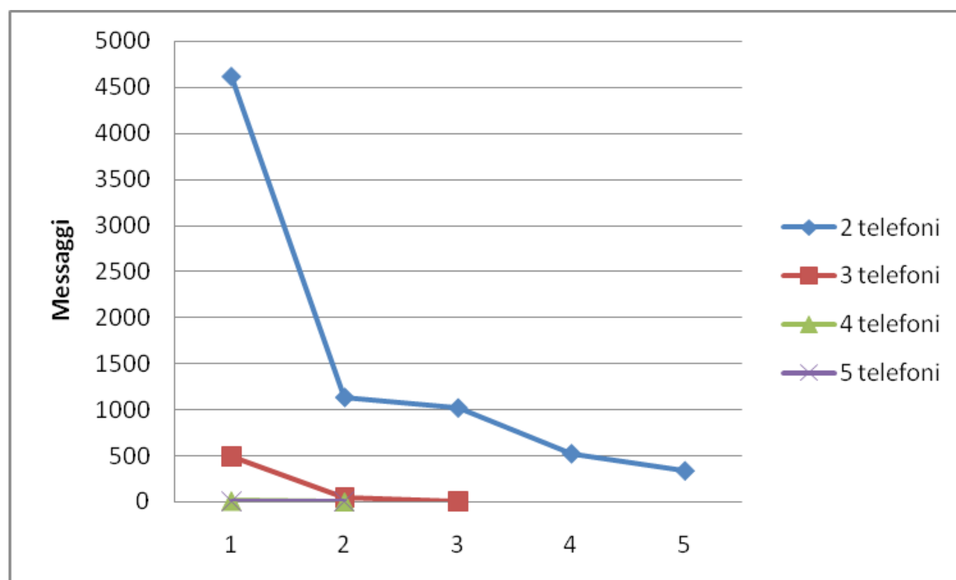

Grafico 7 Numero medio di messaggi da 5017 Byte (in ordinata) scambiati tra i nodi in funzione del numero di gruppi (in ascissa) e di telefoni presenti nel sistema.

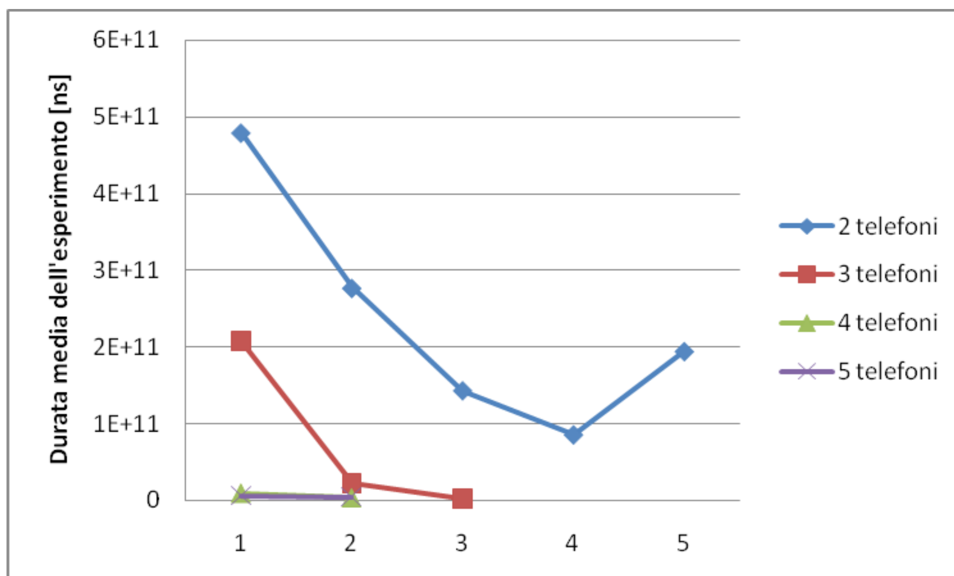

Grafico 8 Durata media dell'esperimento (espressa in ns, in ordinata) in funzione del numero di gruppi (in ascissa) e di telefoni presenti nel sistema. La dimensione dei messaggi scambiati è 5017 Byte.

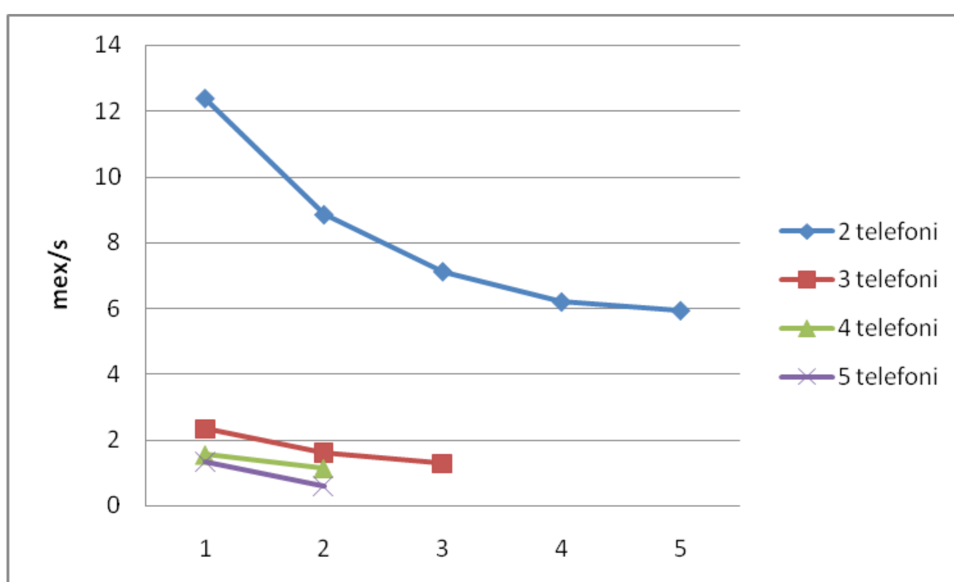

Grafico 9 Frequenza media di spedizione di messaggi da 5017 Byte (in ordinata, espressa in mex/s) in funzione del numero di gruppi (in ascissa) e di telefoni presenti nel sistema.

Il grafico delle durate medie dell'esperimento presenta un'anomalia nel caso della presenza di due telefoni e cinque gruppi. Infatti, la durata media dell'esperimento in quel caso è superiore a quella del caso in cui sono presenti due telefoni e quattro

gruppi. Gli altri grafici risultano però privi di anomalie. Rispetto all'esecuzione dell'esperimento con messaggi di 32000 Byte, il numero medio di messaggi spediti, la durata media dell'esperimento e la frequenza media di spedizione dei messaggi sono aumentate. L'aumento è più evidente al diminuire del numero di telefoni presenti nel sistema. Messaggi più piccoli sono spediti in rete e ricevuti dalla rete più in fretta e sono processati più velocemente dal supervisore. Per questo motivo, può aumentare, ed aumenta, la frequenza di spedizione dei messaggi. Tuttavia il supervisore riesce a fronteggiare più a lungo tale aumento di frequenza.

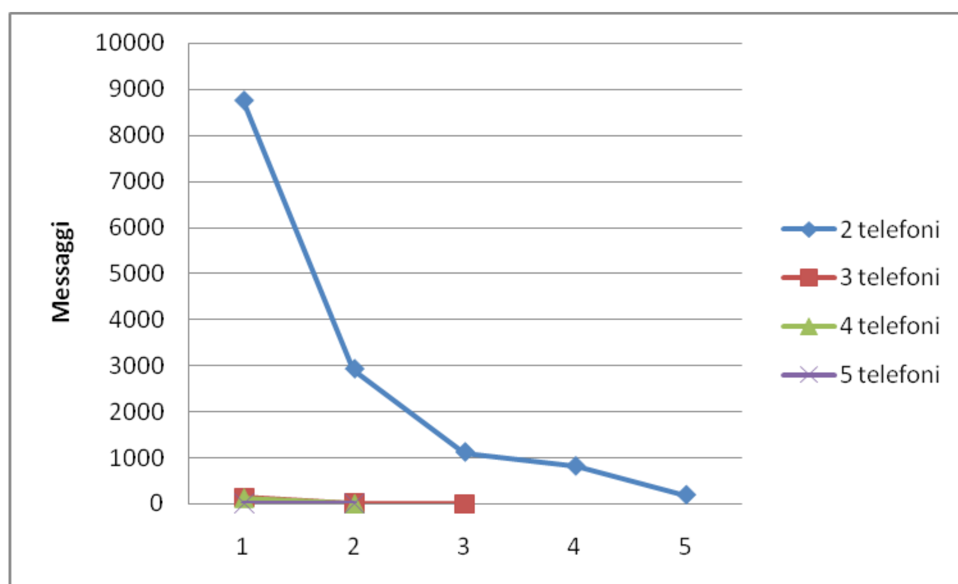

Grafico 10 Numero medio di messaggi da 1812 Byte (in ordinata) scambiati tra i nodi in funzione del numero di gruppi (in ascissa) e di telefoni presenti nel sistema.

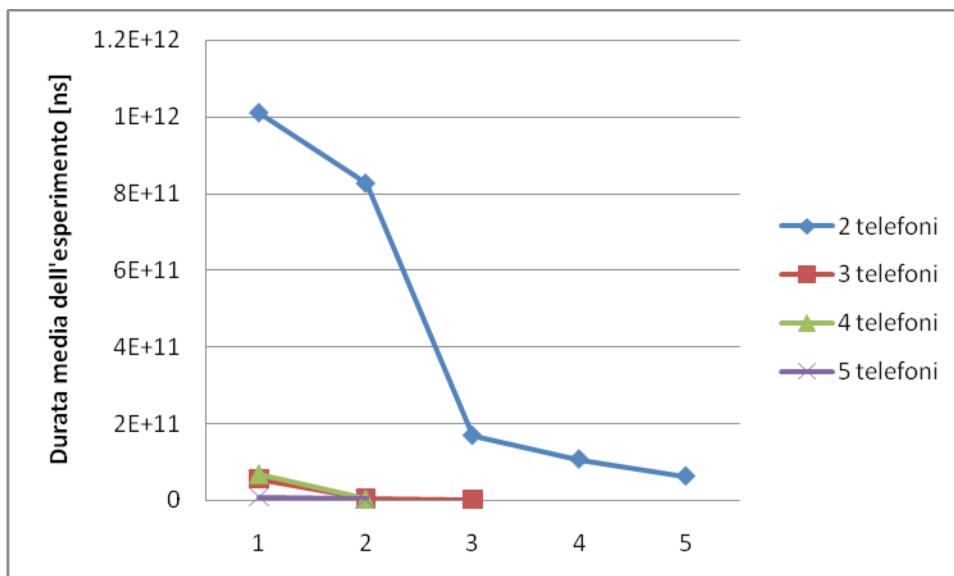

Grafico 11 Durata media dell'esperimento (espressa in ns, in ordinata) in funzione del numero di gruppi (in ascissa) e di telefoni presenti nel sistema. La dimensione dei messaggi scambiati è 1812 Byte.

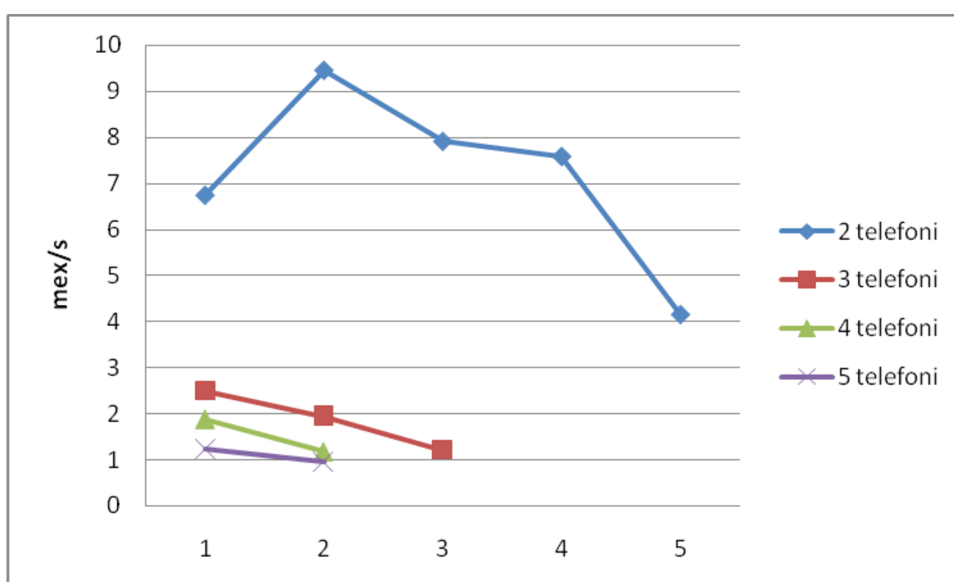

Grafico 12 Frequenza media di spedizione di messaggi da 1812 Byte (in ordinata, espressa in mex/s) in funzione del numero di gruppi (in ascissa) e di telefoni presenti nel sistema.

Il grafico delle frequenze presenta un'anomalia nel caso in cui due telefoni sono connessi a due gruppi, ma risulta essere coerente con gli altri due grafici. Questi ultimi, non presentano invece anomalie. Rispetto all'uso di messaggi da 5017 Byte,

si registra un forte aumento del numero di messaggi spediti, un aumento della durata media dell'esperimento, anche se non così evidente come nel caso precedente, e frequenze di spedizione più o meno simili (anomalie a parte).

I grafici dal 13 al 24 mostrano i dati ottenuti variando il numero di gruppi e la dimensione dei messaggi mantenendo fisso il numero di telefoni presenti nel sistema. Tali dati mostrano che il numero medio di messaggi spediti, la durata media dell'esperimento e la frequenza media di spedizione dei messaggi tendono a diminuire all'aumentare del numero di gruppi e della dimensione dei messaggi scambiati. Questo perché ogni supervisore, oltre a far fronte ad un traffico di messaggi crescente, impiega sempre più tempo per rispondere a messaggi sempre più grandi.

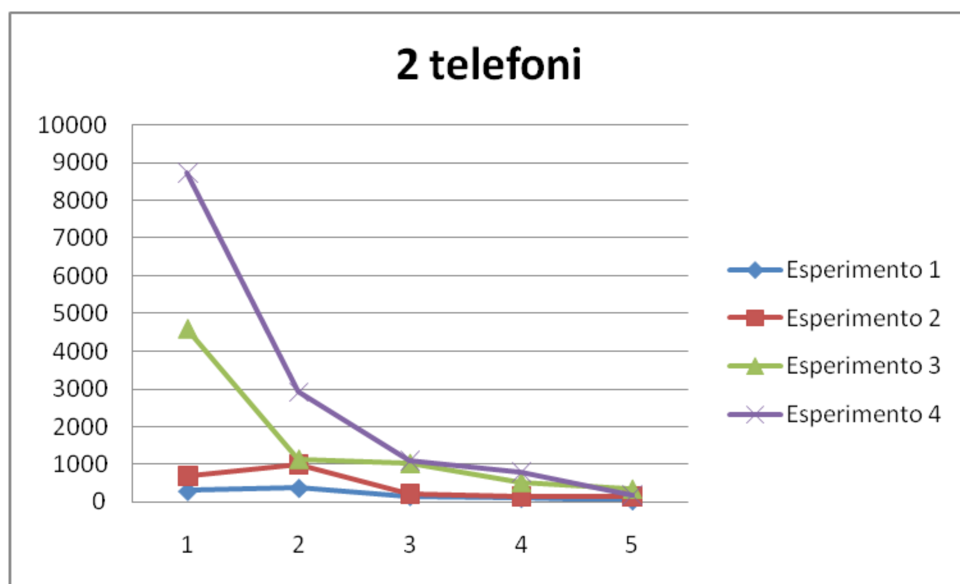

Grafico 13 Numero medio di messaggi spediti da 2 telefoni (in ordinata) in funzione del numero di gruppi (in ascissa) e della dimensione dei messaggi scambiati.

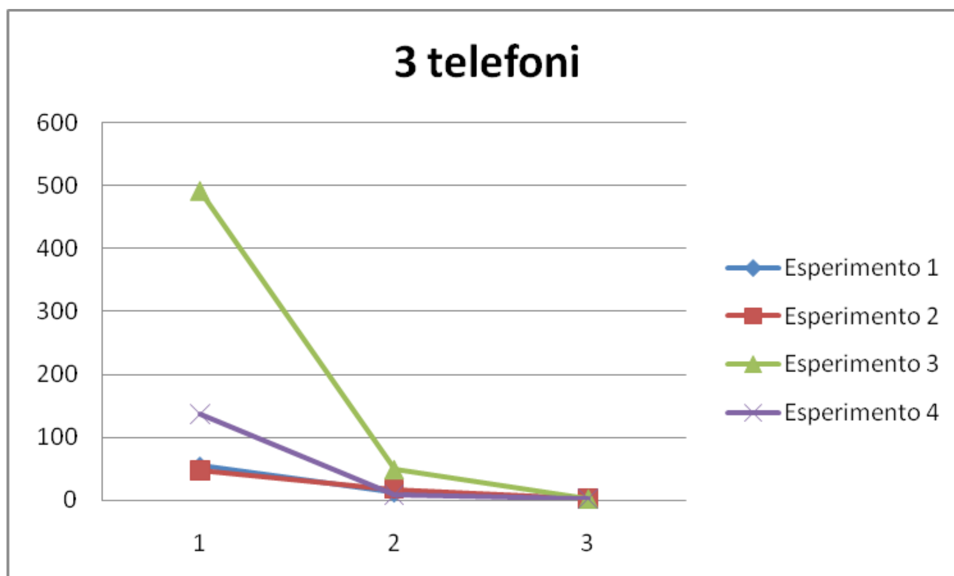

Grafico 14 Numero medio di messaggi spediti da 3 telefoni (in ordinata) in funzione del numero di gruppi (in ascissa) e della dimensione dei messaggi scambiati.

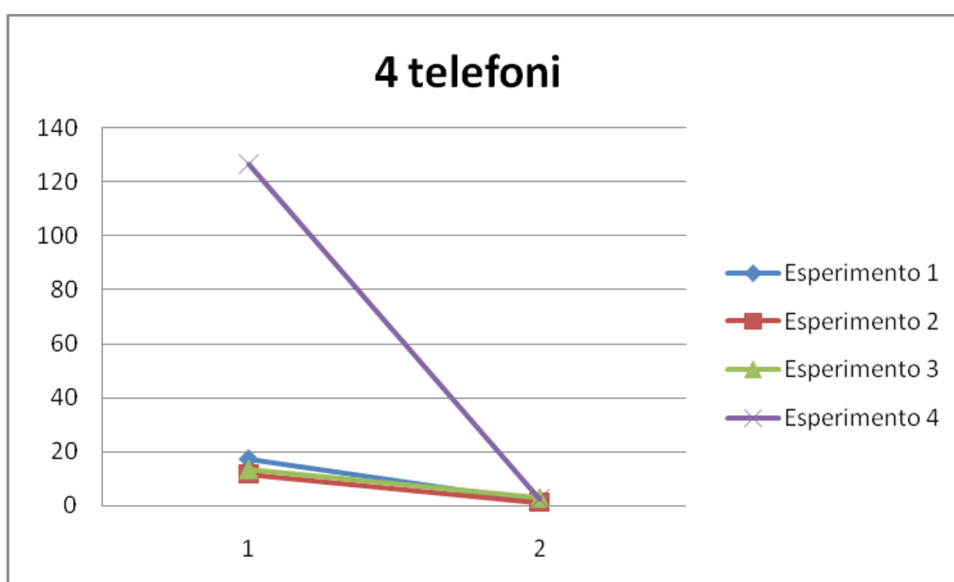

Grafico 15 Numero medio di messaggi spediti da 4 telefoni (in ordinata) in funzione del numero di gruppi (in ascissa) e della dimensione dei messaggi scambiati.

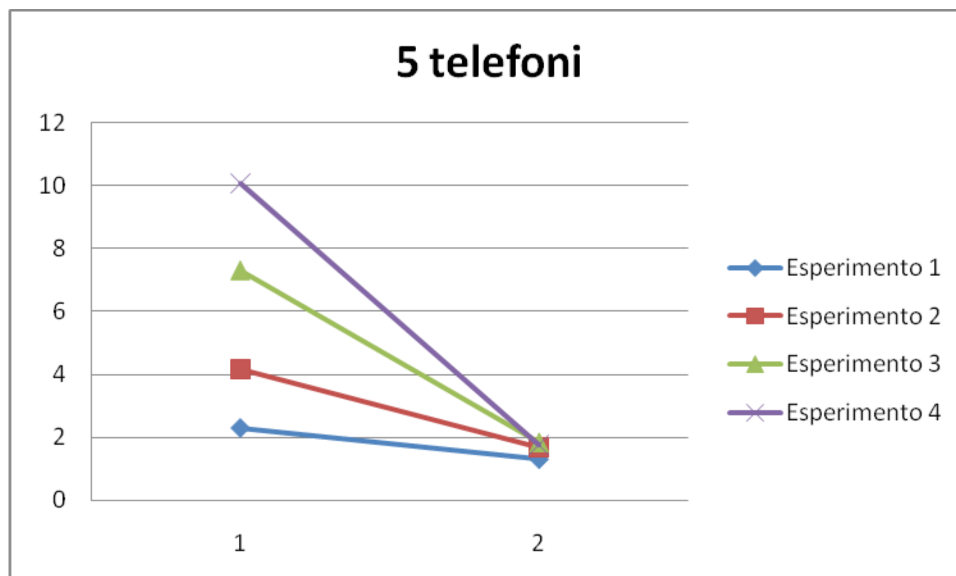

Grafico 16 Numero medio di messaggi spediti da 5 telefoni (in ordinata) in funzione del numero di gruppi (in ascissa) e della dimensione dei messaggi scambiati.

Tenendo fisso il numero di telefoni presenti nel sistema, si nota come, anomalie a parte, il numero di messaggi scambiati diminuisca all'aumentare della loro dimensione, a causa del maggior traffico presente in rete. Il caso ideale è quello illustrato nella figura precedente, in cui si nota anche come tale numero tenda ad azzerarsi all'aumentare del numero di gruppi per lo stesso motivo. Le varie anomalie presenti nei grafici derivano dalle anomalie presenti nei grafici delle durate medie dell'esperimento: durate maggiori significa maggior possibilità di scambio di messaggi.

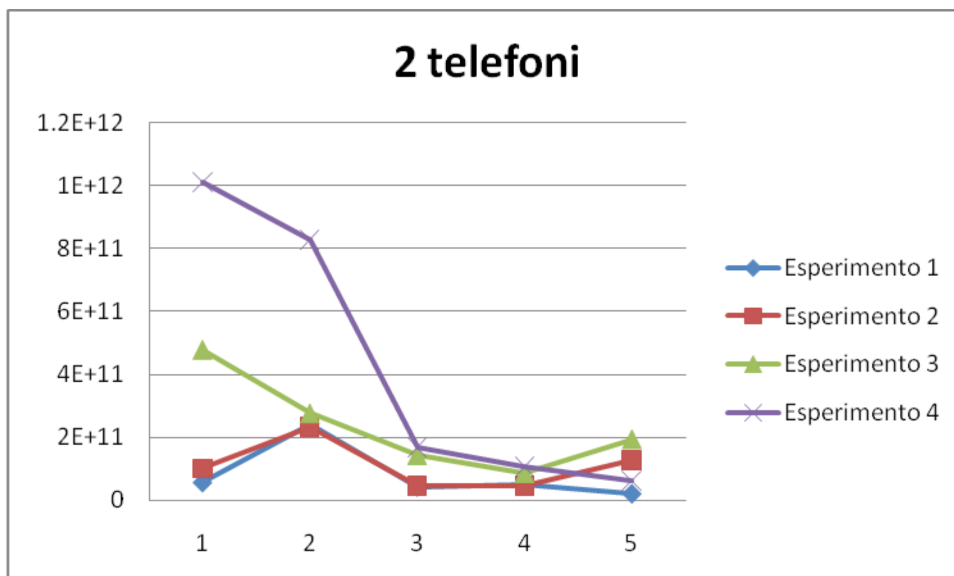

Grafico 17 Durata media dell’esperimento (espressa in ns, in ordinata) su 2 telefoni in funzione del numero di gruppi (in ascissa) e della dimensione dei messaggi scambiati.

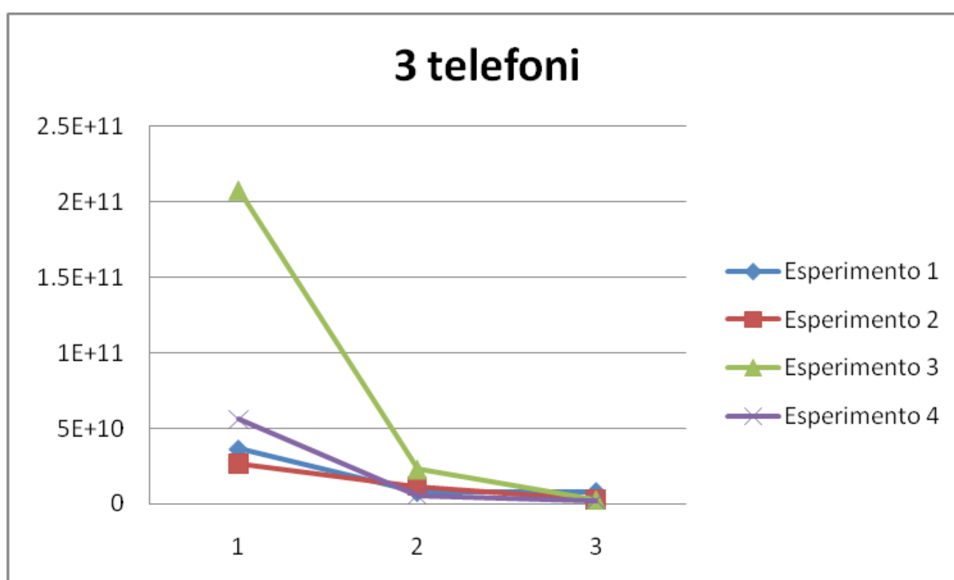

Grafico 18 Durata media dell’esperimento (espressa in ns, in ordinata) su 3 telefoni in funzione del numero di gruppi (in ascissa) e della dimensione dei messaggi scambiati.

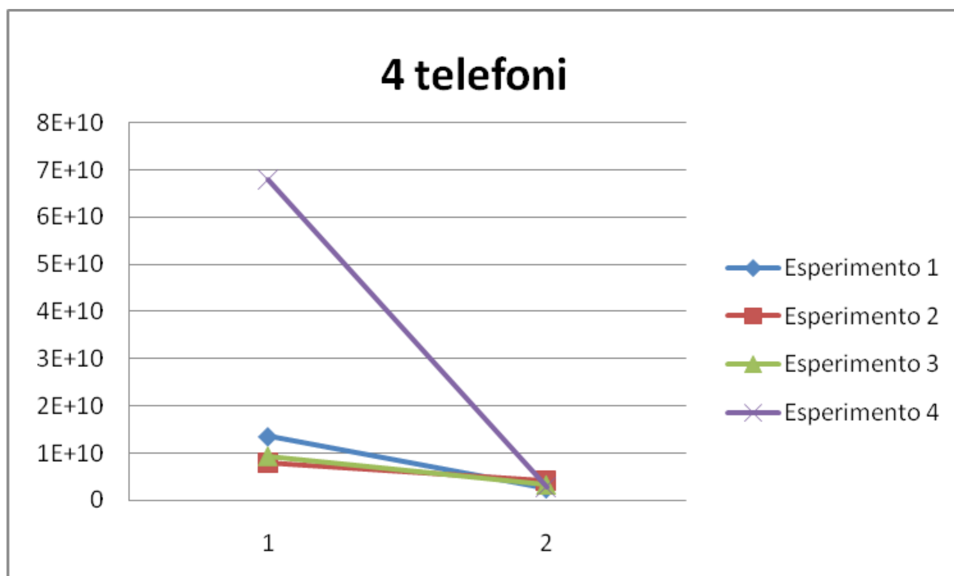

Grafico 19 Durata media dell'esperimento (espressa in ns, in ordinata) su 4 telefoni in funzione del numero di gruppi (in ascissa) e della dimensione dei messaggi scambiati.

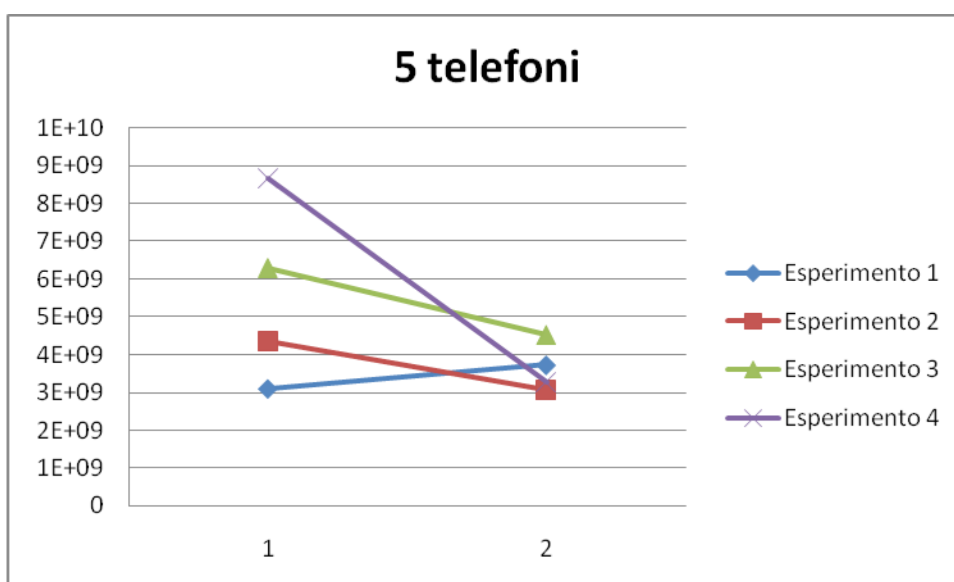

Grafico 20 Durata media dell'esperimento (espressa in ns, in ordinata) su 5 telefoni in funzione del numero di gruppi (in ascissa) e della dimensione dei messaggi scambiati.

Tenendo fisso il numero di telefoni presenti nel sistema, si nota come la durata media dell'esperimento aumenti al diminuire della spedizione dei messaggi: il supervisore impiega meno tempo a processare i messaggi più piccoli ed a rispondere così ai

follower, e la soglia di 1s è raggiunta meno facilmente. Anche la durata media dell'esperimento tende ad azzerarsi all'aumentare del numero di gruppi, a causa del maggior traffico presente in rete.

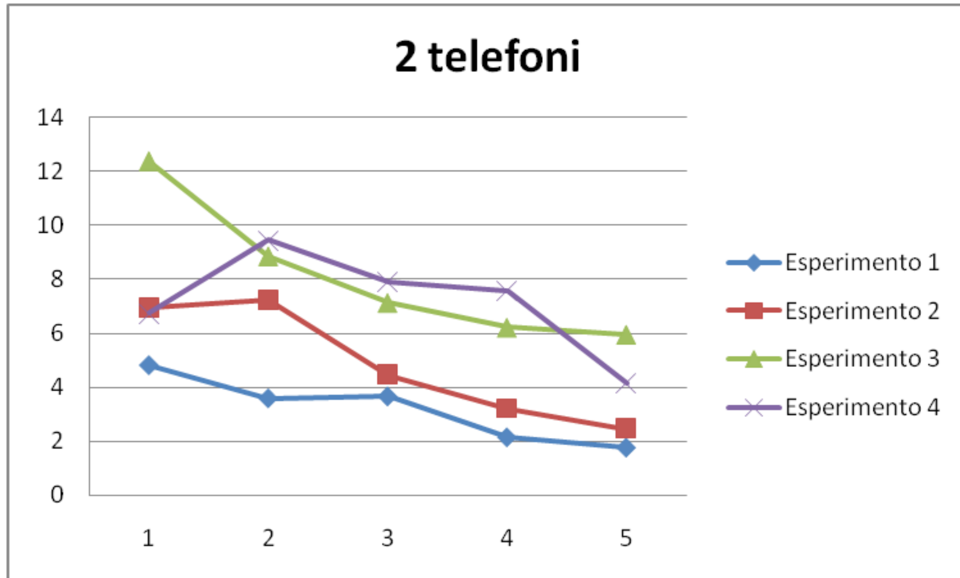

Grafico 21 Frequenza media di spedizione dei messaggi tra 2 telefoni (in ordinata, espressa in mex/s) in funzione del numero di gruppi (in ascissa) e della dimensione dei messaggi scambiati.

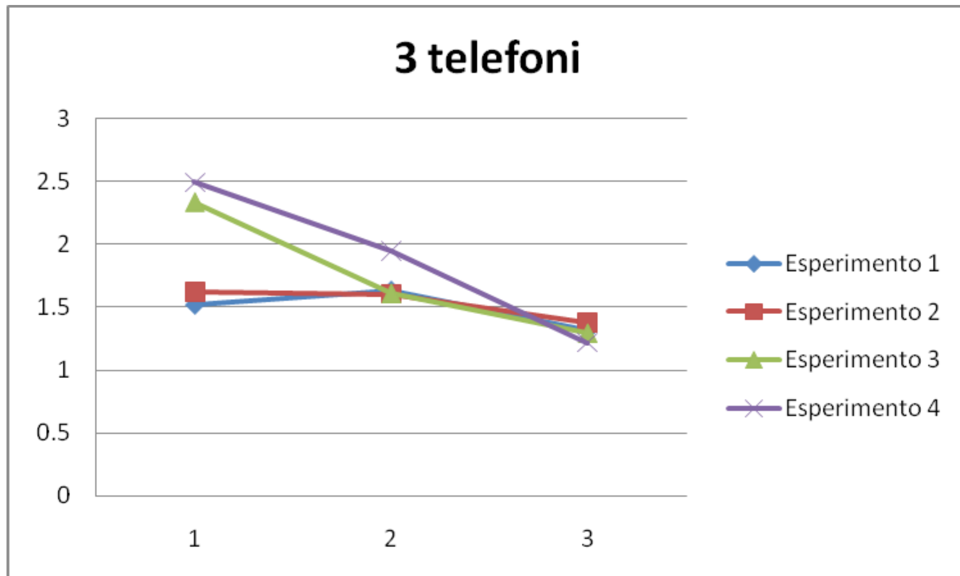

Grafico 22 Frequenza media di spedizione dei messaggi tra 3 telefoni (in ordinata, espressa in mex/s) in funzione del numero di gruppi (in ascissa) e della dimensione dei messaggi scambiati.

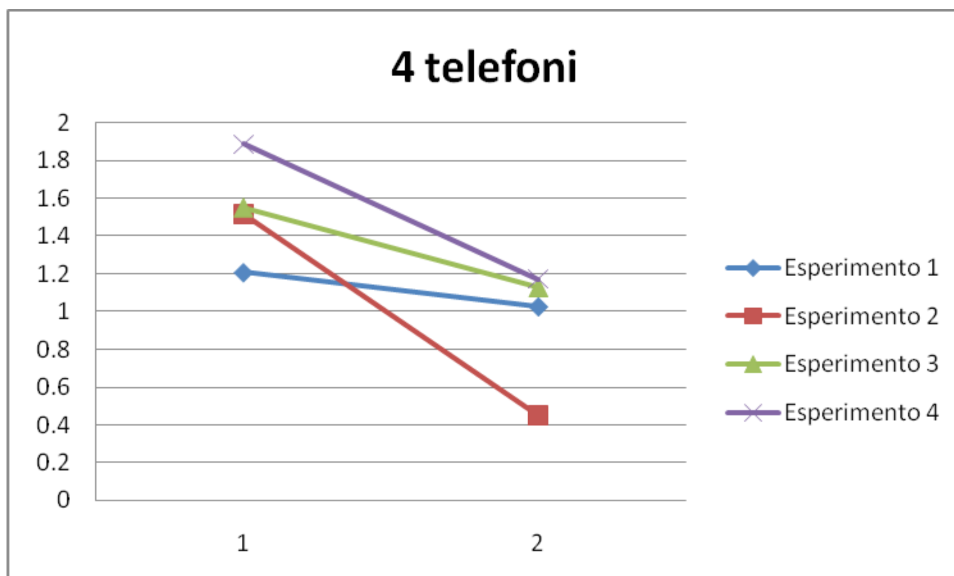

Grafico 23 Frequenza media di spedizione dei messaggi tra 4 telefoni (in ordinata, espressa in mex/s) in funzione del numero di gruppi (in ascissa) e della dimensione dei messaggi scambiati.

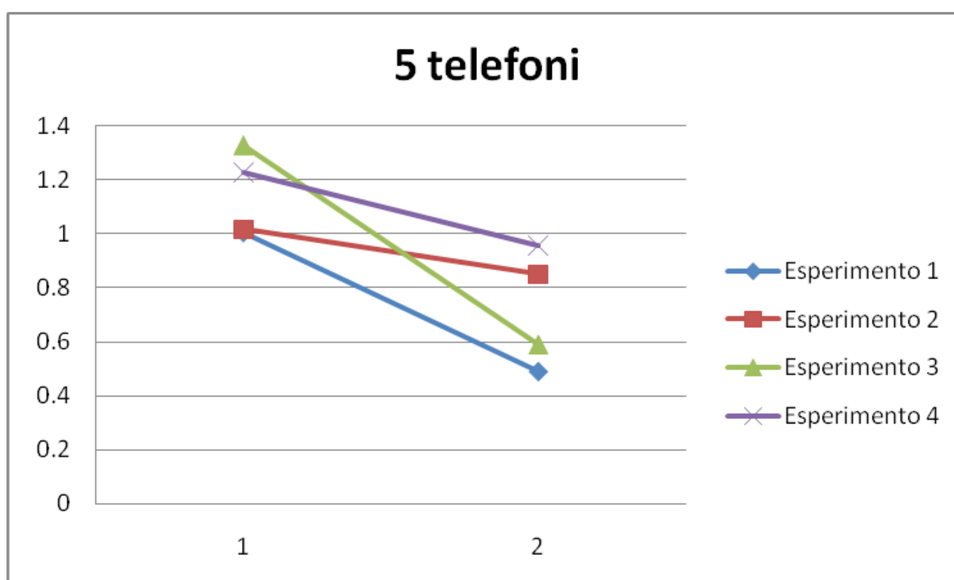

Grafico 24 Frequenza media di spedizione dei messaggi tra 5 telefoni (in ordinata, espressa in mex/s) in funzione del numero di gruppi (in ascissa) e della dimensione dei messaggi scambiati.

Anche la frequenza media di spedizione dei messaggi, salvo anomalie, diminuisce all'aumentare della dimensione degli stessi. Tale diminuzione è meno sempre meno evidente aumentando il numero di telefoni e di gruppi presenti nel sistema. La

diminuzione più evidente si ha nel passaggio da due a tre telefoni: la maggior complessità della gestione di una sessione multipoint rispetto ad una sessione singlepoint in AllJoyn influisce pesantemente sulla velocità di ricezione e spedizione dei messaggi.

6.3 A3Test_5

Scopo dell'applicazione A3Test_5 è confrontare i tempi di risposta dei follower a messaggi inviati dal supervisore mediante spedizioni unicast, multicast e broadcast. Questo perché tali spedizioni avvengono in modalità differenti. L'applicazione A3Test_5 consente di misurare le variazioni dei tempi di risposta in funzione di:

- numero di telefoni presenti nel sistema;
- dimensione dei messaggi scambiati dai telefoni nel sistema;
- numero di gruppi presenti nel sistema.

Il suo funzionamento è analogo a quello dell'applicazione A3Test_3, ma l'esperimento base è un altro: il supervisore di ogni gruppo spedisce dapprima un messaggio unicast ad un follower e ne rileva il tempo di risposta. Poi, in caso di presenza di n >1 follower, il supervisore effettua la stessa misurazione spedendo lo stesso messaggio ad un numero sempre maggiore di follower e rilevando il tempo che intercorre tra la spedizione del messaggio e la ricezione dell'ultima risposta. La spedizione multicast è effettuata in sequenza verso 2, 3, … , n-1 follower. Completate le spedizioni multicast, il supervisore invia lo stesso messaggio in broadcast verso tutti i follower e misura sempre il tempo di ricezione dell'ultima risposta. Effettuata quest'ultima misura, l'esperimento termina. Ho progettato l'applicazione A3Test_5 in modo da ripetere automaticamente il suddetto esperimento per 32 volte consecutive e da disconnettere tutti i telefoni da tutti i gruppi, escluso "control", una volta terminato l'ultimo esperimento. I risultati sono raccolti dai supervisori di ogni gruppo, eccetto "control", dunque il lanciare un esperimento equivale a lanciare tanti esperimenti quanti sono i gruppi creati.

I risultati ottenuti, e di seguito illustrati, mi hanno spinto a modificare l'interfaccia grafica dell'applicazione A3Test_5 come mostrato in figura 26. I pulsanti "Start" e "Stop" equivalgono rispettivamente ai pulsanti "Start trial" e "Stop trial" dell'applicazione A3Test_3, mentre i pulsanti "i_n" servono a connettere il telefono al gruppo "A3Test5_i_n", dove, come per l'applicazione A3Test_3, "i" identifica la dimensione del messaggio ed "n" identifica il gruppo. La corrispondenza tra i valori di i e le dimensioni dei messaggi sono le stesse dell'applicazione A3Test_3. E' quindi possibile scegliere arbitrariamente quale telefono dovrà essere il supervisore di un nuovo gruppo. I risultati ottenuti mediante l'applicazione A3Test_3 mostrano come non abbia mai senso creare più di 5 gruppi. Per questo motivo, i pulsanti "i_n" dell'applicazione A3Test_5 consentono di creare massimo 5 gruppi.
L'interfaccia grafica dell'applicazione A3Test_5 è costituita dalla classe "a3.test5.Test5Activity". Il descrittore del gruppo "control" è rappresentato dalla classe "a3.test5.ControlDescriptor", che definisce come ruolo di supervisore la classe "a3.test5.ControlSupervisorRole" e come ruolo di follower la classe "a3.test5.ControlFollowerRole". I gruppi "A3Test5_i_n" hanno come descrittore la classe "a3.test5.ExperimentDescriptor", che definisce come ruolo di supervisore la classe "a3.test5.ExperimentSupervisorRole" e come ruolo di follower la classe "a3.test5.ExperimentFollowerRole".

Figura 26 L'interfaccia grafica dell'applicazione A3Test_5.

I grafici dal 25 al 28 mostrano i dati ottenuti mantenendo ogni volta fissa la dimensione dei messaggi e variando invece il numero di telefoni e di gruppi presenti nel sistema. In ascissa si trova il numero di follower a cui è destinato un messaggio, mentre in ordinata è presente il tempo di ricezione dell'ultima risposta. I risultati mostrano che tale tempo aumenta all'aumentare del numero di telefoni e di gruppi presenti nel sistema e che l'aumentare del numero di telefoni ha un impatto maggiore sul tempo di risposta rispetto alla creazione di nuovi gruppi. L'andamento del tempo di risposta con più di due telefoni è corretto ed è dovuto al diverso tipo di comunicazione tra essi:

- la comunicazione unicast è eseguita mediante chiamata sincrona al metodo receiveUnicast(A3Message) dello UnicastReceiver del destinatario;
- la comunicazione multicast è eseguita come sequenza di trasmissioni unicast verso nodi diversi, ed ecco perché il tempo di risposta tende ad aumentare;
- la comunicazione broadcast è eseguita per ultima tramite invio di un BusSignal AllJoyn asincrono, il quale raggiunge tutti i telefoni in un arco di tempo più breve e la risposta dei telefoni è dunque più veloce.

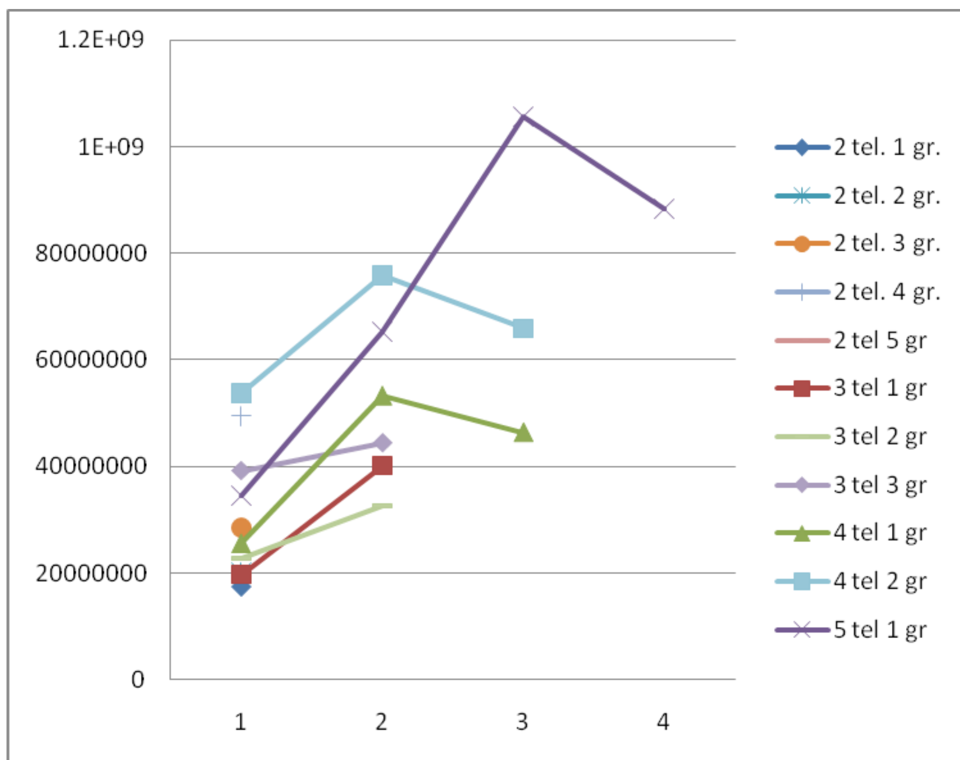

Grafico 25 Tempi medi (in ordinata, espressi in ns) di risposta dell'ultimo follower ad un messaggio di 62484 Byte inviato dal supervisore, in funzione del numero di telefoni e di gruppi presenti nel sistema. In ascissa è espresso il numero di follower destinatari del messaggio.

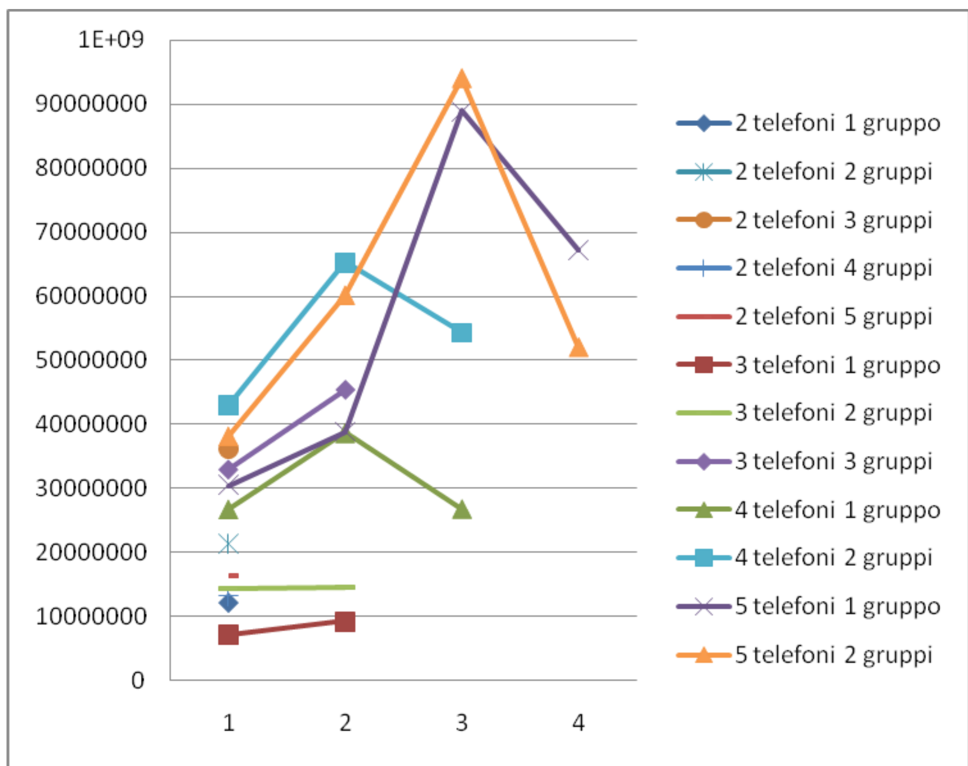

Grafico 26 Tempi medi (in ordinata, espressi in ns) di risposta dell'ultimo follower ad un messaggio di 32000 Byte inviato dal supervisore, in funzione del numero di telefoni e di gruppi presenti nel sistema. In ascissa è espresso il numero di follower destinatari del messaggio.

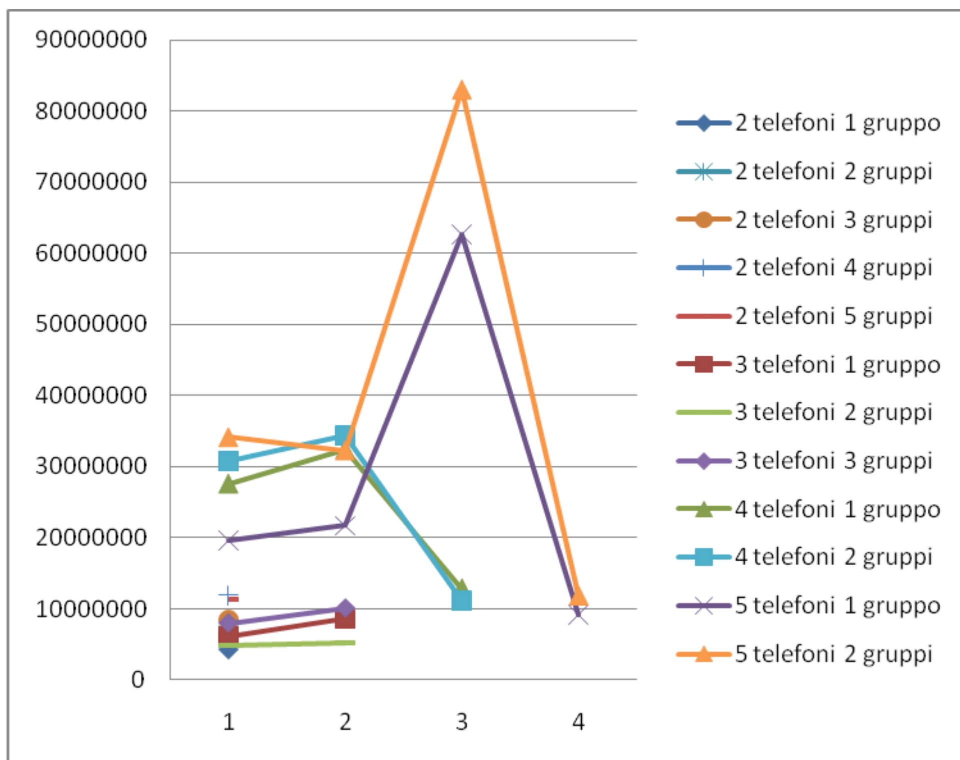

Grafico 27 Tempi medi (in ordinata, espressi in ns) di risposta dell'ultimo follower ad un messaggio di 5017 Byte inviato dal supervisore, in funzione del numero di telefoni e di gruppi presenti nel sistema. In ascissa è espresso il numero di follower destinatari del messaggio.

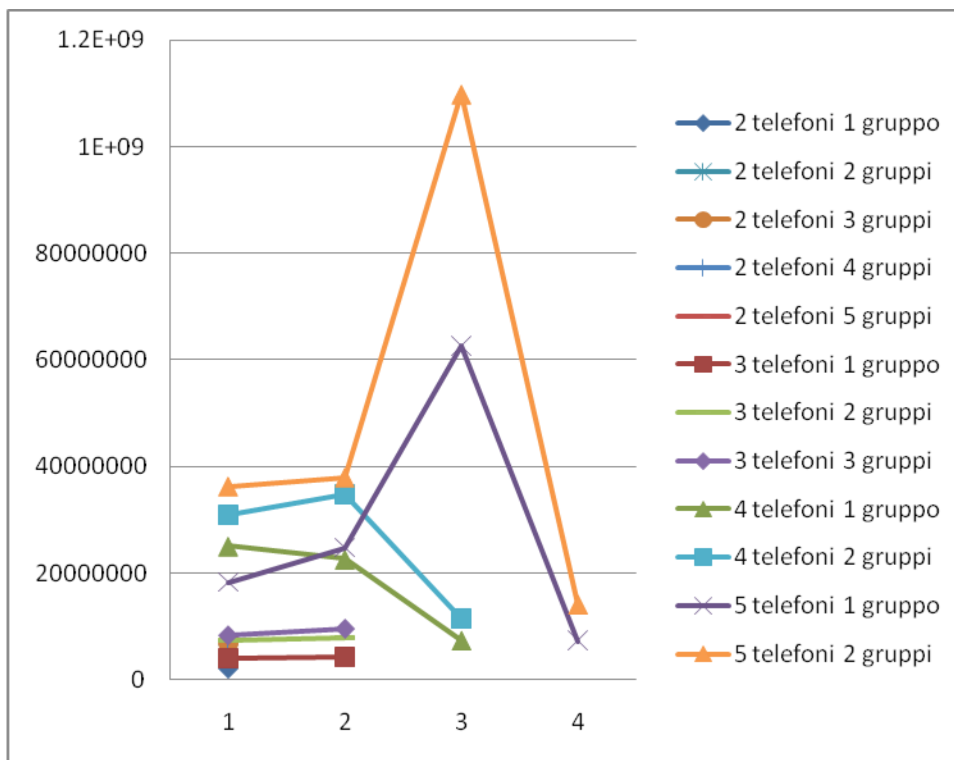

Grafico 28 Tempi medi (in ordinata, espressi in ns) di risposta dell'ultimo follower ad un messaggio di 1812 Byte inviato dal supervisore, in funzione del numero di telefoni e di gruppi presenti nel sistema. In ascissa è espresso il numero di follower destinatari del messaggio.

Un'anomalia è presente nel grafico 28, dove il tempo di risposta del terzo follower con cinque telefoni e due gruppi è maggiore rispetto allo stesso tempo di risposta nel grafico 27, nonostante i messaggi scambiati siano di dimensione minore. I tempi di risposta tendono ad aumentare all'aumentare del numero di telefoni presenti nel sistema. A pari numero di telefoni, i tempi di risposta più bassi si hanno per le configurazioni con meno gruppi.

I tempi di risposta a messaggi di 65484 Byte di 32000 Byte sono molto simili tra loro, così come sono simili tra loro quelli a messaggi di 5017 Byte e di 1812 Byte. I tempi di risposta a messaggi di grandi dimensioni sono più elevati di quelli a messaggi di piccole dimensioni e la differenza è particolarmente evidente nelle configurazioni a quattro e cinque telefoni. Usando messaggi di piccole dimensioni, il tempo di risposta ad un messaggio broadcast è più veloce di quello ad un messaggio unicast, mentre ciò non avviene per i messaggi di grandi dimensioni.

I grafici dal 29 al 32 mostrano i dati raccolti mantenendo costante il numero di telefoni presenti nel sistema e variando la dimensione dei messaggi ed il numero di gruppi creati. Il tempo di risposta presenta lo stesso andamento descritto in precedenza per i medesimi motivi e tende ad aumentare all'aumentare del numero di gruppi e della dimensione dei messaggi. Il grafico 29, ottenuto usando solo due telefoni, è diverso dagli altri perché, essendo ogni gruppo composto da un solo follower, il supervisore aspetta sempre una sola risposta e non ha senso mettere in ascissa il numero di destinatari del messaggio. Nella legenda non compare nemmeno il numero di gruppi creati, in quanto ricavare la media delle misure effettuate in ogni gruppo equivale a ricavare le misure medie su un telefono solo. Ho dunque potuto mettere in ascissa il numero di gruppi anziché il numero di telefoni destinatari del messaggio.

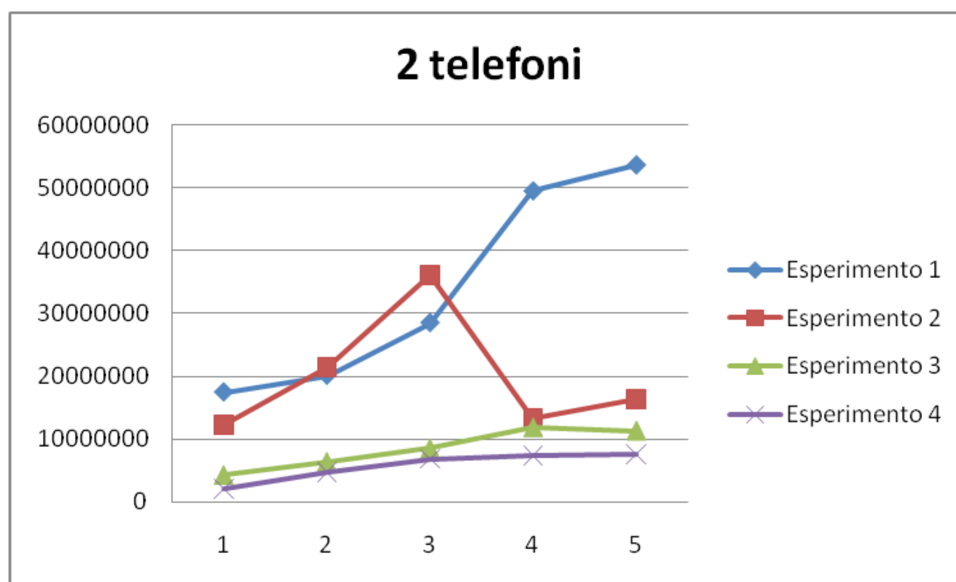

Grafico 29 Tempi medi (in ordinata, espressi in ns) di risposta dell'ultimo follower ad un messaggio inviato dal supervisore in un gruppo di 2 telefoni, in funzione del numero di gruppi presenti nel sistema (in ascissa) e della dimensione del messaggio.

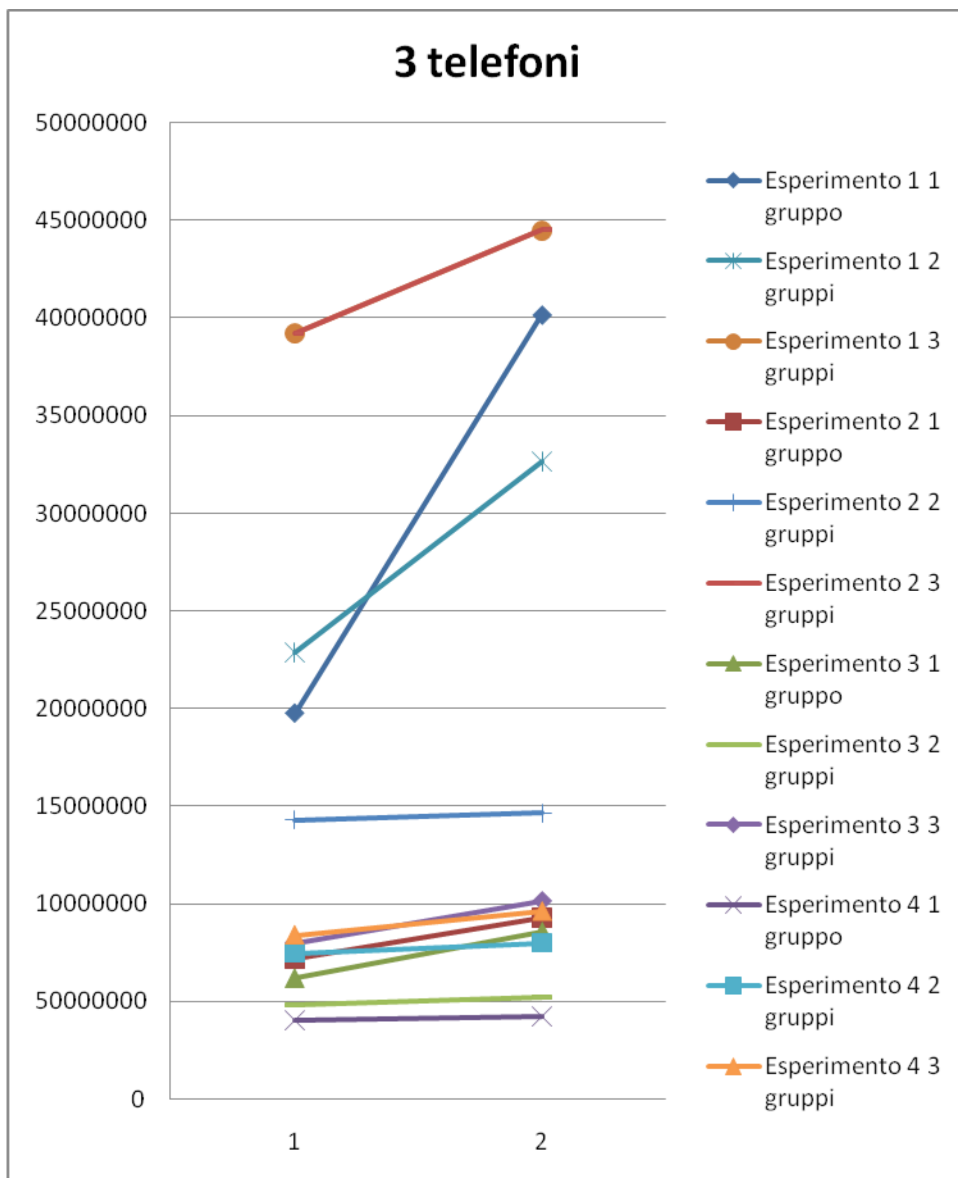

Grafico 30 Tempi medi (in ordinata, espressi in ns) di risposta dell'ultimo follower ad un messaggio inviato dal supervisore in un gruppo di 3 telefoni, in funzione del numero di gruppi presenti nel sistema e della dimensione del messaggio. In ascissa è espresso il numero di follower destinatari del messaggio.

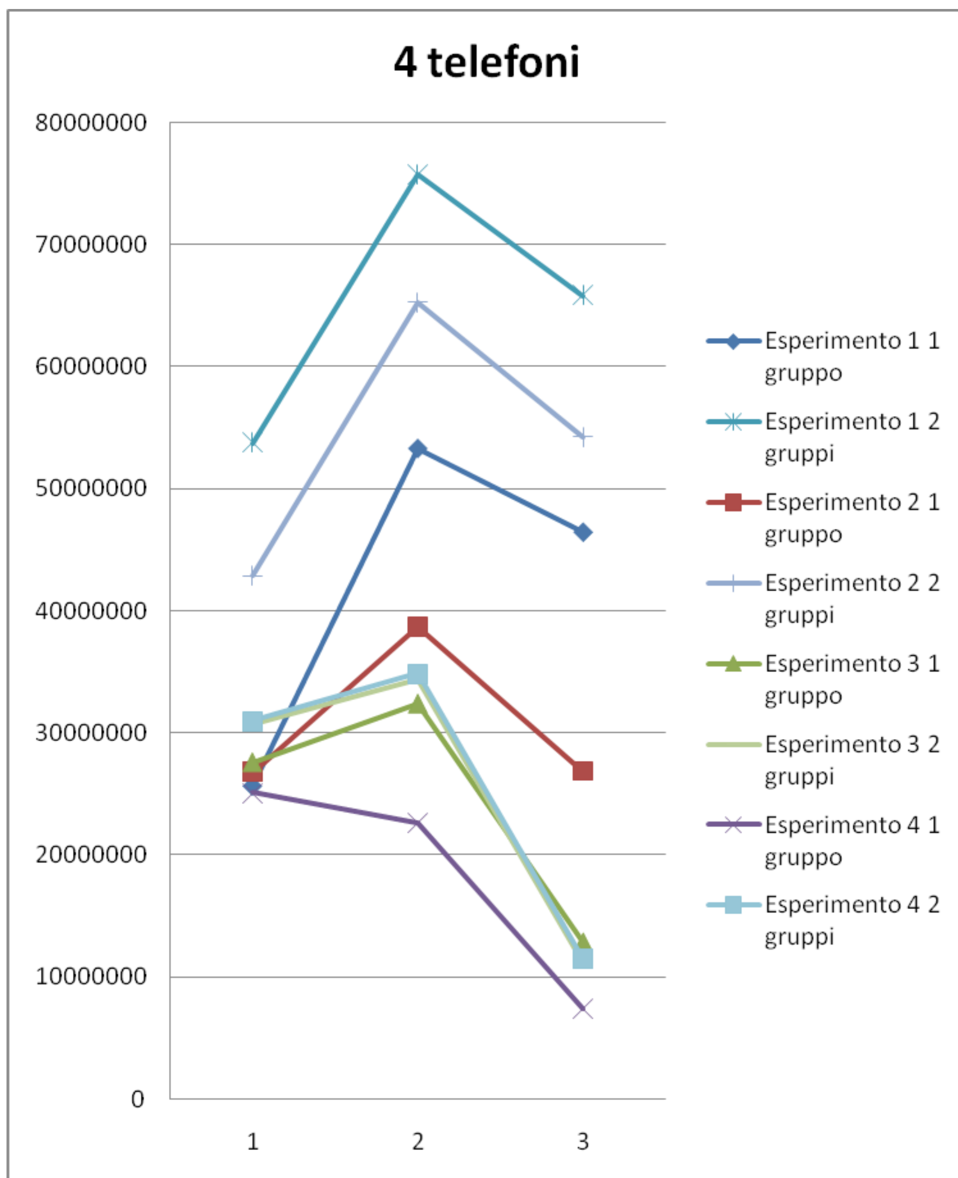

Grafico 31 Tempi medi (in ordinata, espressi in ns) di risposta dell'ultimo follower ad un messaggio inviato dal supervisore in un gruppo di 4 telefoni, in funzione del numero di gruppi presenti nel sistema e della dimensione del messaggio. In ascissa è espresso il numero di follower destinatari del messaggio.

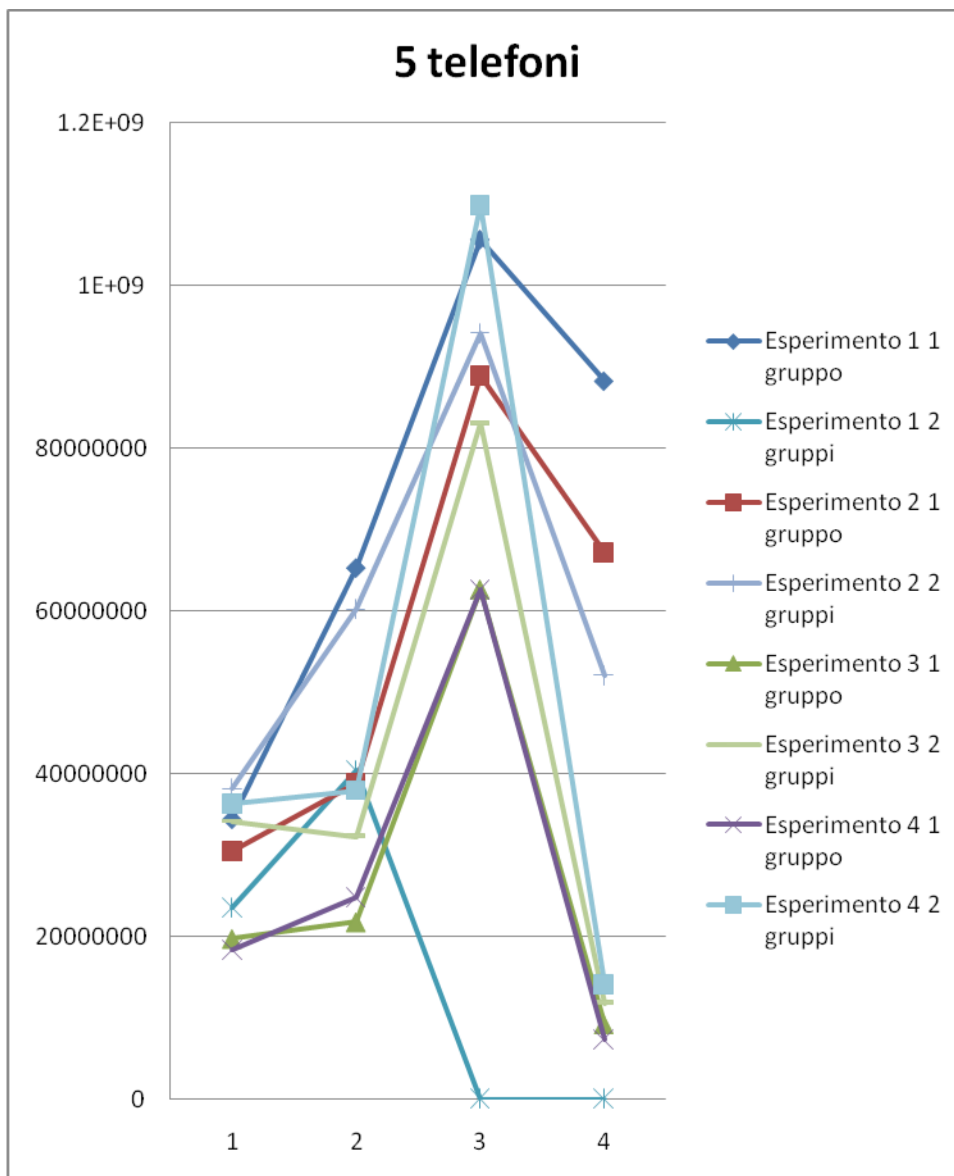

Grafico 32 Tempi medi (in ordinata, espressi in ns) di risposta dell'ultimo follower ad un messaggio inviato dal supervisore in un gruppo di 5 telefoni, in funzione del numero di gruppi presenti nel sistema e della dimensione del messaggio. In ascissa è espresso il numero di follower destinatari del messaggio.

I tempi di risposta aumentano all'aumentare del numero di telefoni, a causa dell'aumento del traffico sulla rete. A parità di numero di telefoni, i tempi di risposta risultano crescere all'aumentare della dimensione dei messaggi, eccezione fatta per l'anomalia dell'esperimento 1 con due telefoni. La distanza tra i tempi di risposta a messaggi grandi e quelli a messaggi piccoli risulta diminuire all'aumentare del numero di telefoni presenti nel sistema.

6.4 A3Test_7

Scopo dell'applicazione A3Test_7 è misurare i tempi di risposta di ogni nodo al messaggio di inizio dell'elezione di un nuovo supervisore, al variare del numero di telefoni e del numero di gruppi all'interno del sistema. Per fare ciò, ho modificato la classe FitnessFunctionManager rimuovendo il timeout entro il quale i nodi devono rispondere ed attrezzandola per la memorizzazione dei tempi di risposta su un file. Tali tempi sono raccolti dai FitnessFunctionManager di ogni gruppo, quindi il lanciare un esperimento equivale a lanciarne tanti quanti sono i gruppi creati. La struttura dell'applicazione A3Test_7 è analoga a quella delle applicazioni A3Test_3 ed A3Test_5, ma l'esperimento base è diverso e consiste nell'avviare l'elezione di un nuovo supervisore. Ho progettato l'applicazione A3Test_7 per ripetere automaticamente questo esperimento per 32 volte e per disconnettere i telefoni da tutti i gruppi, eccetto "control", al termine dell'ultimo esperimento. La figura 27 mostra l'interfaccia grafica dell'applicazione A3Test_7: i pulsanti "Create group", "Start" e "Stop" sono analoghi a quelli delle altre applicazioni, mentre è possibile riconnettere un telefono al gruppo specificato nella casella di testo indicata con "*" premendo il tasto "Reconnect" in caso di necessità.

L'interfaccia grafica dell'applicazione A3Test_7 è costituita dalla classe "a3.test7.Test7Activity". Il descrittore del gruppo "control" è rappresentato dalla classe "a3.test7.ControlDescriptor", che definisce come ruolo di supervisore la classe "a3.test7.ControlSupervisorRole" e come ruolo di follower la classe "a3.test7.ControlFollowerRole". I gruppi "A3Test7_i", dove "i" è il numero progressivo identificativo di un gruppo, hanno come descrittore la classe "a3.test7.ExperimentDescriptor", che definisce come ruolo di supervisore la classe "a3.test7.ExperimentSupervisorRole" e come ruolo di follower la classe "a3.test7.ExperimentFollowerRole".

Figura 27 L'interfaccia grafica dell'applicazione A3Test_7.

I grafici dal 33 al 36 mostrano i tempi di risposta di ogni telefono al variare del numero di gruppi, mantenendo costante il numero di telefoni: tali tempi di risposta aumentano all'aumentare del numero di gruppi.

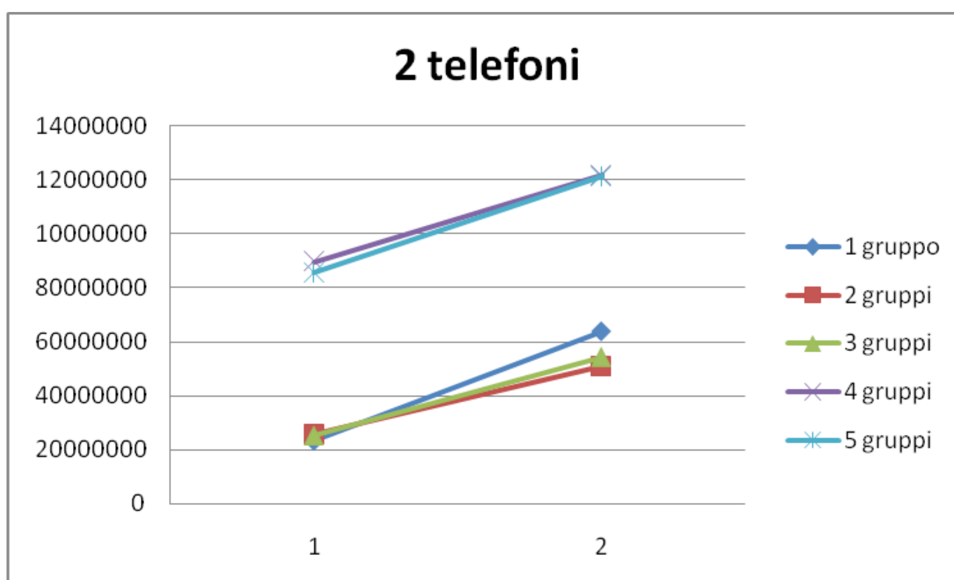

Grafico 33 Tempi medi di risposta (in ordinata, espressi in ns) di un telefono ad un messaggio di elezione del supervisore in un gruppo di 2 telefoni, in funzione del numero di gruppi presenti nel sistema. I numeri in ascissa identificano l'ordine delle risposte ricevute.

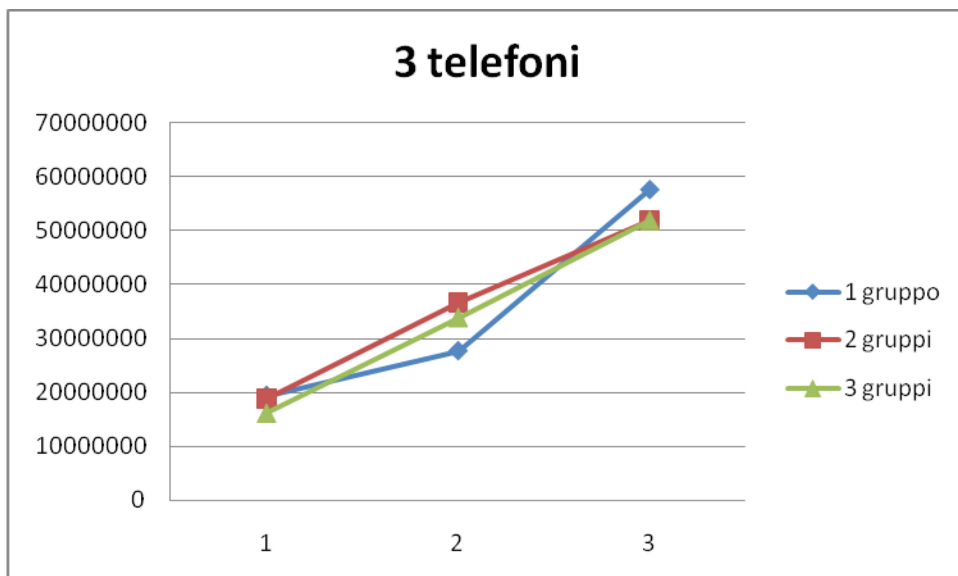

Grafico 34 Tempi medi di risposta (in ordinata, espressi in ns) di un telefono ad un messaggio di elezione del supervisore in un gruppo di 3 telefoni, in funzione del numero di gruppi presenti nel sistema. I numeri in ascissa identificano l'ordine delle risposte ricevute.

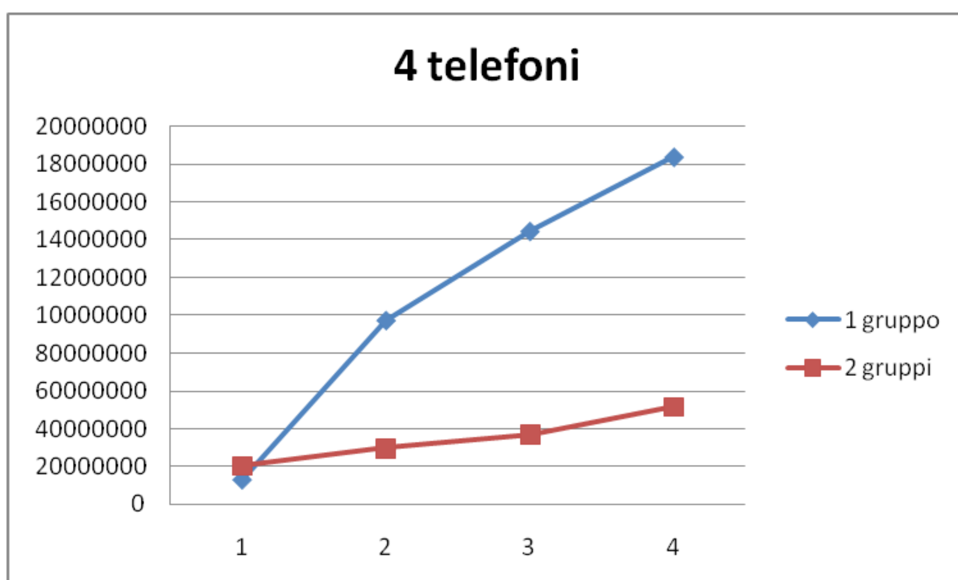

Grafico 35 Tempi medi di risposta (in ordinata, espressi in ns) di un telefono ad un messaggio di elezione del supervisore in un gruppo di 4 telefoni, in funzione del numero di gruppi presenti nel sistema. I numeri in ascissa identificano l'ordine delle risposte ricevute.

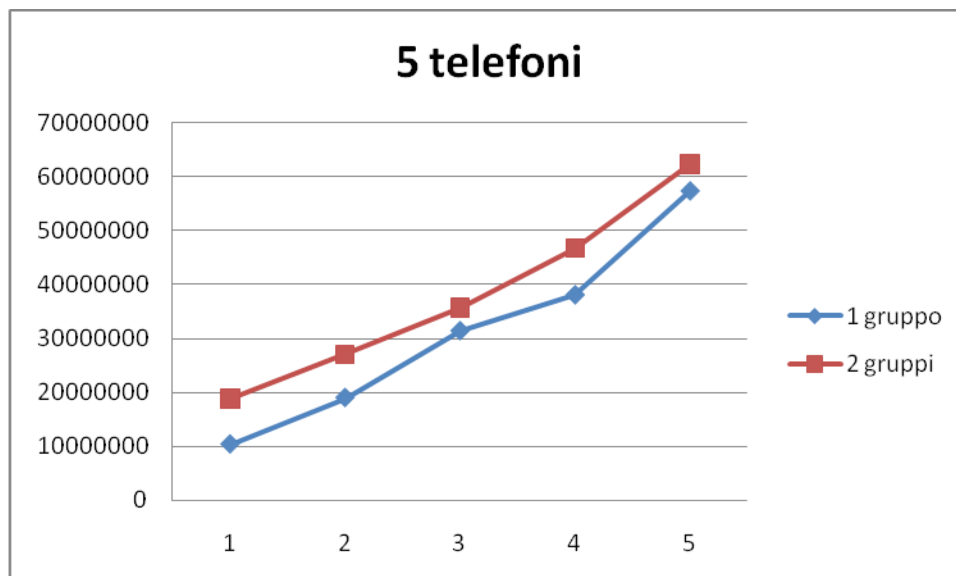

Grafico 36 Tempi medi di risposta (in ordinata, espressi in ns) di un telefono ad un messaggio di elezione del supervisore in un gruppo di 2 telefoni, in funzione del numero di gruppi presenti nel sistema. I numeri in ascissa identificano l'ordine delle risposte ricevute.

Fatta eccezione per l'anomalia presente nel grafico 35, i tempi medi di risposta ricavati in presenza di due, tre, quattro e cinque telefoni risultano essere molto simili e compresi nella fascia 10 – 60 ms. Al contrario di quanto visto con le altre applicazioni, le prestazioni non variano sensibilmente al variare del numero di telefoni e di gruppi presenti nel sistema. Siccome nell'applicazione A3Test_7, al contrario che nelle altre applicazioni, il supervisore non gioca nessun ruolo importante, deduco che la causa principale del peggioramento delle prestazioni di A3Droid sia il supervisore.

Il grafico 33 del tempo medio di risposta con due telefoni pone in risalto una netta differenza tra i tempi di risposta ottenuti in presenza di quattro e cinque gruppi e quelli ottenuti in presenza di meno gruppi. Non potendo creare quattro o cinque gruppi di tre o più telefoni, è impossibile verificare se tale differenza sia anomala oppure determinarne la causa.

I grafici dal 37 al 39 mostrano i tempi di risposta di ogni telefono al variare del numero di telefoni, mantenendo costante il numero di gruppi. Le configurazioni a 4 e 5 gruppi sono ottenibili solo con 2 telefoni, dunque non sono possibili confronti al variare del numero di telefoni.

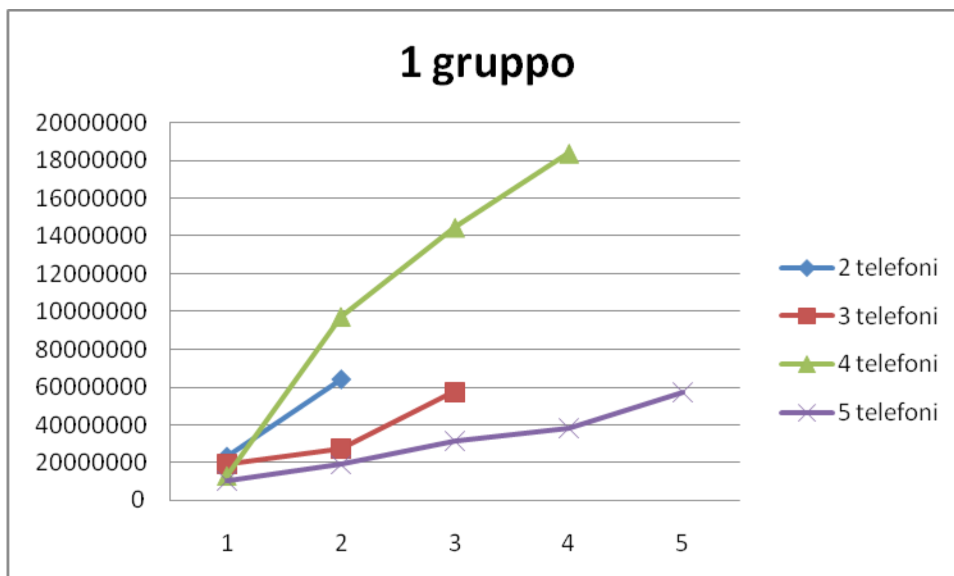

Grafico 37 Tempi medi di risposta (in ordinata, espressi in ns) di un telefono ad un messaggio di elezione del supervisore, in funzione del numero di telefoni presenti in un gruppo. I numeri in ascissa identificano l'ordine delle risposte ricevute. Nel sistema è presente un gruppo solo.

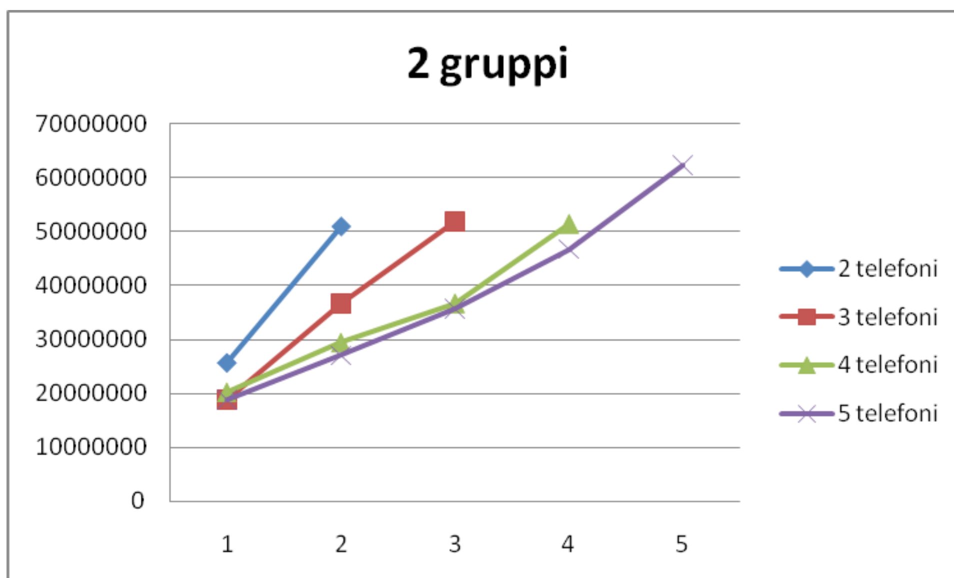

Grafico 38 Tempi medi di risposta (in ordinata, espressi in ns) di un telefono ad un messaggio di elezione del supervisore, in funzione del numero di telefoni presenti in un gruppo. I numeri in ascissa identificano l'ordine delle risposte ricevute. Nel sistema sono presenti 2 gruppi.

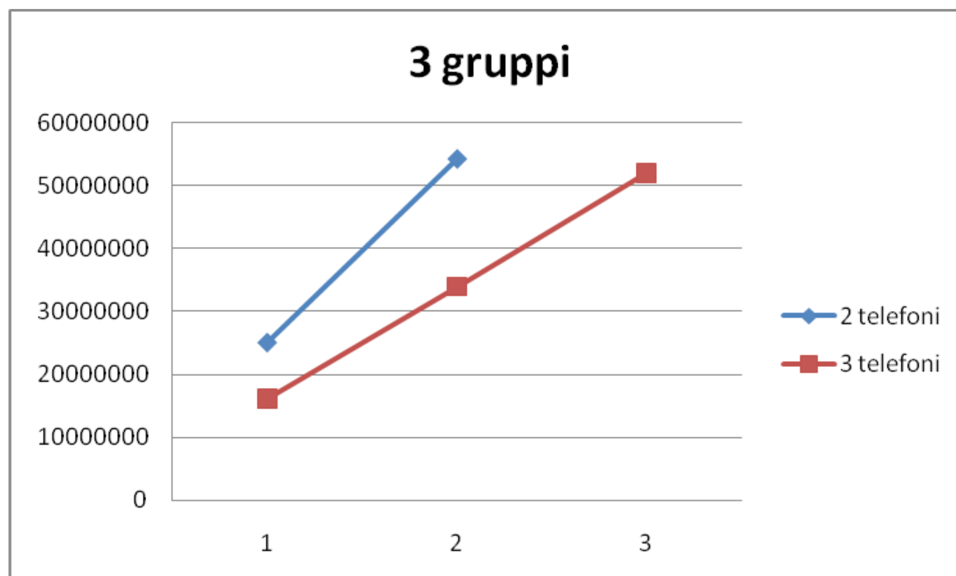

Grafico 39 Tempi medi di risposta (in ordinata, espressi in ns) di un telefono ad un messaggio di elezione del supervisore, in funzione del numero di telefoni presenti in un gruppo. I numeri in ascissa identificano l'ordine delle risposte ricevute. Nel sistema sono presenti 3 gruppi.

Fatta eccezione per il grafico 37, i tempi di risposta sembrano diminuire all'aumentare del numero di telefoni, ma restano sempre compresi nella fascia 10 – 60 ms.

6.5 Discussione dei risultati ottenuti

I grafici precedentemente discussi presentano ancora delle anomalie, dovute al fatto che i supervisori erano distribuiti sui telefoni secondo una distribuzione diversa ogni volta, generata automaticamente e non sempre uniforme. Modificando l'applicazione A3Test_5 ho poi constatato che le prestazioni migliori si ottengono quando i supervisori sono distribuiti uniformemente sui telefoni. Questo perché il supervisore risulta essere il collo di bottiglia dell'applicazione e la condivisione della CPU di uno stesso telefono tra più supervisori li rallenta inevitabilmente tutti. I tempi di risposta ottenuti uniformando la distribuzione dei supervisori sui telefoni sono inferiori di circa 300 ms rispetto ad una configurazione in cui tutti i supervisori risiedono su un unico telefono. Questo risultato influenza ovviamente i risultati ottenuti mediante tutte le applicazioni e spiega le anomalie di tutti i grafici finora presentati: una

configurazione con più telefoni, ma con supervisori distribuiti più uniformemente su di essi, si comporta meglio di una configurazione con meno telefoni, ma con supervisori distribuiti meno uniformemente su di essi.

Risolte queste anomalie, dagli esperimenti da me effettuati emerge che:

- il numero medio di messaggi spediti, il tempo medio di raggiungimento della soglia di 1s e la frequenza media di spedizione dei messaggi tendono a diminuire all'aumentare del numero di telefoni, di gruppi e della dimensione dei messaggi scambiati;
- il passaggio da una configurazione a due telefoni ad una configurazione a tre telefoni determina un peggioramento delle prestazioni dovuto al passaggio di una sessione AllJoyn da singlepoint a multipoint;
- il tempo di risposta di ogni follower ai messaggi del supervisore aumenta all'aumentare del numero di telefoni e di gruppi presenti nel sistema e l'aumentare del numero di telefoni ha un impatto maggiore sul tempo di risposta rispetto alla creazione di nuovi gruppi;
- in presenza di uno, due, o tre gruppi, la risposta ad un messaggio di inizio dell'elezione di un nuovo supervisore avviene entro 60 ms, indipendentemente dal numero di telefoni;
- in presenza di due telefoni e quattro o cinque gruppi, la risposta al messaggio d'inizio dell'elezione di un nuovo supervisore avviene entro 120 ms;
- le prestazioni di A3Droid peggiorano all'aumentare della dimensione dei messaggi scambiati ed il peggioramento è più evidente per le dimensioni dei messaggi più piccole;
- il collo di bottiglia di un'applicazione A3Droid è il supervisore.

Il codice di A3Droid risulta non essere adeguatamente scalabile per operare in sistemi di migliaia di nodi. Per scoprirne il motivo, ho realizzato l'applicazione DueSoliBa, la quale dimostra che ad influire pesantemente sul numero di gruppi introducibili nel sistema è il numero di BusAttachment AllJoyn creati: riducendo il loro numero a 2, si riesce a connettere un telefono ad un numero di gruppi molto maggiore, dell'ordine di molte centinaia. Oltre a migliorare le prestazioni di

A3Droid, la riduzione del numero di BusAttachment porterebbe inevitabilmente ad una riorganizzazione del codice che potrebbe introdurre miglioramenti ancora maggiori. Per esempio, i grafici dell'applicazione A3Test_5 mostrano che, a fronte di un elevato numero di nodi, potrebbe convenire implementare le comunicazioni unicast e multicast come trasmissioni broadcast di messaggi contenenti gli indirizzi delle destinazioni. Tali messaggi sarebbero ricevuti da tutti i telefoni appartenenti al gruppo e sarebbero da essi filtrati in base all'indirizzo. Questo porterebbe alla mancata creazione di BusAttachment AllJoyn aggiuntivi per implementare questi due tipi di comunicazioni, ed anche alla sparizione della necessità di utilizzare per esse metodi sincroni che rallentano il sistema.

Pur introducendo questi miglioramenti, le prestazioni del sistema dipenderebbero comunque sempre da fattori esterni ed indipendenti da un'applicazione A3Droid e da A3Droid stesso, come:

- capacità dell'access point Wi-Fi a cui risultano connessi i telefoni presenti nel sistema;
- banda utilizzata per la trasmissione dei messaggi;
- capacità di ogni singolo telefono in termini di CPU e memoria.

7 Conclusioni

Nei capitoli precedenti ho presentato lo stile architetturale A-3, evidenziando come esso presenti due concetti innovativi rispetto agli altri approcci al paradigma dell'autonomic computing: la suddivisione del sistema in gruppi di nodi e la gestione dell'intero sistema mediante composizione di tali gruppi. Nessuno degli altri approcci presentati possiede queste caratteristiche, motivo per cui era necessaria un'implementazione di A-3. A tale scopo, ho dunque realizzato A3Droid. Le API di A3Droid consentono di creare, distruggere e comporre tra loro i gruppi che si prevede faranno parte del sistema. In particolare, esse possono essere invocate automaticamente dall'applicazione per dividere un gruppo sovrappopolato o unire tra loro gruppi sottopopolati, al fine di garantire la scalabilità del sistema al variare del numero di nodi al suo interno.

Ho poi valutato le prestazioni di A3Droid realizzando applicazioni ottenute estendendo opportunamente le classi di A3Droid stesso rappresentanti ruoli e descrittori dei gruppi. Tale realizzazione è risultata essere facile e veloce, il che conferma che A3Droid fornisce un valido supporto alla progettazione di applicazioni mobile distribuite.

Le prestazioni di A3Droid si sono però rivelate scarse, a causa di un utilizzo non ottimale del framework AllJoyn, sul quale ho costruito la struttura di A3Droid. Le prove da me effettuate dimostrano che è possibile ottenere sensibili miglioramenti riducendo il più possibile il numero di BusAttachment AllJoyn e facendo sì che su ogni dispositivo mobile sia presente al più un supervisore. E' inoltre possibile accelerare le comunicazioni unicast e multicast implementando anch'esse come BusSignal AllJoyn: questa modifica porterebbe all'eliminazione delle classi A3UnicastReceiver ed A3UnicastTransmitter di A3Droid, riducendo così anche il numero di BusAttachment AllJoyn.

AllJoyn sembra essere il framework ideale sul quale poggiare non solo A3Droid, ma

anche eventuali altre future implementazioni di A-3. Infatti esso astrae e rende interoperabili diversi tipi di rete di comunicazione.

Ho progettato le API di A3Droid per il funzionamento su piattaforma Android, ma grazie ad AllJoyn l’uso di A3Droid può essere esteso anche ad altre piattaforme semplicemente traducendo le sue API in diversi opportuni linguaggi di programmazione.

Bibliografia

[1] L. Baresi, S. Guinea. “A-3: an Architectural Style for Coordinating Distributed Components”. In “Proceedings of the 9th Working IEEE/IFIP Conference on Software Architecture (WICSA)”, 2011.

[2] AllJoyn System Description – AllSeen Alliance. https://allseenalliance.org/about/why-allseen/

[3] T. O. Eze, R. J. Anthony, C. Walshaw A. Soper. “A New Architecture for Trustworthy Autonomic Systems”. In “EMERGING 2012: The Fourth International Conference on Emerging Network Intelligence.”, 2012.

[4] J. P. Sousa, D. Garlan. “Aura: An Architectural Framework for User Mobility in Ubiquitous Computing Environments”. In “Software Architecture: System Design, Development, and Maintenance (Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture) Bosch, Gentleman, Hofmeister, Kuusela (Eds), Kluwer Academic Publishers, pp. 29-43, August 2002.”.

[5] H. Viswanathan, E. Kyung Lee, I. Rodero, D. Pompili. “An Autonomic Resource Provisioning Framework for Mobile Computing Grids”.

[6] M. Parashar, S. Hariri. “Autonomic Computing: An Overview”, 2005.

[7] S. Guinea, P. Saeedi. “Coordination of Distributed Systems through Self-Organizing Group Topologies”.

[8] G. Kaiser, J. Parekh, P. Gross, and G. Valletto. “Kinesthetichs extreme: An external infrastructure for monitoring distributed legacy systems”. In IEEE 5th Annual International Active Middleware Workshop, 2003.

[9] I. Satoh. “MobileSpaces: A Framework for Building Adaptive Distributed Applications Using a Hierarchical Mobile Agent System”.

[10] Open Source IoT to advance the Internet of Everything – AllSeen Alliance. https://allseenalliance.org/

[11] K. Twidle, N. Dulay, E. Lupu, M. Sloman. “Ponder2: A Policy System for Autonomous Pervasive Environments”.

[12] M. Bettineschi. “Self-Coordination through Dynamic Group Management”, 2012.

[13] C. A. Mattmann, N. Medvidovic. “The Grid Lite DREAM: Bringing the Grid to Your Pocket”.

[14] J. O. Kephart, D. M. Chess. “The Vision of Autonomic Computing”, 2003.

[15] L.Baresi, S. Guinea, G. Tamburrelli. “Towards Decentralized Self-adaptive Component-based Systems”, 2008.