Alma Mater Studiorum · Università di Bologna

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di Laurea Specialistica in Informatica

ACQUISIZIONE ED ANALISI DI DATI UTILIZZANDO CLOUD COMPUTING: UNO STUDIO DI FATTIBILITÀ

Tesi di Laurea in Sistemi Middleware

Relatore:
Chiar.mo Prof.
FABIO PANZIERI

Correlatore:
Ing.
ANDREA CERFOGLI

Presentata da:
GIOVANNI BORELLI

Sessione terza
Anno Accademico 2009/2010

A Monica*:*
dolce mano sulla mia spalla

Indice

Elenco delle figure

Elenco delle tabelle

Ringraziamenti

Innanzitutto vorrei ringraziare il Professor Fabio Panzieri per gli insegnamenti da lui ricevuti durante questi due anni di laurea specialistica, per il supporto e i consigli da lui ricevuti nella redazione di questa tesi e per l'assoluta disponibilità umana che non ha mai fatto mancare.
Ringrazio calorosamente l'Ing. Andrea Cerfogli che si è sempre stato disponibile a seguire, stimolare e correggere il mio lavoro di tesi all'interno di Smarten s.r.l.
Ringrazio i colleghi tutti di Smarten s.r.l. ed in particolare l'Ing. Andrea Tinti per la disponibilità, il supporto ed incoraggiamento datomi in questi mesi di stesura della tesi.
Colgo l'occasione per ringraziare la mia famiglia (mamma, papà e Chiara), il cuginame più vicino (Ele, Giulia, Checco ..), gli amici di sempre (Silvan, Vale, Biccio, Alain, Livia, Cisco, Lu, Ferra, Galle, Giobbe, Erica, Valli, Eleo, Axel, Lali..), i frati (Luca e Andrea) e gli ospiti (Ago, Franco, Capi, Gian, Giuseppe, Luigi, Riki, Hien, Antonio, Ettore, Wanni, Piero, Jean..) della CDC di Cognento per avermi sempre incoraggiato ed aver sopportato la mia latitanza di questi mesi causa tesi. In particolare ringrazio Pacchio per aver sempre riempito il frigo di casa evitandomi una lenta morte per inedia.
Per i ringraziamenti a Monica per il suo supporto e la sua vicinanza non basterebbero 100 pagine di "grazie" quindi, onde evitare di incidere eccessivamente sul disboscamento, sull'effetto serra e sulla svalutazione reattiva mi limito in questo spazio ad un solo, grande, sentitissimo GRAZIE. Gli altri 99 può venirli a ritirare di persona.

Insomma, ringrazio tutti quelli che mi hanno sUpportato e sOpportato: so che è stata dura ;-)

Introduzione

Scopo di questa tesi è quello di valutare lo stato dell'arte delle tecnologie più diffuse e recenti che possano consentire la realizzazione di un sistema software distribuito per la raccolta e l'analisi di dati su Cloud Computing. Tale software dovrà presentare una struttura multi-tier in cui il tier di presentazione sarà centralizzato ed allocato su una piattaforma di Cloud Computing e i tier di application logic e di data model saranno distribuiti su svariati nodi dinamici ed eterogenei.

Il tier di presentazione allocato sulla piattaforma di Cloud Computing è la parte critica del sistema software da realizzare: non solo dovrà attingere ad un numero variabile di nodi distribuiti sulla rete internet per ottenere i dati ma dovrà anche gestire le logiche applicative eterogenee associate ai dati forniti distribuite sui diversi nodi. Pertanto lo scopo centrale di questo documento è quello di illustrare lo studio di fattibilità, la realizzazione di un dimostratore minimale di funzionalità e le sue relative misurazioni di correttezza e performance per la realizzazione del tier di presentazione centralizzato allocato su una piattaforma di Cloud Computing.

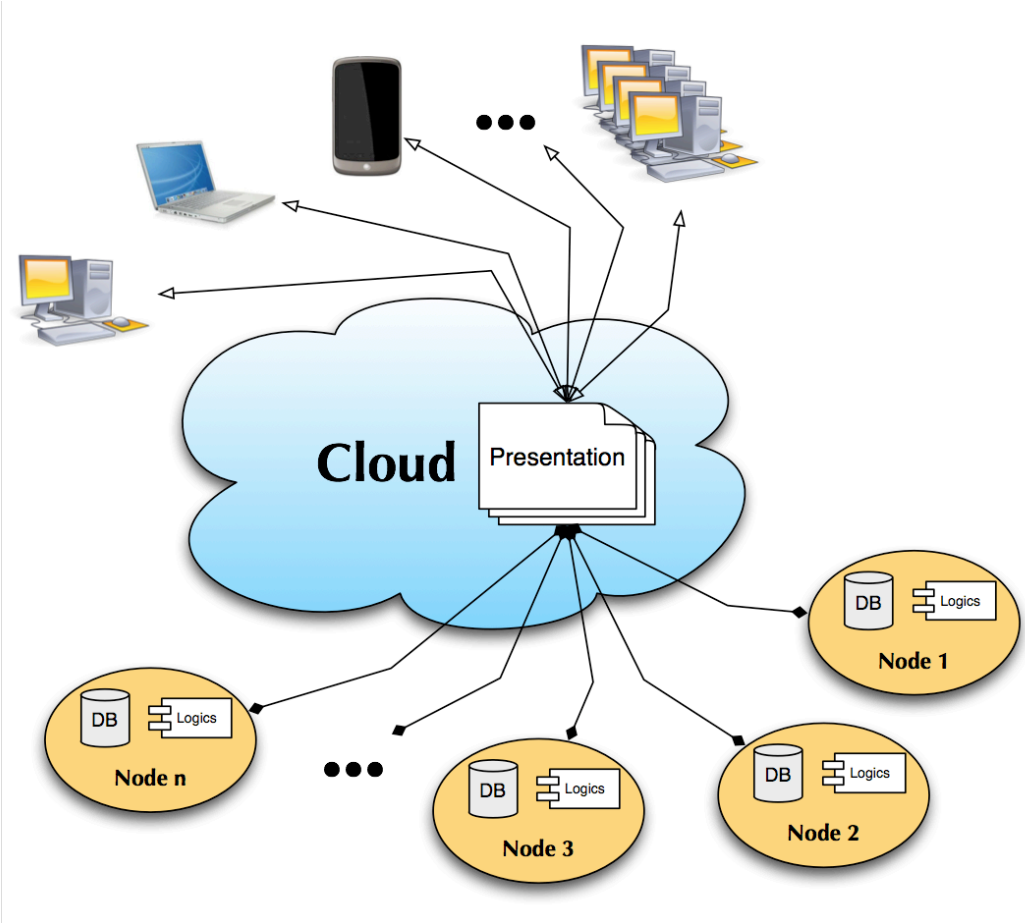

Figura 1: Disposizione dei tier di Dati, Logica e Presentazione

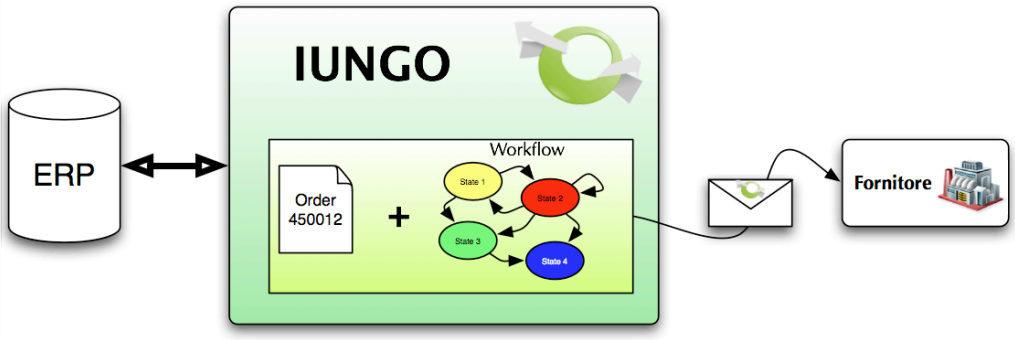

Figura 2: Flusso di dati dal ERP al Fornitore

Scopo di questa tesi è anche quello di presentare le analisi e le considerazioni emerse nello studio delle tematiche e delle motivazioni tecniche, commerciali e culturali che gravitano attorno ad un progetto software che è strategico per una realtà aziendale IT e che andrà disposto su un Cloud Computing gestito da società terze.

Motivazioni Le motivazioni che hanno portato all'idea e all'esigenza di realizzare lo studio di fattibilità oggetto di questa tesi nascono direttamente dall'esperienza e dalle prospettive di sviluppo di Iungo (dal latino "Collego"), software per la gestione integrata della supply chain aziendale che permette una comunicazione dinamica e flessibile tra il sistema ERP dell'azienda che lo utilizza ed i suoi fornitori mediante l'utilizzo di email contenenti form HTML.
Iungo è un software proprietario sviluppato da Smarten s.r.l., una realtà imprenditoriale piccola e molto dinamica: conta una decina di dipendenti di cui cinque sviluppatori software con un profilo formativo di stampo ingegneristico dediti al project management dei progetti Iungo e alla industrializzazione e sviluppo del core di Iungo stesso.

Iungo si occupa principalmente di assegnare un protocollo di comunicazione e un workflow evolutivo alle comunicazioni che avvengono tra una realtà aziendale e i suoi fornitori: principalmente ordini d'acquisto ma anche richieste d'offerta, solleciti, call-off e così via. Allo stato attuale possiamo

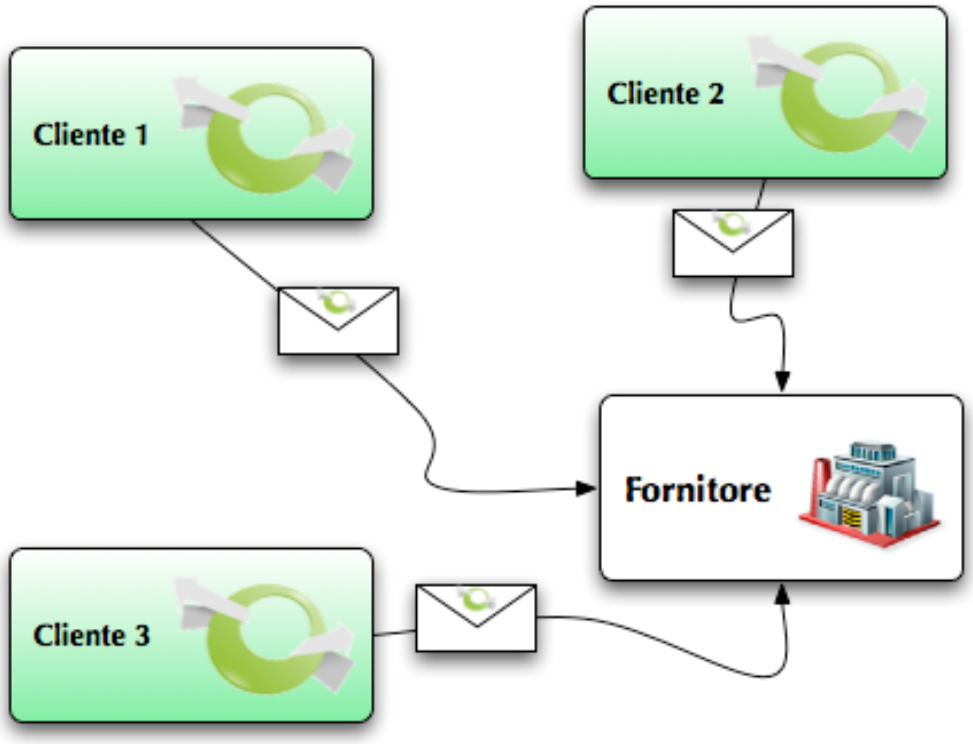

Figura 3: Comunicazioni disaggregate dai diversi Iungo verso un Fornitore

trovare un numero abbastanza significativo di installazioni diverse di Iungo che vivono e svolgono le loro attività in modo assolutamente indipendente e isolato tra loro raggiungendo con le loro comunicazioni circa 10.000 fornitori.

Le aziende che utilizzano Iungo appartengono principalmente a tre settori industriali: il settore meccanico, quello dei beni di consumo e il settore moda e lusso. Tale caratterizzazione settoriale comporta inevitabilmente che aziende appartenenti allo stesso settore si avvalgano degli stessi fornitori per l'approvvigionamento di materiali indispensabili per il ciclo produttivo piuttosto che per materiali prodotti unicamente da una specifica realtà industriale.

In tale condizione le varie installazioni di Iungo entrano indirettamente in contatto fra loro: il fornitore infatti riceverà email generate dallo Iungo

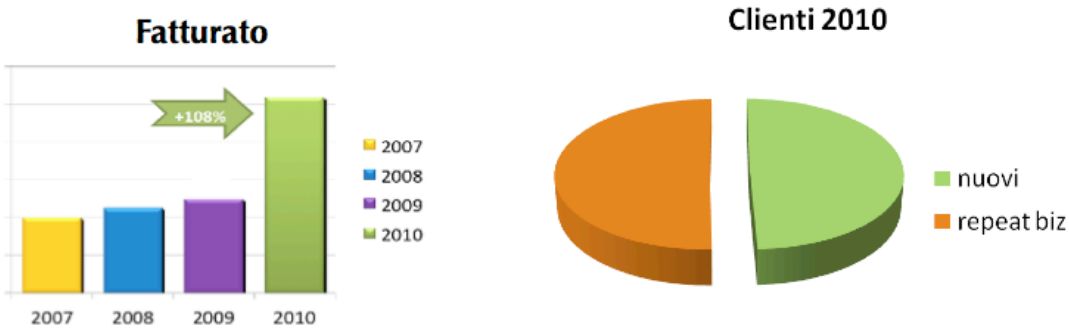

Figura 4: Andamenti di fatturato di Iungo e percentuale relativa a nuove installazioni nel 2010

di un cliente per i suoi ordini d'acquisto ed email generate dallo Iungo di un altro cliente per gli ordini d'acquisto di tale altro cliente. Questa "interazione indiretta" tra i diversi Iungo avviene quindi in modo disconnesso, eterogeneo e scorrelato: il fornitore si trova davanti a due dati che probabilmente hanno due logiche e due presentazioni considerevolmente differenti, ma che per l'utilizzo che lui deve farne sono strettamente correlati e indistinguibili, al punto che è molto probabile che entrambi verranno elaborati nello stesso modo dal ERP del fornitore come ordini di vendita con destinatari diversi.

Visto il successo commerciale che Iungo continua a riscontrare (fig 4) e la sempre maggiore diffusione di installazioni di Iungo questa situazione di "interazione indiretta e scorrelata" non può che essere destinata ad aumentare considerevolmente. Già in questo momento si riscontrano situazioni di fornitori che ricevono ordini d'acquisto su email generate da quattro Iungo di quattro clienti diversi.

È proprio per risolvere questa condizione disagevole per il fornitore di disaggregazione di dati e di logiche operative sui essi che nasce l'idea di Iungo Collector: un software che deve riuscire ad essere per il fornitore un unico punto d'accesso per la raccolta e l'analisi dei dati che tutti i suoi clienti

che utilizzano Iungo vogliono trasmettergli.

Stato dell'arte Il paradigma di Cloud Computing nasce per dare una risposta ad una precisa richiesta del mercato IT: coniugare la logica on-demand all'utilizzo di risorse per fornitura di un servizio.

Prima del Cloud Computing le prospettive tecnologiche disponibili ad una azienda IT che volesse fornire un servizio su cui basare il proprio business erano principalmente due:

1. Acquisto e gestione interna delle infrastrutture fisiche (server, strutture di rete, di raffreddamento..) che potessero soddisfare i livelli di servizio richiesti;

2. Noleggio di risorse da società esterne specializzate con modalità tradizionali di hosting o housing.

Entrambe le soluzioni hanno i loro pregi e i loro difetti specifici: la gestione interna delle strutture ad esempio permette un controllo totale delle prestazioni e della sicurezza a fronte di un investimento cospicuo iniziale ed uno continuato nel tempo per la gestione e manutenzione; l'affidarsi a società terze tramite hosting o housing tradizionali invece toglie il controllo sui livelli di servizio a fronte di una richiesta economica molto più contenuta.

Ecco dove interviene il paradigma del Cloud Computing: utilizzando le tecnologie di virtualizzazione che sono già da tempo a disposizione del mercato IT è possibile allocare e de-allocare dinamicamente le risorse a disposizione di un certo applicativo in base alle necessità dell'applicativo stesso. Questo approccio ha permesso la creazione di listini basati su un modello pay-per-use che consentono di raffinare il costo di un determinato servizio sulla base di quanto questo venga effettivamente utilizzato e non sull'utilizzo massimo che solo saltuariamente viene raggiunto per effetto di Flash-Crowd [2].

Questa prospettiva ha reso il Cloud Computing un'alternativa ai tradizionali strumenti di hosting/housing molto attraente sopratutto per le realtà aziendali medio-piccole su cui i costi relativi alle risorse di calcolo possono incidere in modo molto significativo.

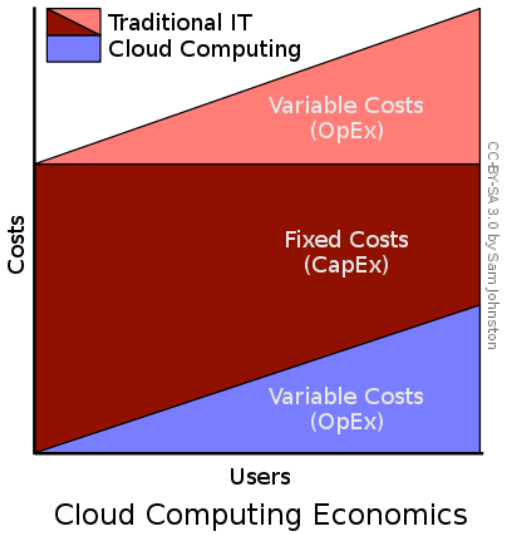

Figura 5: Confronto di costi fra Cloud Computing e sistemi tradizionali

Grazie a tutte queste prerogative il successo del paradigma di Cloud Computing è in costante crescita, la sua diffusione e l'interesse da esso generato sono sempre maggiori.

Metodologia Con l'obiettivo di presentare organicamente quanto ricercato e realizzato per la stesura di questo documento, si procederà con una metodologia deduttiva che presenti i passaggi e le motivazioni tecniche e culturali affrontati durante lo studio di fattibilità realizzato.

Si partirà inizialmente da una descrizione più completa possibile dello stato dell'arte delle tecnologie che gravitano attorno agli obiettivi descritti: verrà descritta la struttura attuale di Iungo, l'utilizzo che ne viene fatto dai suoi utenti e le sue peculiarità tecniche che risultano determinanti nelle scelte progettuali di Iungo Collector. Verrà quindi descritto lo stato dell'arte del Cloud Computing: dalle sue definizioni alle diverse declinazioni presenti sul mercato, all'impatto culturale che sta avendo sulle strategie aziendali e sui modelli di business. Verranno inoltre descritte le tecnologie di comunicazione fra sistemi software ritenute più plausibili per la realizzazione del progetto: Java Remote Method Invocation (RMI) e Webservices (su proto-

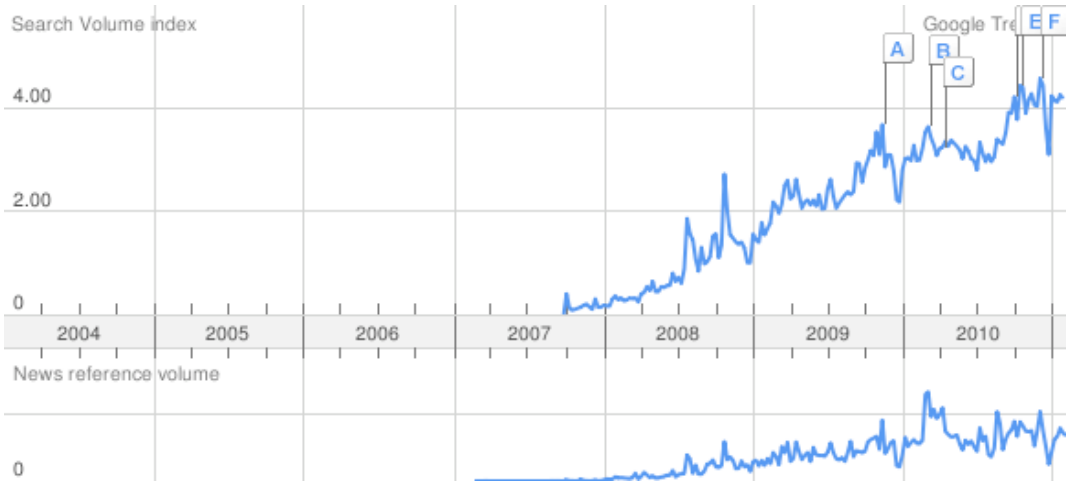

Figura 6: Andamento ricerche della parola "Cloud Computing" registrato da Google Trends

collo SOAP), ciascuna con i suoi punti di forza e punti di debolezza e gli inevitabili compromessi a cui si è giunti.

Quindi verrà presentato il vero e proprio studio di fattibilità: verranno messe a confronto le tecnologie analizzate e verranno illustrate le motivazioni che hanno portato a ritenerne alcune più adatte di altre per la realizzazione di Iungo Collector.

Infine verrà descritta la realizzazione del prototipo di funzionalità implementato: dai suoi aspetti progettuali salienti che hanno dovuto garantire che quanto realizzato fosse effettivamente un dimostratore di funzionalità, ai suoi aspetti implementativi, ai test e misurazioni eseguite.

Risultati attesi Con questa tesi si intende fornire una analisi e un confronto tra le tecnologie disponibili per la realizzazione di un applicativo disposto su un paradigma di Cloud Computing. Nello specifico ci si attende di poter fornire uno studio di fattibilità completo per Iungo Collector che delinei l'insieme di tecnologie migliore per la sua realizzazione e quali siano le criticità maggiori a cui si andrà incontro durante la sua realizzazione e messa in opera.

Capitolo 1

Stato dell'arte

1.1 Iungo

Iungo è un software per la Supply Chain collaboration che permette una comunicazione dinamica e flessibile tra il sistema ERP dell'azienda che lo utilizza e i suoi fornitori mediante l'utilizzo di email contenenti form HTML.

Il fornitore che riceve l'e-mail generata da Iungo compila un form html in essa contenuto (ad esempio una pagina HTML rappresentante un ordine di acquisto) ed invia i dati tramite una comunicazione HTTP POST verso l'installazione di Iungo che l'ha generata, la quale provvede all'elaborazione delle informazioni ricevute basandosi su una logica di workflow configurabile e, quando previsto, esporta i dati ricevuti verso il sistema ERP aziendale.

L'elemento centrale di questo sistema è quindi rappresentato da un workflow costituito dagli stati assunti dalle informazioni, dalle transizioni di stato previste, dalle operazioni che Iungo deve eseguire in seguito ai cambi di stato e, infine, dagli eventi che li scatenano. Per poter scambiare informazioni con gli ERP Iungo deve adattarsi ai processi aziendali del contesto in cui viene inserito e quindi deve disporre di workflows personalizzati per cliente che determinano interazioni eterogenee.

Il successo commerciale ha portato ad avere circa 10.000 fornitori che ricevono email generate da Iungo e molti di essi ricevono email provenienti

da più clienti (installazioni di Iungo) oppure utilizzano essi stessi Iungo per comunicare con i loro fornitori.

1.1.1 Struttura e tecnologie

Dal punto di vista tecnologico Iungo è un applicativo web sviluppato in Java ed utilizza Tomcat come application container. Strutturalmente rispetta il più possibile la suddivisione logica tra i tier di Dati, Logiche e Presentazione. Il tier di Dati è gestito da Apache Turbine [23] che fornisce una interfaccia ad oggetti per l'interazione trasparente via JDBC con i DBMS più diffusi sul mercato (MySql, Microsoft SQL server, Oracle ed altri), il tier di Presentazione è realizzato tramite l'integrazione di Apache Turbine con Apache Velocity [24] e il tier di Logiche -che è il vero cuore del sistema- è interamente realizzato in Java (compatibile con Java $>$ 1.5).

TIER	*INPUT*	*OUTPUT*
Dati	DBMS tra cui MySql, MS Sql Server e Oracle db	Oggetti Java rappresentati i dati dei DBMS e metodi per il recupero di tali dati dai DBMS
Logica	Oggetti Java rappresentanti i dati dei DBMS e metodi per il recupero di tali dati, richieste di elaborazione da parte dell'utente	Dati strutturati e organizzati secondo le logiche di programma e le richieste di elaborazione da parte dell'utente
Presentazione	Richieste dell'utente via HTTP	Pagine HTML inviate via HTTP o come allegati ad email

Tabella 1.1: Interazioni fra i tier di Iungo

1.2 Cloud Computing

In seguito alla sempre più accentuata tendenza a diminuire dei costi delle memorie dati (fig 1.2) e alla sempre maggiore diffusione di internet intesa

anche come crescita del numero di punti di accesso al web gratuiti o a costi molto contenuti, le risorse per il calcolo sono diventate sempre più potenti e raggiungibili indipendentemente dalla posizione (fig 1.1).

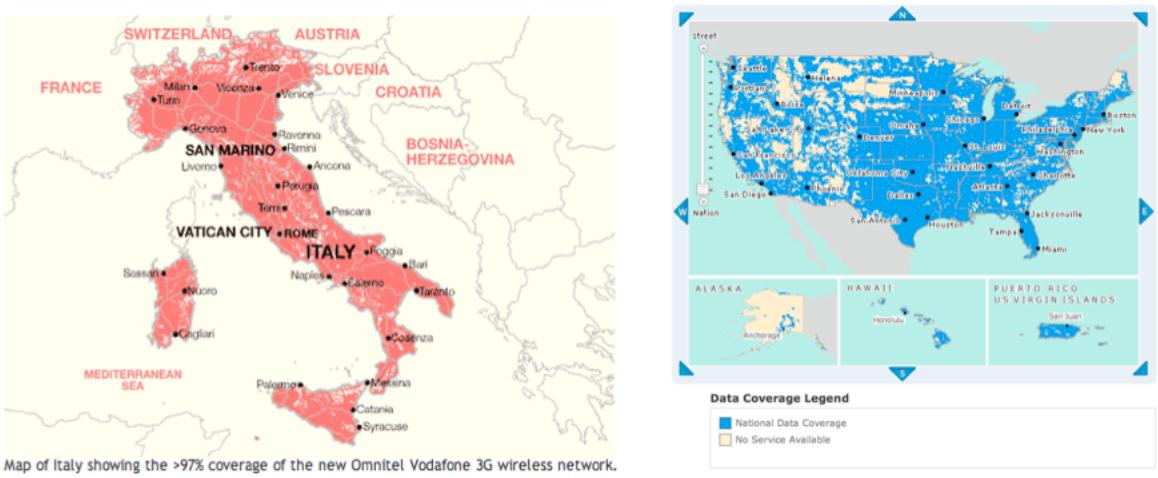

Figura 1.1: Copertura di reti 3G in Italia (Vodafone) e negli Stati Uniti (AT&T)

Questo andamento tecnologico ha permesso la teorizzazione e la realizzazione del paradigma di calcolo definito Cloud Computing in cui le risorse (ad esempio processori e memorie) sono rese disponibili agli utenti sotto forma di "noleggio" e possono essere allocate (cioè messe a servizio di un certo utente) o rilasciate via Internet secondo una modalità on-demand.

L'idea che è alla base del Cloud Computing non è nuova: già negli anni '60 John McCarthy aveva teorizzato che l'accesso alle infrastrutture di calcolo sarebbe stato fornito alle utenze pubbliche in modo simile ad un servizio [4]. Lo stesso termine "Cloud" è già stato usato in passato in vari contesti: ad esempio negli anni '90 veniva usato per descrivere le grandi reti dei Bancomat. Tuttavia il termine "Cloud Computing" ha realmente iniziato a guadagnare poporalità solo dopo che l'amministratore delegato di Google Eric Schmidt lo ha usato per descrivere il modello di business per la

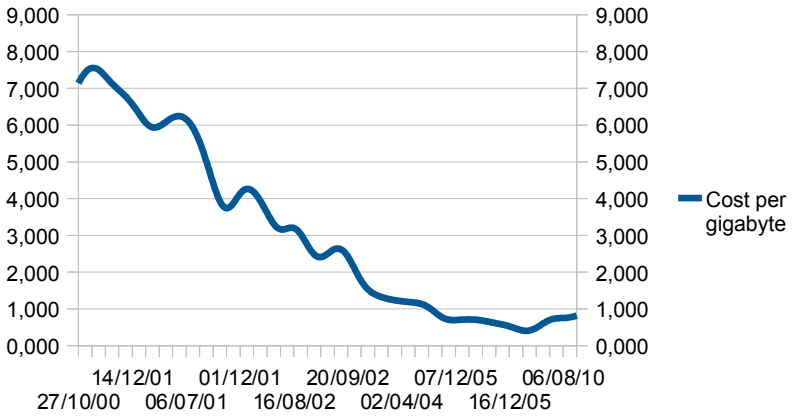

Figura 1.2: Andamento del costo di un Gigabyte (in USD) [3]

fornitura di servizi tramite Internet nel 2006. Da allora il termine "Cloud Computing" è entrato in uso nel gergo del marketing per rappresentare idee anche molto differenti fra loro al punto che, a causa della mancanza di una definizione universalmente conosciuta ed accettata, è diventato uno strumento potente per la generazione di attese (hype) commerciali e fonte di scetticismi, fraintendimenti e confusione.

Per questi motivi si sono visti recentemente diversi tentativi e sforzi che vanno nella direzione di standardizzare la definizione di "Clou Computing". Ad esempio in [5] sono state messe a confronto circa 20 definizioni differenti provenienti da un numero diverso di fonti nel tentativo di creare la definizione standard.

In questa tesi verrà adottata la definizione di "Cloud Computing" fornita da "The National Institute of Standards and Technology" (NIST)[6] che riesce a coprire tutti gli aspetti del Cloud Computing che sono essenziali e di interesse per lo scopo di questa tesi:

<<Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly pro-

visioned and released with minimal management effort or service provider interaction>>

(Il cloud computing è un paradigma per che permette di ottenere un facile accesso di rete on-demand ad un gruppo di risorse computazionali configurabili (ad esempio reti, server, memorie, applicativi e servizi) che possono essere rapidamente allocate e rilasciate con un ridotto impegno di gestione e di interazione con il fornitore del servizio)

La ragione principale che giustifica l'esistenza di percezioni differenti di cloud computing risiede nel fatto che a differenza di altre terminologie tecniche con "Cloud Computing" non si indica una nuova tecnologia, ma piuttosto si indica un nuovo modello operativo che riunisce e mette in relazione un insieme di tecnologie esistenti al fine di ottenere un nuovo paradigma di business. La maggior parte delle tecnologie coinvolte in un Cloud come ad esempio la virtualizzazione, il load balancing e il listino basato sull'utilizzo sono tutt'altro che novità: con il Cloud Computing queste tecnologie esistenti vengono rimodellate e messe insieme per soddisfare le richieste attuali del mercato dell'Information Technology.

Per la realizzazione di Iungo Collector si è ritenuto fondamentale l'utilizzo del paradigma di Cloud Computing. Tale scelta è stata determinata dalla convinzione che sia necessario per la realizzazione di un software in grado di crescere e sostenere la rapida evoluzione del mercato in cui opera l'utilizzo di quanto più recente, innovativo e flessibile il panorama tecnologico metta a disposizione.

Tuttavia come accennato in questa sezione, il termine "Cloud Computing" indica un'idea e una modalità operativa più che una vera e propria tecnologia specifica: nelle sezioni seguenti verrà fornita una panoramica più completa possibile dello stato dell'arte del Cloud Computing e in sezione 1.2 verranno discusse ed esaminate le varie scelte che il mercato offre ad una realtà aziendale come quella di Smarten s.r.l..

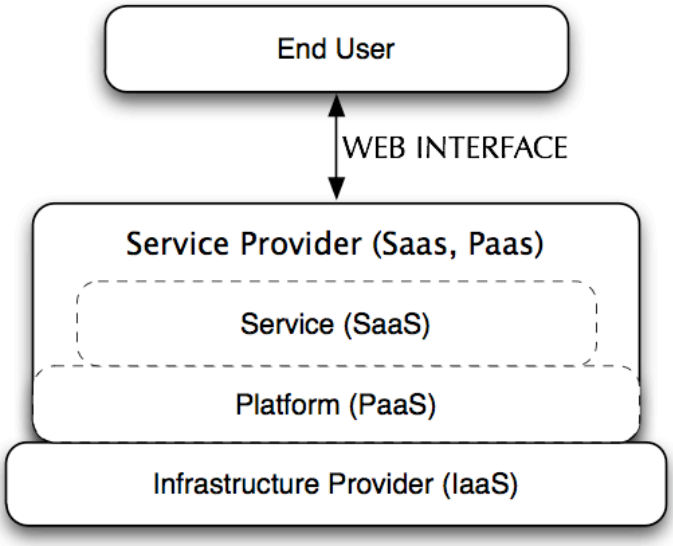

Figura 1.3: Struttura dei providers

1.2.1 Panoramica generale

In un ambiente di Cloud Computing quello che era convenzionalmente inteso essere il ruolo dell'erogatore di un servizio è suddiviso in due ruoli solitamente interpretati da due entità distinte:

1. Un ruolo è detto "infrastructure provider" e gestisce le piattaforme di Cloud Computing mettendo a disposizione tramite noleggio le risorse di tali piattaforme seguendo un modello di listino pay-per-use, basando cioè il costo di erogazione del servizio sull'effettivo tempo di fruizione delle risorse;

2. L'altro ruolo è detto "service provider" che noleggia una o più delle risorse messe a disposizione dall'infrastructure provider, implementa un servizio che utilizzi le risorse così ottenute e lo mette a disposizione dell'utente finale rendendo a lui trasparente la provenienza e la composizione fisica di tali risorse.

La diffusione sempre più ampia di questo paradigma ha avuto negli ultimi anni e sta avendo tutt'ora un impatto molto significativo su tutta l'industria dell'Information Technology (IT) portando i grandi colossi come Google, Amazon e Microsoft a posizionarsi nel mercato dei Cloud Computing providers mettendosi in concorrenza per fornire agli utenti delle loro Cloud un servizio sempre più potente, affidabile ed economicamente conveniente. D'altro canto le imprese IT stanno rivedendo i loro modelli di business, le loro politiche aziendali e le strutture dei loro applicativi con l'obiettivo di trarre il maggior beneficio possibile dalle piattaforme di Cloud Computing.

Di seguito verrà fornito un elenco delle principali caratteristiche che rendono il paradigma di Cloud Computing molto attraente per le imprese del settore IT:

- *Nessuna barriera iniziale*: come già accennato in precedenza, il Cloud Computing è basato su un modello tariffario pay-per-use che non prevede un costo iniziale per la creazione della infrastruttura che si andrà ad utilizzare nè in termini economici nè in termini di tempo, know-how tecnico e risorse umane. L'impresa IT noleggia una infrastruttura Cloud che risponda alle sue esigenze ed è immediatamente pronta all'uso;

- *Riduzione dei costi operativi*: le risorse (processori e memorie) di una piattaforma di Cloud Computing possono essere allocate e rilasciate rapidamente in base alle esigenze d'uso del momento. Questo fa si che non sia necessario premunirsi in forma stabile di una eccedenza di risorse per far fronte ad eventuali e saltuari picchi d'utilizzo: nel periodo di basso utilizzo verranno allocate poche risorse, nei momenti di picco d'uso verranno allocate più risorse con tempi pressochè istantanei che verranno rilasciate quando le richieste d'uso torneranno su livelli più bassi. Questa modalità di allocazione e rilascio delle risorse insieme al calcolo dei costi basato sull'utilizzo risorse effettivamente allocate nel corso del tempo permettono di ottenere risparmi molto significativi;

- *Elevata scalabilità*: gli infrastructure providers possono attingere ad una grande quantità di risorse messe a disposizione da centri di calcolo e renderle velocemente disponibili all'uso da parte dei service providers i quali possono di conseguenza espandere i loro servizi su larga scale per gestire i rapidi incrementi d'uso;

- *Accesso semplice*: gli applicativi disposti su un Cloud sono provvisti generalmente di un accesso web-based e pertanto risultano facilmente accessibili da un gran numero di dispositivi forniti di connettività Internet. Tali dispositivi possono comprendere non solo desktop o laptop computer ma anche smartphones e PDA;

- *Riduzione dei rischi d'impresa e delle spese di manutenzione*: il mettere in outsource la costruzione e la manutenzione della infrastruttura su cui disporre i propri applicativi permette di spostare i rischi d'impresa (ad esempio rotture hardware) verso gli infrastructure providers che solitamente possiedono il know-how e gli strumenti necessari per farvi fronte. L'outsourcing permette inoltre di tagliare i costi di manutenzione e aggiornamento dell'hardware e di ridurre i costi di formazione dello staff.

A fronte di tutti i vantaggi sopra descritti che il paradigma di Cloud Computing può portare sono presenti diversi punti di criticità che richiedono una attenta valutazione da parte dei service providers:

- *Riduzione di controllo*: l'outsourcing totale della infrastruttura su cui andare a disporre i propri servizi riduce fortemente il controllo del service provider sulla struttura principalmente su due versanti:

    - *Qualita'*: il Service Level Agreement (SLA) dei servizi erogati dovrà per forza di cose avere come limite superiore il SLA concordato con l'infrastructure provider: se ad esempio l'infrastructure provider garantisce un uptime del 90% il service provider non

potrà garantirne uno del 91%. Allo stesso modo il livello di performance dei servizi sarà strettamente legato a quello messo a disposizione dall'infrastructure provider che -per quanto grandi possano essere le risorse a cui può attingere- è inevitabilmente finito e soggetto agli andamenti del suo mercato e del suo parco clienti;

- *Scelte strategiche*: nella maggior parte dei casi l'infrastruttura è parte di primo rilievo in scelte strategiche aziendali dei service provider: non è detto che le priorità di un service provider coincidano con quelle del suo infrastructure provider e di fatto si introduce un ulteriore (e potenzialmente ostile in quanto esterno alla struttura aziendale) stake-holder che può giocare un ruolo anche decisivo nelle scelte strategiche del service provider.

- *Trust*: il delegare a terzi la gestione fisica di dati e applicativi critici per il proprio business introduce una intera classe di problemi che riguardano la fiducia (trust, appunto) che l'utente finale deve riporre nel service provider e che lo stesso service provider deve riporre nell'infrastructure provider:

  - *Trattamento corretto dei dati*: trattandosi di informazioni critiche per il proprio business diventa indispensabile che i dati affidati al service provider e all'infrastructure provider siano trattati nel modo corretto con garanzie di segretezza, reperibilità e integrità;
  - *Rispetto del SLA*: la necessità per il service provider di garantire agli utenti finale un certo SLA diventa una condizione di criticità in quanto tale garanzia si basa sul rispetto da parte di una entità su cui ha pochissimo controllo (l'infrastructure provider) del SLA con lui concordato per la fruizione delle risorse computazionali;
  - *Disaster recovery*: nel caso del Cloud Computing e sopratutto nel momento storico attuale con "disaster" si può intendere più

che un cataclisma naturale (che comunque può essere prevenuto mediante una opportuna replicazione dei dati), il fallimento o acquisizione da parte di altre società del service o infrastructure provider che potrebbero quindi improvvisamente e senza preavviso interrompere l'erogazione del loro servizio.

1.2.2 Struttura del Cloud Computing

Generalmente l'architettura del Cloud Computing può essere vista come una serie di "layer" (strati):

- *Hardware layer*: questo layer è il responsabile della amministrazione delle risorse fisiche del cloud, risorse che possono includere server fisici, routers, switches, sistemi di alimentazione e di raffreddamento. Tipicamente il layer hardware è implementato in un data center che solitamente contiene migliaia di server organizzati in "rack" (armadi) e sono connessi fra loro mediante switch, routers ed altre infrastrutture. L'hardware layer viene coinvolto principalmente in questioni che possono riguardare la configurazione hardware, la tolleranza ai guasti, la gestione del traffico (su rete), potenza e raffreddamento;

- *Infrastructure layer*: è anche detto "virtualization layer" e si occupa principalmente di gestire le richieste di risorse computazionali e di storage andando ad utilizzare le risorse messe a disposizione dall'hardware layer mediante tecnologie di virtualizzazione come Xen [7], KVM [8] e Vmware [9]. L'infrastructure layer è parte fondamentale di un Cloud Computing visto che molte delle caratteristiche principali di questo paradigma sono possibili solo grazie alle tecnologie di virtualizzazione;

- *Platform layer*: il platform layer' è posizionato sopra l'infrastructure layer ed è principalmente formato dai sistemi operativi e dai framework applicativi. Lo scopo principale del platform layer è quello di alleggerire il peso dell'installazione degli applicativi direttamente nei

Virtual Machine (VM) containers. Ad esempio il Cloud Computing di Google (Google App Engine) opera a livello di Platform layer fornendo Application Programming Interfaces (API) [10] per l'implementazione di storage, database e logiche di business tipiche di applicativi web;

- *Application layer*: l'application layer è posizionato alla cima della scala gerarchica dei layer e consiste delle reali applicazioni installate sul Cloud. A differenza delle applicazioni "tradizionali", gli applicativi su Cloud possono sfruttare le caratteristiche di auto-dimensionamento delle risorse messe a disposizione dai layer inferiori per raggiungere migliori performance, maggiore disponibilità (uptime) e ridotti costi operativi.

Se confrontati con i layer dei tradizionali contesti di application hosting (ad esempio server farm), l'architettura di un Cloud risulta più modulare e meno monolitica in maniera analoga al modello ISO-OSI [11] per i protocolli di rete: ogni layer è collegato in modo sufficientemente blando con quello superiore e quello inferiore da permettergli una evoluzione abbastanza indipendente.

Questa modularità architetturale permette al Cloud Computing di poter ospitare un vasto numero di applicativi e contemporaneamente di ridurre l'overhead (di tempi e costi) di manutenzione e gestione.

1.2.3 Modelli di business

Il Cloud Computing realizza un modello di business orientato ai servizi: le risorse a livello hardware e di piattaforme vengono fornite on-demand. Dal punto di vista concettuale ciascuno dei livelli descritti precedentemente può essere implementato come un servizio fornito al livello superiore e viceversa ciascuno livello può essere visto come "utente" del livello inferiore.

Tuttavia nella pratica il Cloud viene visto come un insieme di servizi che possono essere raggruppati in tre categorie: Software as a Service (SaaS), Platform as a Service (PaaS) ed Infrastructure as a Service (IaaS):

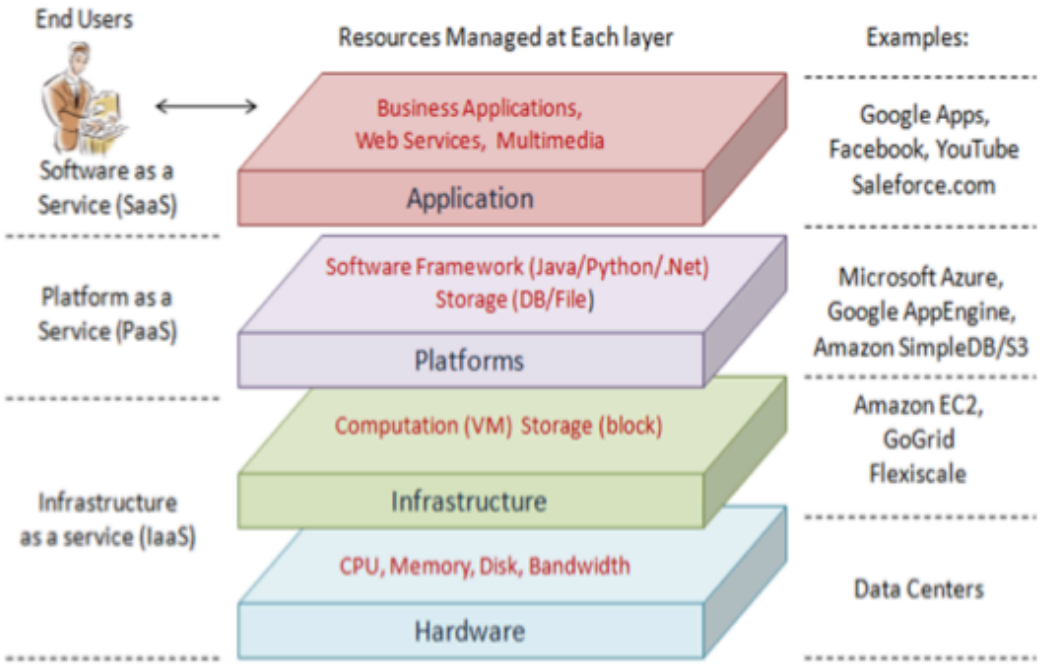

Figura 1.4: Struttura a livelli del Cloud Computing

Software as a Service Attraverso questo modello di business il Cloud fornisce applicativi (ad esempio servizi on demand) in modalità centralizzata accessibili via Web. I software offerti sono condivisi tra tutti gli utenti e possono differire tra loro solo per funzionalità opzionali richieste e abilitate durante la sessione di configurazione. L'applicativo utilizzato pur essendo condiviso tra vari clienti garantisce la separazione logica dei dati di questi ultimi. Il software, mediante un'architettura hardware virtualizzata, può essere inoltre fornito in modalità multi-istanza dedicata ciascuna a un cliente diverso. Tali applicativi mettono spesso a disposizione interfacce Web Services che consentono l'integrazione e l'interoperabilità con altri software al fine di sviluppare nuove applicazioni di tipo SOA (Software Oriented Architeture). Una caratteristica di questo paradigma è l'incremento dell'affidabilità sia per quanto riguarda il salvataggio che l'elaborazione delle risorse mediante data center interni al provider stesso. Sarà compito di quest'ultimo infatti assicurare la replicazione e il backup sollevando l'utente da tale onere. Mediante l'uso di licenze di tipo on demand si possono ottenere servizi a costo variabile, a seconda del vero utilizzo, anziché pagare per licenze complete al primo acquisto. Utilizzando licenze di tipo "pay-as-you-go" vengono pagate solo le risorse utilizzate: ad esempio viene conteggiato il numero di ore utilizzate del servizio, il carico computazione richiesto etc. Tutto ciò comporta un notevole risparmio per le risorse utilizzate minimizzando gli sprechi.

Caratteristiche principali:

- accesso al software mediante Web e senza nessuna installazione sul proprio elaboratore;
- gestione centralizzata delle attività da un'unica sede centrale evitando la singola configurazione di ogni elaboratore di ciascun utente;
- distribuzione delle applicazione in modalità "one-to-many" (con architettura multi-tenant) in modo che ciascuna istanza serva molteplici clienti;

- aggiornamenti software centralizzati che evitano a ciascun utente di scaricare patches e upgrades manualmente;

- possibilità di integrazione con altri software sia come parte di Mashup che come plugin per quelli esistenti.

Infrastructure as a Service talvolta chiamato Hardware as a Service, è un modello di Cloud Computing in cui un'organizzazione mette a disposizione servizi infrastrutturali che possono essere divisi in tre categorie principali:

- Dispositivi: quali servers aziendali, dispositivi di storage, di rete e di sicurezza;

- Impianti e strutture: alloggiamenti per i dispositivi sopracitati, data centers, sistemi di raffreddamento, generatori di potenza e sistemi di backup e di sicurezza;

- Sistemi di gestione: per il monitoraggio della performance dell'infrastruttura sia on-site che da remoto e per la modica delle impostazioni in caso di problemi.

Risulta quindi chiaro che una delle tecnologie di sviluppo alla base di IaaS è la virtualizzazione: la creazione di una versione virtuale di una risorsa normalmente fornita fisicamente. Grazie ad essa è infatti possibile partizionare un certo apparato in più sottosistemi in grado di offrire molteplici risorse il più possibile specializzate. Tutto ciò ha permesso alle aziende che richiedono questo tipo di servizi di essere indipendenti rispetto all'acquisto, al mantenimento, alla gestione e alla svalutazione di risorse hardware; ci si preoccuperà infatti solo dell'acquisto di cicli macchina, capacità di storage, larghezza di banda necessarie per l'esecuzione delle proprie applicazioni. Il modello tariffario dell'IaaS che meglio si adatta è il "pay-as-you-go" in quanto risulta più semplice, rispetto ad applicativi software, conteggiare l'uso di risorse effettivamente utilizzate ed applicare quindi la relativa tariffa.

Platform as a Service Platform as a Service è una architettura in grado di fornire piattaforme di sviluppo software accessibili direttamente attraverso la Cloud mediante l'utilizzo di un browser. Ad esempio vengono messi a disposizione strumenti di sviluppo come workflow, strumenti di creazione di interfacce web, database integration, storage e integrazione web-services. Sarà compito del provider incaricarsi delle decisioni riguardanti l'ambiente in cui il software verrà sviluppato ed eseguito, del sistema operativo messo a disposizione, le relative API, il linguaggio di programmazione, le varie configurazioni della piattaforma esonerando quindi l'utente da tali compiti e lasciandolo concentrato sulla parte di sviluppo vero e proprio. Anche questo paradigma, come l'IaaS, fa uso della virtualizzazione: tale infrastruttura permette allo sviluppatore di scrivere software senza preoccuparsi della struttura hardware sottostante.

PaaS differisce notevolmente dai classici strumenti di sviluppo per Web Application e comprende una serie di funzionalità:

- strumento di sviluppo multi-tenant in grado di supportare più utenti ciascuno dei quali con più di un progetto all'attivo;
- la scalabilità dell'applicazione e dei dati deve essere built-in ovvero devono essere elementi base dell'architettura stessa (come ad esempio il bilanciamento di carico, la resistenza ai guasti etc);
- le classiche soluzioni di sviluppo non si preoccupano del controllo a run-time, ma con PaaS questa caratteristica deve essere parte della piattaforma di sviluppo;
- PaaS non richiede l'installazione di ulteriore software di sviluppo sulla propria macchina a parte un qualsiasi web browser;
- semplicità di installazione della propria applicazione sviluppata con un semplice click del mouse; fatturazione "pay-as-you-go" quindi senza costi iniziali di start-up.

Con PaaS si è fatto un notevole passo in avanti nel mondo degli sviluppatori: non sono più richieste specifiche competenze per sviluppare applicazioni Web.

1.2.4 Tipi di cloud

Tra le varie considerazioni che una impresa IT deve fare nella realizzazione di un applicativo da disporre su un cloud una risulta centrale e discriminante: si deve dare priorità ai bassi costi operativi o alla elevata raggiungibilità e sicurezza?

Sulla base di questa domanda si possono delineare tre differenti tipi di cloud, ciascuno con i propri punti di forza e le proprie debolezze:

- **Cloud pubbliche**: ovvero cloud in cui il service provider mette a disposizione le sue risorse come servizi per il pubblico utilizzo. I sistemi Cloud pubblici offrono diversi importanti benefici ai service provider: l'abbattimento dei costi iniziali di start-up e la delega della gestione dei rischi all'infrastructure provider. Tuttavia le cloud pubbliche soffrono della mancanza di controllo assoluto sui dati, reti ed impostazioni di sicurezza, cosa che ne inficia l'usabilità e l'efficienza in molti scenari di business. Un esempio di public cloud e' OpSource Cloud Hosting [20];

- **Cloud private**: sono dette anche "cloud interne" e sono realizzate per l'uso esclusivo di una singola organizzazione. Una cloud privata può essere gestita direttamente dall'organizzazione o da un provider esterno. Se da un lato questo tipo di cloud fornisce il maggior controllo possibile sulle performance, affidabilità e sicurezza, d'altro canto risultano essere praticamente identiche alle tradizionali server farm e rendono indispensabile un cospicuo investimento iniziale;

- **Cloud ibride**: è una via di mezzo tra i modelli di Cloud pubbliche e di Cloud private che cerca di ridurre i limiti di ciascuno dei due approcci. In una cloud ibrida una parte delle infrastrutture di servizio è contenuta in una cloud privata, mentre la rimanente è nella cloud pubblica.

Le cloud ibride risultano essere più flessibili sia delle cloud private che di quelle pubbliche e in particolare rispetto alle cloud pubbliche forniscono un maggiore controllo sulla sicurezza sui dati degli applicativi mantenendo comunque la capacità di far fronte agli incrementi e alle contrazioni di domande d'utilizzo degli utenti finali. D'altro canto la progettazione di una cloud ibrida richiede una accurata analisi sulla separazione dei componenti che devono appartenere alla parte pubblica della cloud e quelli che appartengono alla parte privata. Un esempio di private cloud e' Microsoft Private Cloud basate su Microsoft Hyper-V Cloud [21];

- **Cloud private virtuali**: una soluzione alternativa che cerca di ridurre le limitazioni delle cloud private e pubbliche è chiamata Virtual Private Cloud (VPC). Una VPC è essenzialmente una piattaforma posizionata su una cloud pubblica che sfrutta le tecnologie di Virtual Private Network (VPN) permettendo al service provider di progettare la propria topologia di rete e i propri meccanismi di sicurezza (come ad esempio regole del firewall). La VPC prevede sostanzialmente una progettazione più complessa e completa dal momento che non vengono virtualizzate solo server e applicazioni ma anche la rete sottostante. Inoltre nel caso di migrazione di applicativi da una rete privata ad una cloud, la VPC permette una transizione senza richieste particolari di modifiche visto che viene riprodotta (virtualizzata) la rete stessa su cui l'applicativo era già stato progettato.

Era stato predetto che le Cloud ibride sarebbe state il tipo dominante più diffuso per la maggior parte delle organizzazioni [12], tuttavia dal loro concepimento nel 2009 le VPC stanno guadagnando sempre più poplarità e sul medio termine mettono in discussione il dominio delle ibride. Un esempio di VPC e' Amazon VPC [22].

1.2.5 Caratteristiche del Cloud Computing

Il Cloud Computing mette a disposizione diverse caratteristiche salienti che sono diverse dai tradizionali modelli di service computing:

- *Co-proprietà*: in un ambiente di cloud computing i servizi appartengono a provider diversi che sono allocati contemporaneamente nello stesso data center. Le problematiche di performance e amministrazione di questi servizi sono condivise tra il service provider e l'infrastructure provider. L'architettura basata sui layer che abbiamo descritto in precedenza fornisce una suddivisione naturale di responsabilità: il proprietario di ciascun layer deve focalizzarsi solamente su specifici obiettivi relativi al suo layer. Tuttavia questa condizione di co-proprietà introduce difficoltà in comprensione e gestione delle interazione tra i vari stakeholders che possono avere obiettivi e priorità molto differenti se non opposte;

- *Accesso alle risorse condiviso*: l'infrastructure provider fornisce un accesso a risorse computazione che può essere assegnato dinamicamente a più utilizzatori: questo sistema di assegnazione dinamico delle risorse fornisce elevata flessibilità agli infrastructure providers nella gestione dell'utilizzo delle loro risorse e nel controllo dei costi operativi. Per esempio un provider IaaS può sfruttare le tecnologie di migrazione delle macchine virtuali per raggiungere un alto livello di solidità dei server da un lato massimizzando l'utilizzo delle risorse e dall'altro minimizzando i costi derivanti da fattori come il consumo energetico e il raffreddamento;

- *Distribuzione geografica e accesso globale*: le Cloud sono solitamente accessibili tramite Internet ed utilizzano Internet come una rete per fornitore un servizio. Ne consegue che qualunque dispositivo in grado di connettersi ad Internet (che sia un cellulare, un PDA, un netbook, laptop o desktop) può accedere ed usufruire dei servizi messi a disposizione dal Cloud. Per ottenere il massimo dalle performance di rete

molte delle Cloud in uso oggi sono formate da data center dislocati in diversi posti del pianeta e quindi un service provider può sfruttare questa disposizione per massimizzare le performance e la raggiungibilità del servizio;

- *Service-oriented*: come già riportato precedentemente il cloud computing è basato su un modello operativo orientato ai servizi e pertanto pone un grande risalto sulla gestione dei servizi. In una Cloud i provider di IaaS, PaaS e SaaS mettono a disposizione i propri servizi sulla base di un SLA concordato con i loro clienti e pertanto il rispetto e la garanzia del SLA diventa un obiettivo critico di ciascun provider;

- *Approvvigionamento dinamico delle risorse*: una delle caratteristiche fondamentali del Cloud Computing risiede nel fatto che le risorse computazionali possono essere ottenute e rilasciate "al volo". Rispetto ai modelli tradizionali di approvvigionamento delle risorse che calcolano le esigenze basandosi sui momenti di picco delle domande d'utilizzo, il modello di approvvigionamento dinamico permette ai service providers di ottenere (e pagare per) risorse in base alla domanda d'utilizzo attuale, cosa che può diminuire considerevolmente i costi operativi;

- *Auto-organizzazione*: dal momento che le risorse possono essere allocate o rilasciate on-demand, i service provider hanno la possibilità di gestire il loro consumo di risorse sulla base delle loro esigenze. Inoltre la gestione automatica delle risorse garantisce un grande livello di flessibilità che permette ai service providers di rispondere rapidamente ai picchi improvvisi di richieste d'utilizzo come ad esempio gli effetti di "flash-crowd";

- *Listino basato sull'utilizzo:* il Cloud Computing implementa un modello di listino basato sul concetto di "pay-per-use". L'esatto schema di listino può variare da servizio a servizio: per esempio un provider SaaS può noleggiare da un provider IaaS una macchina virtuale con

un sistema di prezzi basato sulle ore d'utilizzo effettivo, e allo stesso modo un provider SaaS che fornisce un servizio on-demand di Customer Relationship Management (CRM) può fatturare ai suoi clienti basandosi sul numero di utenti, piuttosto che sulla dimensione dei dati gestiti (ad esempio come fa Salesforce). Il listino basato sull'utilizzo aiuta a diminuire i costi operativi dal momento che il cliente paga per il servizio in base a quanto effettivamente ne ha usufruito, tuttavia aggiunge complessità al controllo dei costi e la definizione del listino può non essere triviale. Per far fronte a questa crescente complessità società come VKernel [13] fornisce software che può aiutare i clienti di Cloud a capire, analizzare e tagliare i costi inutili dovuti al consumo di risorse.

1.2.6 Prodotti commerciali

Di seguito vengono proposti a titolo esemplificativo alcuni dei servizi Cloud attualmente in commercio:

Amazon EC2

Sicuramente il servizio commerciale di più grande successo in ambito Cloud Computing è Amazon Web Services (AWS). Una delle componenti di AWS più diffusa in ambito IaaS è Amazon Elastic Compute Cloud (Amazon EC2).

Amazon EC2 fornisce capacità computazionale sotto forma di server virtuali "in the cloud" [14]. Uno dei vantaggi principali di EC2 risiede nella possibilità di aumentare e diminuire in modo semplice e veloce le risorse richieste dal cliente. Amazon EC2 è a tutti gli effetti un Web Service e permette di istanziare vere e proprie macchine virtuali "in the cloud" e con la possibilità quindi di installare differenti sistemi operativi e applicazioni, configurazione di utenti e permessi, criteri di accessibilità attraverso la rete.

Una volta creata e configurata la macchina virtuale (denominata AMI Amazon Machine Image) è possibile gestirla come un vero e proprio server con la funzionalità aggiuntiva di poter integrare gli altri servizi di Amazon.

Caratteristiche principali del servizio:

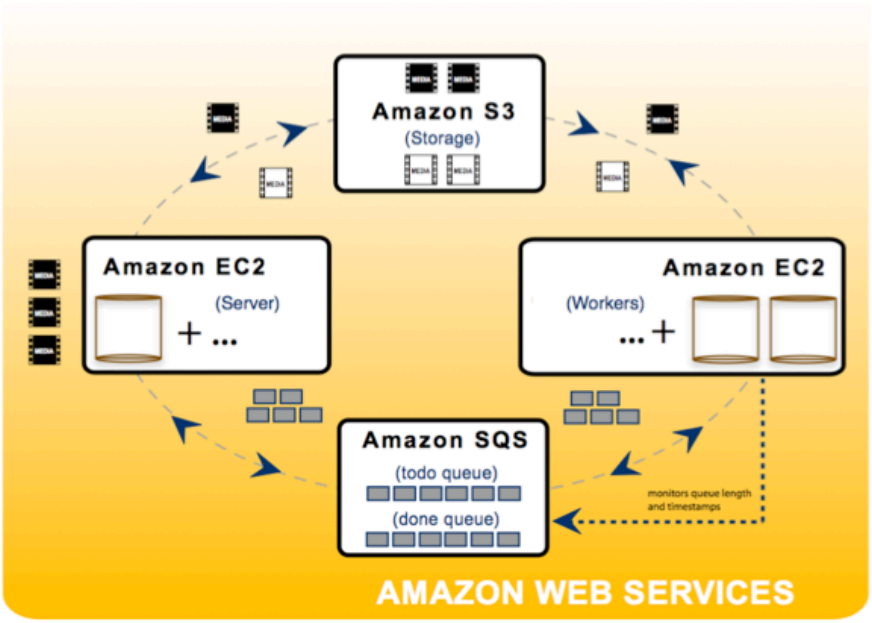

Figura 1.5: Servizi ed interazioni fornite da Amazon

- *Elasticità*: possibilità di aumentare o diminuire la capacità del sistema in pochi minuti. Possono essere acquistate una, centinaia o migliaia di istanze server simultaneamente.

- *Controllo completo*: il cliente ha un controllo completo sulle proprie istanze in modalità amministratore effettuabili attraverso la rete stessa: avvio, spegnimento e reboot del server.

- *Flessibilità*: si possono scegliere differenti tipi di istanze server, sistemi operativi e applicativi software. Amazon EC2 consente inoltre di scegliere una configurazione per CPU, memoria, capacità di storage e la dimensione per la partizione di boot ottimale per il sistema operativo e l'applicazione scelta.

- *Integrazione con altri servizi AWS*: EC2 può essere integrato facilmente con gli altri servizi AWS quali Amazon Simple Storage Service (Amazon S3), Amazon SimpleDB and Amazon Simple Queue Service (Amazon SQS) e altre tecnologie messe a disposizione dalla Cloud.

- *Affidabilita'*: Amazon EC2 offre un ambiente affidabile dove ogni istanza può essere sostituita in modo rapido. Il servizio è attivo all'interno dell'infrastruttura della rete di Amazon e dei suoi data center.

- *Sicurezza*: sono garantiti molti aspetti di sicurezza quali firewall, Amazon Virtual Private Cloud (VPC) per isolare le proprie istanze specificando un range di indirizzi IP e IPsec VPN per cifrare le comunicazioni tra due o più reti private.

- *Economicità*: mediante il paradigma "pay-per-use" con Amazon EC2 si pagherà soltanto per quello che effettivamente si usa. Amazon EC2 mette a disposizione tre tipi di istanza: Standard, High-Memory e High-CPU. Come unità di misura per la potenza computazionale usa l'EC2 Compute Unit (ECU) che corrisponde all'equivalente di 1.0-1.2 Ghz di una CPU Opteron o Xean del 2007.

Figura 1.6: Servizi di Microsoft Windows Azure

Microsoft Azure Services Platform

Microsoft Azure Services Platform [14] è una piattaforma che fornisce strumenti agli sviluppatori informatici per la creazione di Cloud Application, agli Independent Software Vendors (ISVs) per raggiungere rapidamente il mercato mediante il paradigma "pay-as-you-go", agli IT Managers per accedere a nuove risorse senza aggiungere complessità e alle aziende di qualsiasi dimensione per rispondere rapidamente alle loro nuove esigenze aziendali.

Mediante questa architettura è possibile realizzare nuove applicazioni da eseguire "in the cloud" o anche migliorarne di esistenti aggiungendo le caratteristiche tipiche del Cloud Computing. Microsoft Azure Services Platform è a tutti gli effetti un gruppo di tecnologie di tipo Cloud ciascuna delle quali offre una suite di servizi specifici agli sviluppatori di applicazioni.

Di seguito è riportata un schema dell'architettura di Azure:

Come si vede dall'immagine Azure offre una serie di servizi quali: LiveServices per l'interscambio di informazioni tra i propri utenti e tra le applicazioni o siti che ne fanno uso, .NetServices che offre lo strato software per la re-

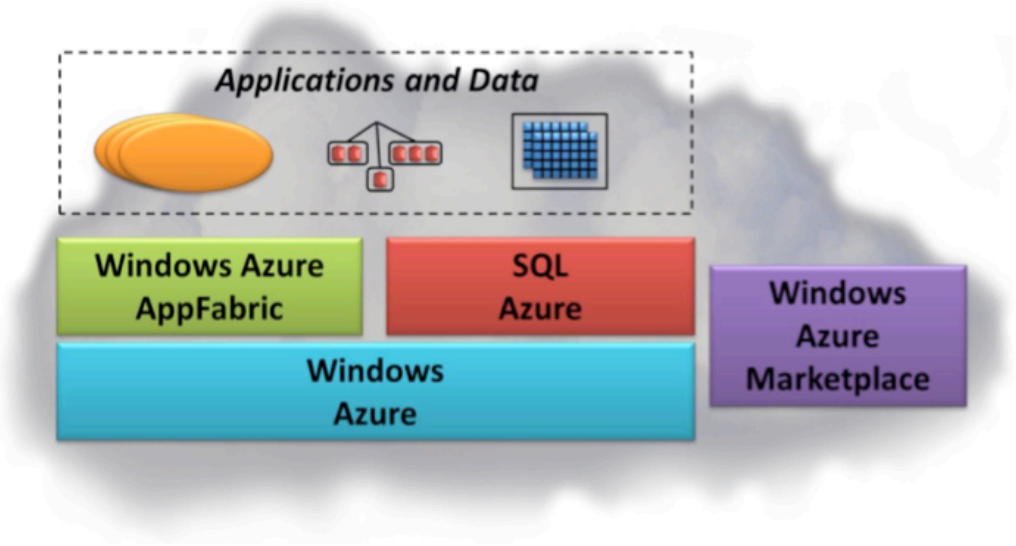

Figura 1.7: Componenti che si appoggiano su Windows Azure

alizzazione delle applicazioni "in the cloud" basandosi sulla piattaforma già esistente .NET e SQLServices per offrire un database relazionale basato sulle tecnologie SQL Server in grado di fornire affidabilità, scalabilità e sicurezza.

Il cuore di questa tecnologia è il sistema operativo Windows Azure, realizzato per essere eseguito "in the cloud" il quale fornisce funzionalità di basso livello come hosting, sicurezza, storage ed elaborazione, e supporta nativamente le tecnologie di virtualizzazione. Attraverso gli strumenti messi a disposizione da Windows Azure, gli sviluppatori possono creare applicazioni net-oriented il cui hosting può essere effettuato sia su server remoti accessibili via Internet, sia sul data center locale di un'azienda.

Windows Azure è un a piattaforma che supporta linguaggi di programmazione e protocolli multipli come SOAP, REST, XML e PHP. Come mostra l'immagine, per creare applicazioni e servizi su Windows Azure, gli sviluppatori possono utilizzare le loro attuali competenze sfruttando Microsoft VisualStudio; inoltre Azure permette di integrare lo sviluppo anche attraverso linguaggi e tecnologia non-Microsoft.

Le principali caratteristiche offerte da Windows Azure sono:

- possibilità di eseguire applicazioni Web in ASP.NET o in .NET "in the cloud";
- ambiente di hosting IIS 7 e .NET Framework 3.5 SP1;
- interfaccia Web per accedere alle varie funzionalità del sistema;
- supporto FastCGI per consentire agli sviluppatori di installare ed eseguire applicazioni Web scritte in un linguaggio non-Microsoft come ad esempio PHP;
- salvataggio di oggetti, tabelle e code "in the cloud";
- sistema di autenticazione e tripla replicazione per mantenere i dati del cliente sicuri;
- strumenti di sviluppo offline e accessibili mediante riga di comando;
- plugin di Visual Studio per effettuare il debugging in locale;
- strumenti Windows Azure per Microsoft Visual Studio che estendono Visual Studio 2008 e Visual Studio 2010 Beta 1 per gestire la creazione, lo sviluppo, il debugging, l'esecuzione e la pacchettizzazione di applicazioni e servizi Web scalabili su Windows Azure.

Google App Engine

Google App Engine [10] è una piattaforma per applicazioni web tradizionali da installare in data centers gestiti da Google. Attualmente i linguaggi di programmazione supportati sono Python e Java. I framework che eseguono su Google App Engine includono Django, CherryPy, Pylons, web2py insieme ad un web application framework sviluppato da Google simile a JSP o Asp.Net.

In questo contesto google si occupa di disporre codice sul cluster, monitorare, recuperare da crash e lanciare istanze degli applicativi secondo necessità. Attualmente le API forniscono possibilità di scrivere e recuperare dati dal database non relazionale BigTable [16], creare richieste HTTP e gestire

Cloud Provider	Amazon EC2	Windows Azure	Google App Engine
Classes of Utility Computing	Infrastructure service	Platform service	Platform service
Target Applications	General-purpose applications	General-purpose Windows applications	Traditional web applications with supported framework
Computation	OS Level on a Xen Virtual Machine	Microsoft Common Language Runtime (CLR) VM; Predefined roles of app. instances	Predefined web application frameworks
Storage	Elastic Block Store; Amazon Simple Storage Service (S3); Amazon SimpleDB	Azure storage service and SQL Data Services	BigTable and MegaStore
Auto Scaling	Automatically changing the number of instances based on parameters that users specify	Automatic scaling based on application roles and a configuration file specified by users	Automatic Scaling which is transparent to users

Figura 1.8: Tabella di confronto fra tre diversi provider

cache e gli sviluppatori hanno permessi di sola lettura sul file-system di App Engine.

1.3 Java Remote Method Invocation

Durante l'analisi delle tecnologie per l'implementazione della struttura di rete di Iungo Collector si sono prese in considerazione due possibili alternative: Java Remote Method Invocation (più semplicemente RMI) e WebServices su SOAP.

In questa sezione verrà fornita una analisi dello stato dell'arte di Java RMI.

Java RMI è una tecnologia sviluppata originariamente da Sun Java (ora di proprietà di Oracle) utilizzata per il calcolo distribuito in linguaggio Java.

Java RMI fornisce un modello semplice e diretto per la computazione distribuita utilizzando oggetti Java. Questi oggetti possono essere creati apposi-

tamente per la necessità o possono essere realizzati come semplici "wrapper" (involucri) di oggetti esistenti.

L'obiettivo principale di RMI è quello di conciliare le politiche di sicurezza e di portabilità del codice Java alle potenzialità del calcolo distribuito permettendo di spostare le logiche di comportamenti e gli agenti nella posizione della rete a loro più congeniale e realizzando quindi una separazione fisica oltre che teorica tra i vari layer che possono comporre un applicativo.

RMI può essere anche utilizzato per connettere nuove componenti a sistemi esistenti "legacy" mediante l'uso dello standard Java Native method Interface (JNI), inoltre mediante l'uso degli standard JDBC (Java DataBase Connector) può interfacciarsi con i maggiori database relazionali. La combinazione delle tecnologie RMI/JNI e RMI/JDBC permette quindi di espandere sistemi legacy e chiusi con nuove componenti scritte in Java.

1.3.1 Principali vantaggi

RMI è essenzialmente il meccanismo di Java per eseguire Remote Procedure Call (RPC), tuttavia esso presenta diversi vantaggi rispetto ai sistemi RPC tradizionali. Il fatto di essere scritto in Java permette ad RMI di ereditare il paradigma object-oriented tipico di Java che lo rende flessibile e facilmente ingegnerizzabile. I sistemi tradizionali di RPC sono stati scritti con l'obiettivo di permettere a due applicativi, eventualmente scritti in linguaggi diversi, di poter comunicare fra loro. Se da un lato questo può risultare un vantaggio, d'altro canto una soluzione più mirata ad uno specifico linguaggio di programmazione come RMI permette di sfruttare a pieno tutte le potenzialità del linguaggio e sicuramente permette una gestione e una manutenzione più uniforme del sistema distribuito che si va a costruire.

Di seguito verrà fornita una panoramica dei principali vantaggi che si possono trarre dall'uso di RMI:

- *Object-oriented*: RMI permette di passare da un capo all'altro della comunicazioni fra applicativi oggetti interi come argomenti e valori di

ritorno, slegandosi quindi dalla limitazione di poter avere solo tipi di dato predefiniti. Questo significa che RMI permette di passare tipi complessi come argomenti: con altri sistemi RPC è possibile passare solo tipi primitivi ed è quindi necessario creare e manutenere una procedura di de-assemblaggio e ri-assemblaggio di oggetti complessi in oggetti primitivi e viceversa;

- *Movimento di logiche*: RMI permette di spostare il comportamento (definito nelle implementazioni di classi) da un end-point id comunicazione all'altro, è quindi possibile definire una certa logica di comportamento su un client e passarla ad un server che si occuperà di utilizzarla per le elaborazioni richieste dal client;

- *Design Patterns*: dal momento che RMI permette di spostare interi oggetti Java, il comportamento definito a livello progettuale dai Design Pattern tipici della programmazione a oggetti [25] viene mantenuto e la progettazione non deve essere modificata a causa del passaggio dei dati su rete;

- *Sicurezza*: RMI utilizza le funzionalità di sicurezza predefinite di Java che permettono di mantenere il sistema in sicurezza quando i client scaricano le implementazioni degli oggetti. Il security manager definito per l'applicazione viene utilizzato da RMI per proteggere il sistema da applets ostili e da codice potenzialmente ostile scaricato dalla rete. In condizioni critiche un server può essere configurato per non scaricare alcuna implementazione di codice;

- *Semplicità di scrittura e di utilizzo*: RMI rende semplice l'implementazione di server e client Java e l'implementazione delle politiche e metodi di comunicazioni fra i due end-point. Un server RMI richiede circa tre-quattro righe di codice per essere operativo e una interfaccia remota è semplicemente una interfaccia Java come le altre: questo permette di poter realizzare in poco tempo e con poche spese prototipi su cui poter valutare i progetti;

- *Connettività verso sistemi esistenti/legacy*: come già accennato Java RMI permette di connettere nuove strutture software ad applicativi esistenti legacy tramite l'integrazione con JNI: è sufficiente realizzare gli end-point di comunicazione utilizzando RMI e quindi sfruttare JNI per connettere quanto realizzato con il sistema legacy esistente. Allo stesso modo RMI può interagire con i database relazionali esistenti utilizzando JDNC senza modificare i sorgenti non-Java preesistenti che usano il database;

- *Scrivi una volta, esegui ovunque*: RMI segue il motto di Java "Write once, run anywhere": un sistema basato su RMI è eseguibile senza nessuna modifica su qualunque JVM e quindi su un gran numero di sistemi operativi;

- *Garbage Collector distribuito*: RMI usa un sistema di Garbage Collection distribuito per riciclare gli oggetti presenti su server remoti che non hanno più riferimenti da alcun client. Analogamente al garbage collector di una JVM, il garbage collector distribuito permette di definire gli oggetti sui server secondo necessità, sapendo che saranno rimossi (e quindi le risorse ad essi associate rilasciate) quando non avranno più bisogno di essere acceduti dai client;

- *Computazione Parallela*: RMI è multi-threaded e permette quindi ai server di sfruttare i thread Java per rispondere in modo concorrente alle richieste dei client;

- *Nessun Overhead*: RMI è parte del core di Java a partire dalla Java Development Kit (JDK) 1.1 e quindi esiste su ogni JVM a partire dalla versione 1.1. Tutti i sistemi RMI comunicano mediante lo stesso protocollo pubblico e tutti i sistemi Java comunicano fra loro direttamente senza nessun overhead dovuto a traduzioni di protocolli.

1.3.2 Principali svantaggi

A bilanciare i vantaggi descritti precedentemente devono essere presi attentamente in considerazione i seguenti svantaggi in cui si può incorrere realizzando un applicativo con RMI:

- *Performance*: parlando in termini di cicli di CPU e memoria allocata Java non è certo famoso per essere particolarmente performante. Il punto di forza principale di Java, cioè l'avere un unico codice compilato (Java Bytecode) che viene eseguito correttamente su un gran numero di JVM per sistemi diversi, è anche causa di grosse inefficienze. La JVM difatti si configura come una vera e propria Virtual Machine che viene eseguita su una macchina reale e quindi, come un ulteriore layer di traduzione fra l'applicativo e il processore, introduce un consistente overhead al momento dell'esecuzione del codice.

  Quindi se già un sistema scritto in Java può non eccellere in performance rispetto ad applicativi scritti in altri linguaggi di programmazione magari compilati e non interpretati, sicuramente con RMI, che è anch'esso un componente scritto in Java e parte integrante della JVM, la situazione non migliora;

- *Protocollo custom*: a differenza di soluzioni di comunicazione fra applicativi basati su webservices o comunque su protocolli HTTP un client applicativo che usi RMI disposto in reti aziendali particolarmente rigide in termini di sicurezza e di connettività può avere problemi nel passare i controlli di firewall e proxy verso l'RMI server (sopratutto se questo è disposto su una rete esterna). Non solo: vista la generale scarsa conoscenza di RMI da parte degli amministratori di sicurezza di reti aziendali e la loro naturale diffidenza verso tutto ciò che entra nella rete interna, il solo fatto di scaricare dall'esterno della rete aziendale codice che verrà eseguito da un elaboratore all'interno della rete porta a reazioni generalmente ostili verso questo protocollo o comunque a necessità di mediazione da gestire accuratamente.

- *Meccanismo sincrono*: le comunicazioni tra client e server tramite RMI sono strettamente sincrone: la chiamata di invio e ricezione di un oggetto con RMI è bloccante e pertanto può portare a situazioni di stallo che necessitano di una gestione spesso difficoltosa ed empirica;

1.3.3 Movimento di logiche: un esempio

Il punto forte di RMI che può risultare molto utile nella realizzazione di Iungo Collector è la possibilità di passare le logiche di elaborazione fra end-point diversi della comunicazione. Questa funzionalità permette di trattare client e server allo stesso modo: semplicemente come end-point di comunicazione che si passano in entrambe le direzioni dati e logiche computazionali.

Per meglio comprendere l'importanza di questa funzionalità verrà illustrato un semplice esempio che prevede il passaggio di logiche con RMI:

Supponiamo di dover implementare un server computazionale strutturato come mostrato in figura 1.9 il cui scopo principale è quello di eseguire computazioni arbitrarie richieste dai vari client (eseguite da lui perchè più efficiente o perchè l'unico a poter raggiungere risorse necessarie all'elaborazione) e restituire il risultato di tali elaborazioni.

Per realizzare tale sistema si procede a definire una interfaccia locale (cioè non remota) del tipo:

```
public interface Task{
Object run();
}
```

Quando il metodo run() è chiamato, vengono eseguite una serie di computazioni il cui risultato deve essere restituito ai client che hanno richiesto l'elaborazione. Questa interfaccia è volutamente generica per rappresentare un qualunque tipo di operazione che possa essere eseguita con la chiamata del metodo run(). L'interfaccia remota che permette l'esecuzione da remoto di tale metodo sarà definita come:

```
import java.rmi.*;
public interface ComputeServer extends Remote{
```

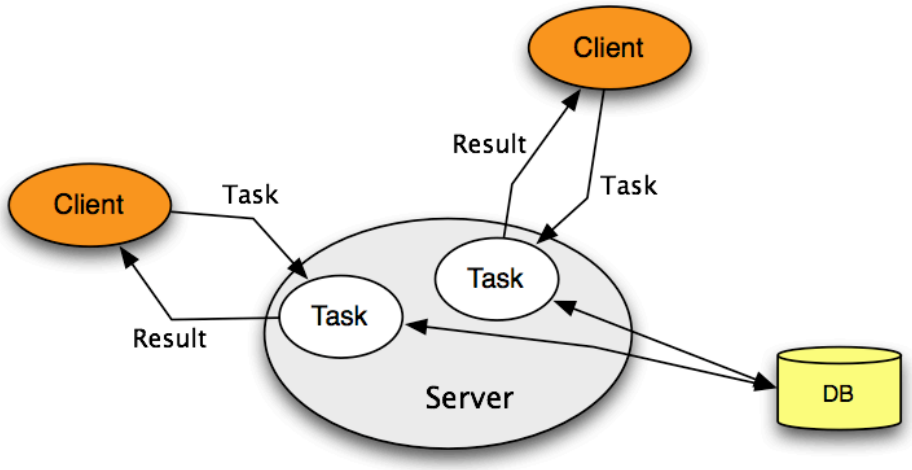

Figura 1.9: Esempio di server su RMI

```
Object compute(Task task) throws RemoteException;
}
```

L'unico scopo di una tale interfaccia remota è quello di permettere ad un client di creare un oggetto di tipo Task da trasmettere al server per l'esecuzione e restituire i risultati dell'elaborazione. Di seguito un esempio di come potrebbe essere implementato il server:

```
import java.rmi.*;
import java.rmi.server.*;
public class ComputeServerImpl
extends UnicastRemoteObject
implements ComputeServer
{
public ComputeServerImpl() throws RemoteException { }
public Object compute(Task task) {
return task.run();
}
public static void main(String[] args) throws Exception {
System.setSecurityManager(new RMISecurityManager());
```

```
ComputeServerImpl server = new ComputeServerImpl();
Naming.rebind("ComputeServer", server);
return;
}
}
```

Dando una occhiata al metodo "compute", si può notare che è anch'esso molto semplice: prende semplicemente l'oggetto passato come argomento dal client e restituisce il risultato della elaborazione eseguita dall'oggetto.

Il metodo "main" contiene le istruzioni minime essenziali per costruire un server RMI: viene dichiarato un SecurityManager che si occupa di gestire gli aspetti di sicurezza, viene istanziato un oggetto ComputeServer che viene associato ad un certo nome invocabile dai client per connettersi a lui.

Per quanto riguarda il lato client del nostro esempio supponiamo di voler chiedere al server di calcolare in modo efficiente le previsioni meteo secondo diversi modelli matematici. Per ciascuno di tali modelli è sufficiente:

- Creare una classe PrevisioneMeteo che implementi l'interfaccia "Task" sopra descritta, che abbia nel metodo compute tutti calcoli da eseguire per realizzare una previsione meteorologica e che ritorni un oggetto di tipo MappaMeteo contenente tutte le informazioni necessarie a definire la mappa della previsione meteo calcolata;

- Una volta scritta la classe PrevisioneMeteo il cliente dovrà invocare il metodo compute del ComputeServer passando come argomento l'oggetto PrevisioneMeteo del caso istanziato con i parametri del caso;

- Quando il sistema RMI di ComputeServerImpl viene eseguito riceve l'oggetto PrevisioneMeteo come un parametro in ingresso, ne scarica l'implementazione dal client e quindi invoca il metodo compute del server che prende tale oggetto PrevisioneMeteo come oggetto Task di argomento;

- L'oggetto Task (che è la PrevisioneMeteo passata) inizia l'esecuzione e qualora questa dovesse richiedere altre classi ad essa relativa, queste saranno scaricate nella loro implementazione secondo necessità;

- L'elaborazione viene eseguita sul compute server mentre il thread del cliente attende i risultati. Quando la computazione raggiunge la sua conclusione l'oggetto MappaMeteo viene passato come valore di ritorno dal server verso il client che provvede a scaricarne l'implementazione.

Con questo piccolo esempio abbiamo mostrato come RMI permetta di far eseguire su un end-point della comunicazione un pezzo di codice con le sue logiche ed i suoi dati specifici, ad un altro end-point di comunicazione in modo estremamente semplice e flessibile.

1.3.4 Architettura

Il sistema RMI è progettato per fornire un fondamento diretto e semplice per la costruzione di sistemi di elaborazione distribuiti object-oriented. In questa sezione vedremo quali sono gli aspetti architetturali della comunicazione tramite RMI.

Quando un server è esportato viene definito il suo tipo di riferimento: nell'esempio illustrato nella sezione precedente abbiamo ipotizzato un server di tipo UnicastRemoteObject che realizza un server point-to-point non replicato. In altri contesti avrebbe potuto essere opportuno definire il server come estensione di un MulticastRemoteObject che avrebbe presentato le strutture semantiche e sintattiche per un servizio multi-poit replicabile.

Quando un client riceve un riferimento ad un server, RMI scarica uno "stub" (pezzo di codice) che traduce le chiamate fatte a quel riferimento in chiamate verso il server remoto. Lo stub assembla gli argomenti del metodo utilizzando i meccanismi di serializzazione degli oggetti (l'oggetto passato come argomento deve quindi essere Serializable) ed invia le chiamate così organizzate via rete fino al server.

Dal lato del server viene ricevuta l'invocazione dal sistema RMI e viene relazionata ad uno "skeleton" (scheletro) che è procede alla de-serializzazione degli argomenti e alla chiamata dell'implementazione del metodo sul server. Quando l'esecuzione del metodo arriva a conclusione e viene restituito l'oggetto da mandare al client (oppure viene lanciata un eccezione) lo skeleton organizza e serializza il risultato e lo invia come risposta allo stub del client.

Lo stub del client quindi riceve il risultato, lo de-serializza e -in base a cosa ha restituito il server- resituisce il valore o lancia l'opportuna eccezione.

Questa architettura permette al riferimento di definire il comportamento delle comunicazione, ad esempio i che estendono l'UnicastRemoteObject vengono utilizzati per comunicare con un singolo server che è in ascolto su un particolare host e una specifica porta.

1.3.5 Sicurezza e Firewalls

Le criticità di sicurezza introdotte da RMI sono chiare e non trascurabili: di fatto si tratta di passaggio di codice da un host ad un altro e della sua esecuzione, se uno dei due end-point non è affidabile potrebbe trasmettere codice ostile e procurare danni al sistema.

Per prevenire questo tipo di eventualità la tecnologia RMI adotta una serie di misure atte ad evitare l'accesso del codice scaricato al resto del sistema: tutto ciò viene scaricato è eseguito all'interno di una "sandbox" e deve passare al vaglio di un SecurityManager personalizzabile. Queste due misure di sicurezza permettono di impedire l'accesso in scrittura al file system su cui il codice opera, controllando ogni richiesta di apertura di file e ogni richiesta di connessione.

Sul fronte della confidenzialità dei dati che vengono trasmessi su rete tramite RMI è possibile configurare i socket in modo che operino basandosi su SSL che provvede a fornire un meccanismo di criptazione sicuro.

Aldilà delle feature di RMI appena descritte è possibile che i componenti software che utilizzano RMI siano collocati all'interno di reti locali in cui

siano presenti precisi standard e protocolli di sicurezza per le comunicazioni fra host all'interno della rete stessa e verso server esterni alla rete. Una misura minima di sicurezza che è presente praticamente in tutte le situazioni di reti aziendali sicure sono i Firewall.

Un Firewall è un software che permette di stabilire una serie di regole che limitino il transito di dati entranti e uscenti una determinata rete basandosi sulla provenienza dei dati, sulla loro destinazione e sul loro contenuto.

RMI cerca di affrontare le limitazioni imposte dai Firewall mettendo a disposizione la possibilità di racchiudere i messaggi del protocollo RMI all'interno di messaggi inviati con protocolli diversi tipicamente considerati sicuri e non processati dai Firewall. Al momento della prima connessione di un client verso il riferimento ad un UnicastRemoteObject RMI tenta la trasmissione dei dati provando una di queste soluzioni alla volta:

1. Instaurazione di un canale diretto con la porta del server utilizzando i socket;

2. Se fallisce il tentativo (tipicamente il Firewall non riconosce il contenuto del messaggio come ammissibile al transito sulla sua rete) RMI incapsula il messaggio diretto allo skeleton del server all'interno del corpo di una richiesta HTTP POST che quindi viene inviato all'URL composto dal nome del server e dalla porta su cui è in ascolto RMI sul server;

3. Se anche questo tentativo dovesse fallire (ad esempio perchè la porta del server a cui ci si sta tentando di connettere è chiusa dal Firewall) il client RMI tenta di inviare il messaggio HTTP POST precedentemente costruito all'url composto dal nome del server e la porta 80 (che è la porta standard per il protocollo HTTP). Il server dovrà essere fornito di uno script CGI (fornito con l'implementazione di RMI) che si occupa di ricevere le chiamate di questo tipo sulla porta 80 e di passarle al server RMI in esecuzione sull'host.

La tecnica che porta ad un risultato positivo nella trasmissione dei dati verrà poi utilizzata per tute le future comunicazioni fra client e server. Nel caso in cui nessuno dei tre metodi risulti funzionante la chiamata remota fallisce.

1.4 Web Services

Una valida alternativa che ha senso considerare per l'implementazione della struttura di comunicazione di Iungo Collector è costituita dai Web Services (WS).

La crescita del Web sta avendo negli ultimi anni (fig 1.10) è la prova della efficacia che riesce ad avere l'utilizzo di protocolli semplici per le comunicazioni via Internet come base per un gran numero di servizi ed applicativi. In particolare il protocollo HTTP permette a dei semplici client (browser) di visualizzare pagine web ed altre risorse riferendosi ad essi tramite il loro URL (Uniform Resource Location).

Estendendo questa visione alle interazioni fra applicazioni, i WS offrono l'infrastruttura per creare una interoperabilità fra applicativi disposti su server diversi e non necessariamente creati dalle stesse persone. Tale interoperabilità è basata sullo scambio di messaggi XML (eXtended Markup Language) che vengono inviati dal client per richiedere al server l'esecuzione di un determinato servizio che mette a disposizione specificando i parametri del caso e il server utilizza a sua volta messaggi XML per restituire i risultati delle operazioni eseguite.

Il protocollo SOAP (Simple Object Access Protocol) [26] specifica le regole per l'utilizzo del XML per trasmettere le comunicazioni tra client e server al fine di ottenere un protocollo request-reply. SOAP è utilizzato per incapsulare i messaggi e trasmetterli via HTTP o altri protocolli (ad esempio TCP o SMTP).

In questo capitolo verrà fornita una visione dello stato dell'arte di questa tecnologia.

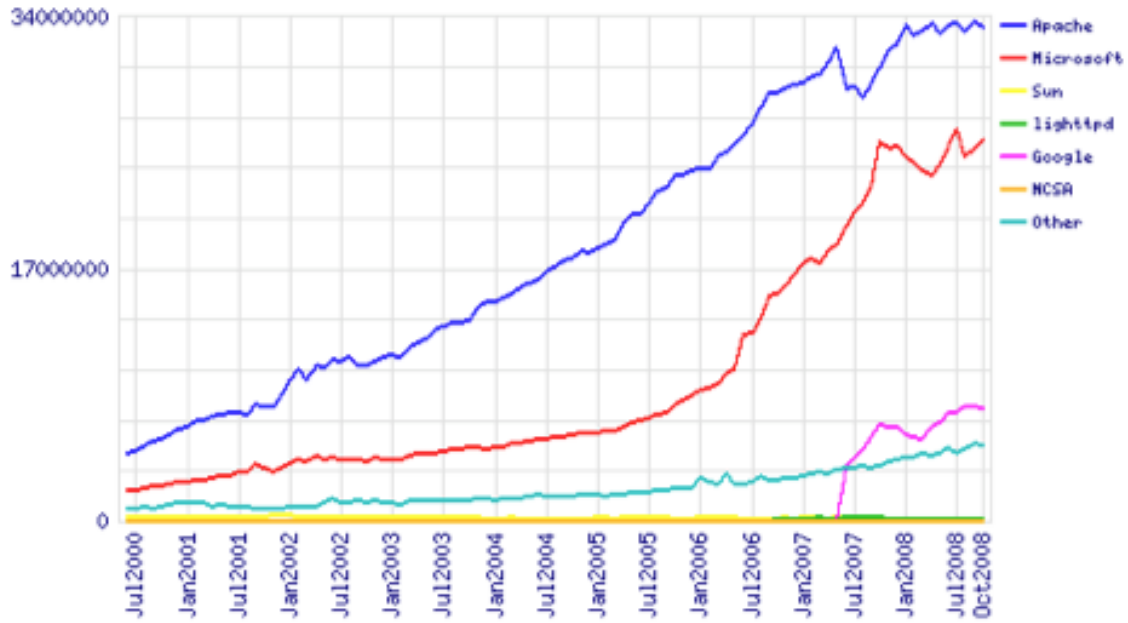

Figura 1.10: Andamento del numero di server web per tipologia

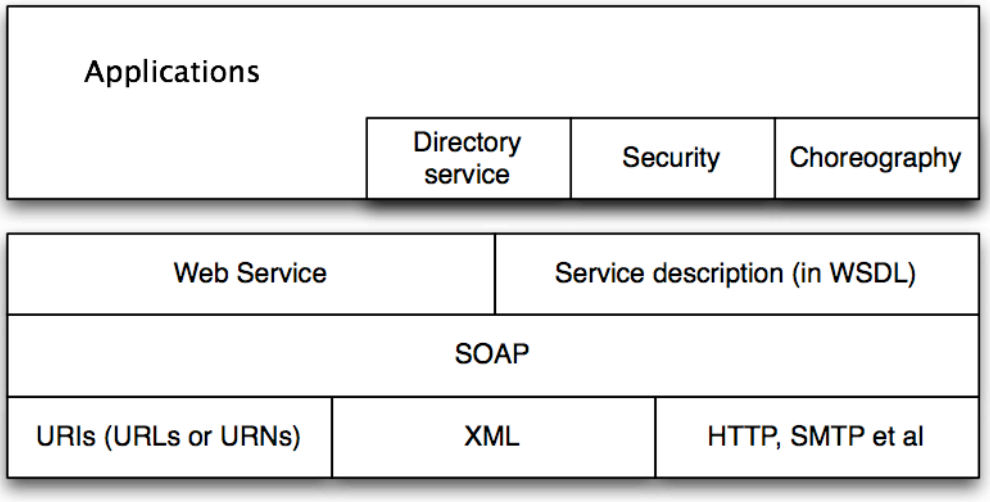

Figura 1.11: Gerarchia e struttura dei WebServices

Un web service è tipicamente una collezione di operazioni che possono essere eseguite da un client attraverso Internet. Ciascun servizio fornisce una sua specifica descrizione che riporta le informazioni necessarie all'utilizzo del servizio stesso. Tale descrizione è realizzata mediante WSDL (Web Services Description Language) che fornisce una descrizione dei tipi, messaggi, tipi di porte, operazioni e porte usate che è indipendente dal linguaggio di programmazione.

La maggior parte dei WebServices utilizza il protocollo SOAP per lo scambio di informazioni che prevede l'incapsulamento di messaggi XML. SOAP può essere utilizzato per realizzare un protocollo asincrono o anche un protocollo di tipo request-reply (basato però anch'esso sullo scambio di messaggi asincrono).

Esistono implementazioni del protocollo SOAP e di librerie per la manipolazione di XML praticamente per tutti i linguaggi di programmazione esistenti e la struttura stessa del WS non richiede che venga utilizzato lo stesso linguaggio di programmazione per il client e il server.

Originariamente SOAP era basato unicamente su HTTP, ma la versione attuale (1.2) è progettata per poter utilizzare un numero maggiore di protocolli di trasporto, come ad esempio SMTP, TCP o UDP. Questo garantisce una attinenza di SOAP ai protocolli più utilizzati su Internet e quindi lo rende un protocollo compatibile con tutte le infrastrutture di rete presenti. Grazie a queste caratteristiche risulta quindi possibile implementare tramite SOAP un meccanismo di RPC su HTTP/TCP/UDP/SMTP.. fra linguaggi differenti.

1.4.1 Principali vantaggi

In questa sezione verrà fornito un elenco dei principali vantaggi che un progetto software può trarre dall'utilizzare i WS basati su SOAP come meccanismo di comunicazione:

- *Indipendenza dai linguaggi*: grazie al fatto che SOAP utilizza XML per scambiare informazioni fra gli end-point della comunicazione, il pro-

tocollo risulta indipendente dai linguaggi di programmazione utilizzati per implementare il client e il server. Essendo una tecnologia molto diffusa che utilizza protocolli standard su cui esiste una vasta letteratura, esistono implementazioni di librerie per l'utilizzo di SOAP e la manipolazione di XML praticamente per ogni linguaggio di programmazione. Per la tesi è di interesse particolare l'integrazione con Java per la quale si segnala l'esistenza di JWSDP (Java Web Services Developer Pack) che fornisce un set di API relative agli XML e ai WS di cui la più importante è JAX-RPC (Java Api for XML-based Remote Procedure Call);

- *Standard*: SOAP è una W3C Recommendation ed è quindi integrato con tecnologie standardizzate e molto diffuse: è quindi possibile affermare che ogni cambiamento della sua struttura sarà approfonditamente valutato dai gruppi di lavoro W3C e avverrà in modo omogeneo rispetto agli avanzamenti delle altre tecnologie su cui si basa;

- *Leggibilità dei messaggi*: dal momento che le comunicazioni di SOAP avvengono tramite XML, e dato che XML è un linguaggio perfettamente leggibile dall'uomo ne risulta che i messaggi scambiati (a meno di meccanismi di cifratura) risultano leggibili e facilmente debuggabili;

- *Asincronia*: dato che la struttura di SOAP originariamente era concepita per basarsi unicamente su HTTP è garantito il comportamento asincrono che permette di evitare situazioni stallo in attesa di risposta da un end-point della comunicazione;

- *Sicurezza*: sono definiti dal W3C standard per garantire segretezza, autenticità e integrità di messaggi XML [17] come la possibilità di utilizzare firme digitali basate su certificati X.509, SPKI o chiavi PGP da applicare alla totalità dei messaggi o anche a solo a parti di questi. Questo implica che SOAP eredita dall'XML la possibilità di garantire segretezza, autenticità e integrità dei messaggi scambiati tra end-point della comunicazione;

- *Basso rischio*: le comunicazioni SOAP nella maggior parte dei casi sono considerate dai Firewall "a basso rischio" o comunque "da non bloccare" sia per quanto riguarda la loro destinazione (porta 80 se usa HTTP piuttosto che porta 25 se usa SMTP) che per il loro contenuto (non codice eseguibile potenzialmente ostile ma semplicemente chiamate a procedure di software presente sul server e relativi parametri di esecuzione).

1.4.2 Principali svantaggi

In questa sezione verrà fornito un elenco delle criticità che devono essere considerate e gestite da chi voglia utilizzare WS basati su SOAP come meccanismo di comunicazione in un progetto software:

- *Performance*: come risulta dallo studio in [18] le comunicazioni via WebServices risultano essere meno preformanti rispetto a comunicazioni analoghe eseguite con RMI sia in termini di tempo che in termini di overhead introdotto;

- *Comunicazioni limitate ai dati*: il limite maggiore di SOAP (sopratutto se paragonato a RMI) è quello di realizzare un meccanismo RPC molto basilare in cui tra un end-point di comunicazione e l'altro possano transitare unicamente chiamate di funzioni e parametri: risulta impossibile passare logiche di esecuzione e strutture complesse;

- *Implementazione laboriosa*: se da un lato l'indipendenza dai linguaggi di programmazione può essere visto come un vantaggio (come specificato in sezione 1.4.1) d'altro canto introduce una complessità dal punto di vista implementativo dovuta al fatto che si deve realizzare qualcosa di poco specializzato e decisamente general-purpose;

- *Propagazione manuale delle modifiche*: SOAP non fornisce alcun meccanismo per la propagazione automatica delle eventuali modifiche all'

interfaccia del WS piuttosto che cambiamenti di URL. Ad ogni modifica del server che coinvolga l'interfaccia del WS (e con modifica si intende anche estensione di funzionalità per far fronte a nuove esigenze) è necessario procede all'aggiornamento manuale di tutti i client;

- *Mancanza di trasparenza*: SOAP non fornisce alcun meccanismo di trasparenza rispetto alle operazioni di marshalling/unmarshalling dei dati e non presenta caratteristiche di trasparenza di posizione. Lo sviluppatore deve quindi manutenere manualmente le procedure di marshalling/unmarshalling dei dati da lui definiti introducendo un costo significativo di gestione dell'applicativo.

1.4.3 Sicurezza e Firewall

Come già accennato nelle sezioni precedenti SOAP garantisce un buon livello di sicurezza: non trasporta codice eseguibile e quindi rischi che un WS venga utilizzato per portare un attacco ad un server sono relativamente bassi.

Per questo motivo e per il fatto che i messaggi SOAP sono incapsulati in protocolli standard, largamente utilizzate e che utilizzano porte di sistema generalmente "aperte", SOAP è ritenuto particolarmente indicato per le situazioni in cui sia presente la necessità di mettere in comunicazione applicativi appartenenti a sotto-reti diverse su cui possano operare sistemi di sicurezza come Firewall con regole anche particolarmente restrittive.

Capitolo 2

Studio di fattibilità

In questo capitolo verranno inizialmente riportati i requisiti di Iungo Collector così come sono emersi emersi da una intervista con il Product Engineer di Smarten s.r.l., da tale elicitazione iniziale verrà ricavata descrizione formale dei requisiti funzionali e non funzionali, quindi verranno messe a confronto le tecnologie analizzate nel capitolo precedente e verranno descritte le motivazioni che hanno portato alla scelta della tecnologia ottimale per l'implementazione del dimostratore. Verranno quindi analizzati i principali casi d'uso di Iungo Collector ed infine verranno descritti i vari aspetti che hanno riguardato lo sviluppo vero e proprio del dimostratore di funzionalità minime e le modalità ed i risultati dei testi di performance su esso realizzati.

2.1 Elicitazione informale dei requisiti

In seguito ad una intervista con il Product Engineer di Smarten s.r.l. è emerso quello che può essere uno schema di massima del funzionamento di Iungo Collector (fig 2.1):

1. Il fornitore F1, tramite un'interfaccia grafica semplice ed intuitiva, richiede i dati aggiornati che lo riguardano e Iungo Collector si preoccupa di eseguire le operazioni di recupero

2. Iungo Collector interroga quindi tutte le installazioni dei Clienti con dati riguardanti il fornitore F1

3. Le installazioni di Iungo presso i clienti recuperano le informazioni richieste dai loro databases locali e creano una struttura dati che include, oltre ai dati richiesti, le informazioni sulle elaborazioni possibili secondo la logica configurata nella loro installazione (il workflow) e restituiscono il risultato allo Iungo Collector richiedente;

4. Iungo Collector elabora i dati ricevuti e li restituisce all'interfaccia grafica del Fornitore richiedente.

5. Se il fornitore esegue operazioni sui dati ricevuti, l'interfaccia grafica segnala le operazioni a Iungo Collector che a sua volta le invia alle installazioni Iungo coinvolte.

Punto critico di questa struttura è il meccanismo di comunicazione fra lo Iungo Collector, le varie installazioni di Iungo e i moduli di presentazione dei dati utilizzati dai fornitori.

Iungo Collector deve quindi avere le seguenti caratteristiche:

1. Deve consentire la registrazione delle installazioni di Iungo in modo che il Collector abbia una visione aggiornata di quali siano raggiungibili e di quale sia il loro stato;

2. Deve essere in grado di richiedere le informazioni mantenute nei database locali delle installazioni di Iungo;

3. Deve consentire l'invio verso le installazioni di Iungo delle operazioni eseguite dal Fornitore attraverso la sua interfaccia web;

4. Deve adottare meccanismi di sicurezza che garantiscano fra tutti gli end-point della comunicazione:

    - i principi di segretezza e autenticazione nello scambio di informazioni

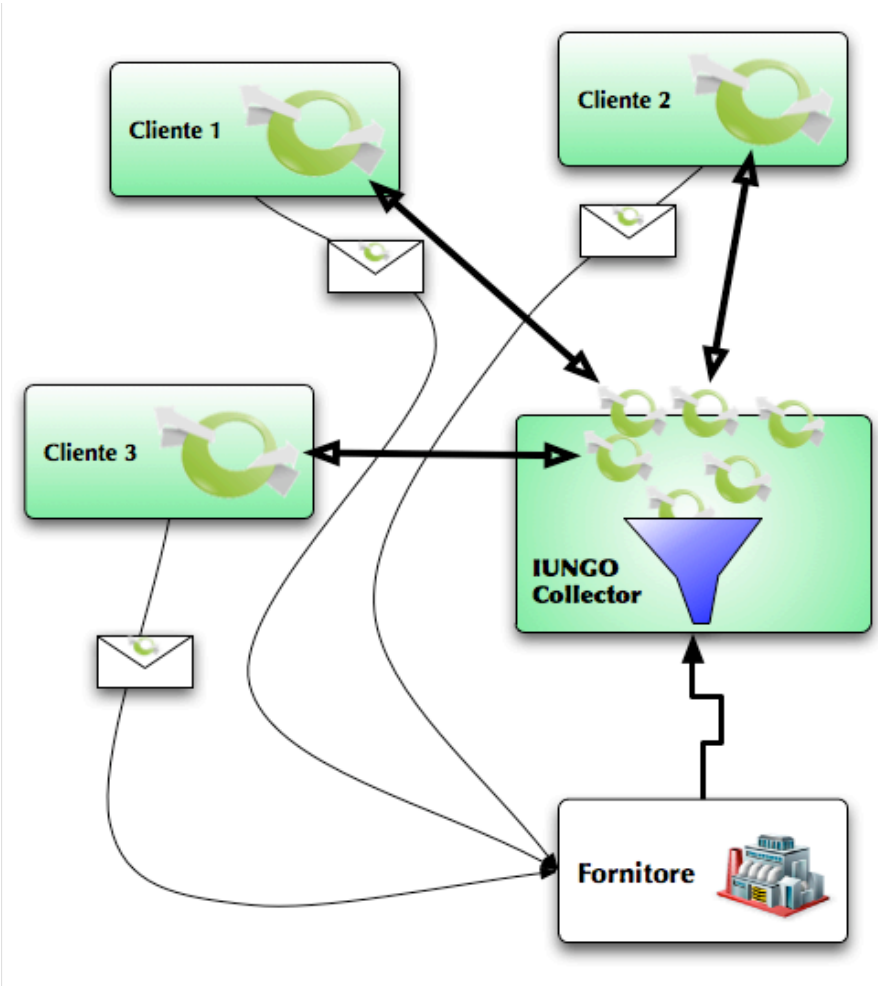

Figura 2.1: Iungo Collector nelle comunicazioni fra diversi Iungo e il fornitore

- l'integrità delle informazioni scambiate
- protezione dalla perdita di informazioni in caso di malfunzionamenti/guasti del sistema

5. Deve essere replicabile, scalabile e capace di bilanciare il carico delle richieste su canali differenti per far fronte alla mole di dati oggetto della comunicazione.

In figura 2.1 così come in tutte le altre figure riguardanti la struttura di Iungo Collector e in tutte le parti descrittive della sua struttura, il componente principale è rappresentato e descritto come un unico nodo centralizzato. Questa rappresentazione tuttavia non deve far pensare che tale componente centralizzato sia un "Single Point of Failure": come enunciato nell'ultimo punto dell'elenco precedente Iungo Collector dovrà essere replicato per garantire un buon livello di fault tolerancy. Dal momento che il nodo centrale non deve mantenere in locale i dati inseriti e modificati dai suoi utenti, ma deve bensì occuparsi di trasferirli ai nodi periferici opportuni, si può suppore che esso non debba necessariamente avere un database locale e che quindi possa essere visto come una componente stateless con una cache temporanea. Pertanto il meccanismo di replicazione può essere interamente delegato alla piattaforma di Cloud Computing su cui il nodo centrale verra' posizionato ed è sufficiente che venga implementato un meccanismo di replicazione passiva[1].

Date queste premesse in tutti i futuri riferimenti alla struttura centralizzata di Iungo Collector su Cloud Computing si tenga a mente che in realtà si sta considerando un insieme di repliche opportunamente gestite dal Infrastructure Provider.

2.2 Formalizzazione dei requisiti

In questa sezione verranno illustrati i requisiti funzionali e i requisiti non funzionali del software da realizzare. I requisiti funzionali descrivono ciò che

il sistema deve fare specificando quali funzioni il sistema deve fornire per soddisfare le richieste definite in sezione 2.1, similmente i requisiti non funzionali descrivono le caratteristiche di servizio che non coinvolgono direttamente funzionalità offerte dal sistema.

2.2.1 Requisiti funzionali

RF01: Il sistema deve permettere ad un numero variabile di client di autenticarsi ad esso per lo scambio di dati.

RF02: Il sistema deve poter inviare richieste di dati ai singoli client selezionati secondo determinati criteri basati sulle caratteristiche dei client stessi.

RF03: Il sistema deve poter ricever i dati inviati dai client assieme alle operazioni logiche da applicare su di essi in caso di inserimento di nuovi dati, modifica o cancellazione dei dati esistenti.

RF04: Il sistema deve poter provvedere ad una rappresentazione grafica dei dati che ne renda trasparente la provenienza e l'applicazione delle logiche ricevute dai client sorgenti.

RF05: Il sistema deve poter restituire i dati eventualmente variati ai diversi client.

2.2.2 Requisiti non funzionali

RNF00: Il sistema deve essere disponibile su un paradigma di Cloud Computing.

RNF01: Il sistema deve risultare efficiente (in termini di tempi di risposta e dimensione dei dati trasmessi su rete) nel recupero, analisi, presentazione e ritorno dei dati da lui gestiti.

RNF02: Il sistema deve risultare raggiungibile da client disposti all'interno di reti LAN protette da firewall arbitrariamente restrittivi.

RNF03: Il sistema deve garantire un trattamento sicuro dei dati in termini di integrità, segretezza e autenticità.

RNF04: Il sistema deve risultare facilmente estendibile e modificabile.

RNF05: Capacità di sopportare picchi d'utilizzo e di far fronte ad una forte crescita del numero di utenti partendo da un bacino iniziale ipotetico nell'ordine di grandezza di 10.000 unità.

RFN06: Disponibilità 24/7 worldwide.

2.3 Contesto di Trust

Come accennato durante la descrizione dello stato dell'arte (sez 1.2) il paradigma di Cloud Computing introduce una classe di problematiche riguardanti il concetto di "Trust" che difficilmente vengono incluse nei requisiti funzionali ma che allo stato dei fatti possono rappresentare una criticità se non gestite accuratamente.

2.3.1 Trust in IungoCollector

Per prima cosa è necessario dare una definizione di cosa si intende per Trust: come descritto in [19]

<<Trust is defined as the belief the trusting agent has in the trusted agent's willingness and capability to deliver a mutually agreed service in a given context and in a given time slot. >>

(Il Trust è definito come la convinzione che un soggetto ha nel fatto che un altro soggetto abbia la volontà e la capacità di fornire un servizio in un determinato contesto e in un determinato lasso temporale sulla base di un accordo reciproco)

In un contesto di Cloud Computing questo concetto ha valenza centrale in quanto il numero di attori è solitamente superiore ai due e il servizio offerto da ciascuno di loro è essenziale per il business degli altri.

Nel caso specifico di IungoCollector il problema di Trust è sicuramente non trascurabile (fig 2.2): non solo è necessario trovare un infrastructure provider "Trustworthy" per il collocamento del nodo centrale del sistema, ma è necessario riuscire a trasmettere Trust sia ai Fornitori che accedono

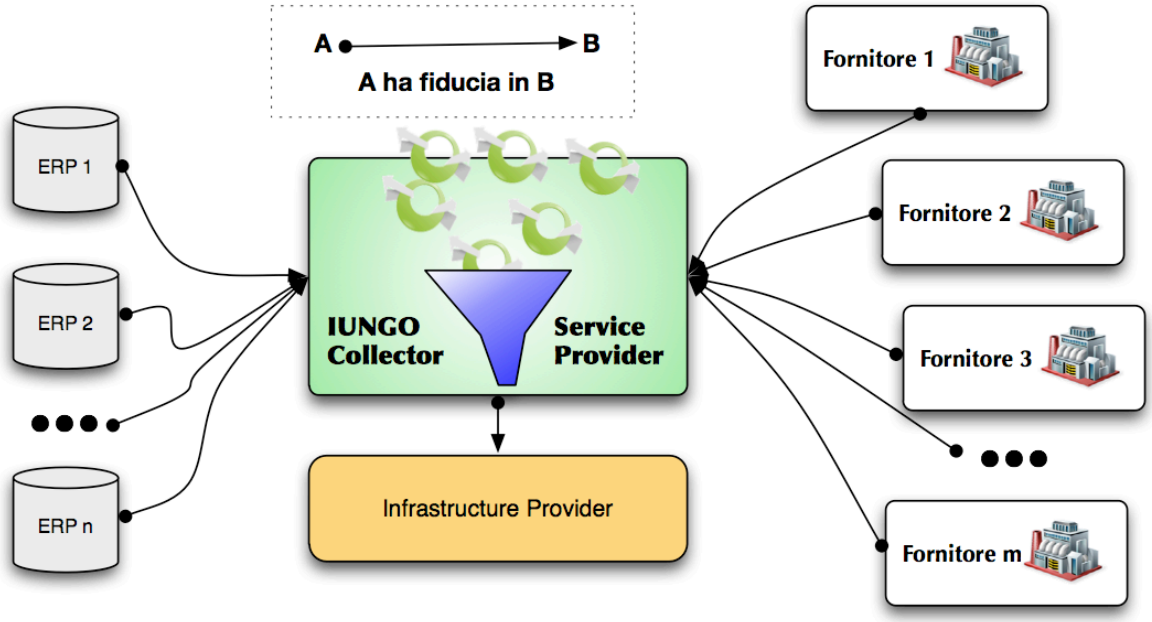

Figura 2.2: Relazioni di Trust in Iungo Collector

all'interfaccia web di IungoCollector sia sopratutto ai Clienti che devono consentire a IungoCollector di recuperare da remoto i dati dal loro Sistema ERP.

Risulta quindi necessario curare la reputazione di IungoCollector in termini di sicurezza ed affidabilità:

- IungoCollector deve offrire la possibilità che tutte le comunicazioni in entrata e in uscita dai suoi nodi siano criptate;

- IungoCollector deve appoggiarsi su un infrastructure provider affidabile e dalla buona reputazione e deve possibilmente rendere noto ai suoi utenti l'identita del provider per poter trarre indirettamente vantaggio dalla sua reputazione.

2.4 Scelte tecnologiche

In seguito alla elicitazione dei requisiti funzionali e non funzionali è possibile effettuare le scelte tecnologiche che caratterizzeranno la realizzazione del dimostratore di fattiblità e di Iungo Collector stesso.

2.4.1 Quale Cloud?

La prima scelta da affrontare riguarda il Cloud Computing. Il fatto di utilizzare o meno il paradigma di Cloud Computing non è di per se oggetto di discussione di questa tesi e viene dato per assodato, tuttavia dallo studio dello stato dell'arte di tale tecnologia descritto in sezione 1.2 risulta evidente che siano necessarie delle scelte riguardanti la tipologie di Cloud di cui servirsi.

Cloud	**Pubbliche**	**Private**	**Ibride**
Performance	Paragonabile ad un normale servizio di hosting, eccetto per la capacità di gestire i picchi di utenza	Legate alla quantità e qualità di risorse acquistate e alle capacità dei sistemisti che le amministrano	Paragonabile ad un normale servizio di hosting, eccetto per la capacità di gestire i picchi di utenza
Sicurezza	Il livello di sicurezza è totalmente affidato al provider che e quindi assolutamente variabile in base al SLA con lui concordato	Il livello di sicurezza del sistema software è arbitrariamente alto, tuttavia per ottenere una protezione da eventi catastrofici naturali e politici è necessario raddoppiare la struttura e de-localizzarla opportunamente	Tipicamente buona dal momento che le comunicazioni viaggiano su VPN. Per il resto legata allo SLA concordato con il provider
Flessibilità	La flessibilità è limitata alla configurabilità del sistema fornito dal provider	Flessibilità assoluta in base alle esigenze e alle disponibilità economiche	Flessibilità limitata alla configurabilità del sistema fornito dal provider. Solitamente la Cloud può essere configurata anche nella topologia di rete.

Costi	Costi iniziali nulli, costi di gestione generalmente molto contenuti e pay-per-use, nessun costo di formazione dello staff	Elevati costi iniziali, costi di manutenzione e gestione non trascurabili, costi necessari per la formazione tecnica dello staff o per l'acquisizione di figure professionali già formate	Costi iniziali nulla, costi di gestione generalmente molto contenuti (ma superiori alle Cloud Pubbliche) e pay-per-use. Consigliabile una formazione dello staff che amministrerà la rete virtuale.

Tabella 2.1: Confronto tra i diversi tipi di Cloud

Nella tabella 2.1 si sono riassunte le caratteristiche principali delle tre tipologie di Cloud più diffuse e descritte nei capitoli precedenti.

Per quanto riguarda Iungo Collector è necessario tenere presente di alcune considerazioni generali che riguardano l'applicativo e la realtà aziendale in cui si va contestualizzare:

- **Novita'** nonostante Iungo abbia una storia rispettabile e sia ormai giunto ad un livello di industrializzazione che ne definisce formalmente le strutture dati e le funzionalità, Iungo Collector si presenta come un applicativo totalmente nuovo e senza particolari vincoli strutturali "legacy" da rispettare;

- **High Availability** Iungo Collector è nelle sue intenzioni uno strumento di lavoro indispensabile per l'ufficio vendite e per la struttura produttiva delle realtà aziendali con cui va a interagire: è indispensabile garantire una raggiungibilità 24/7 worldwide (*RNF06*);

- **Risorse Umane** come accennato nella Introduzione di queste tesi, Smarten s.r.l. è una realtà imprenditoriale molto dinamica e piccola: conta una decina di dipendenti di cui cinque sviluppatori software con

un profilo formativo di stampo ingegneristico e orientato alla programmazione allocati interamente al project management dei progetti Iungo dei clienti di Smarten s.r.l. e alla industrializzazione e sviluppo del core di Iungo;

- **Rischio Imprenditoriale** viste le piccole dimensioni della realtà di Smarten s.r.l. risulta indispensabile ridurre al minimo il rischio imprenditoriale di nuovi progetti come Iungo Collector: un basso (o nullo) investimento iniziale è condizione indispensabile per l'avvio di un progetto.

Da queste considerazioni, dai requisiti elicitati in precedenza e da quanto anticipato nella introduzione di questa tesi a proposito della struttura e delle funzionalità di Iungo Collector possiamo dedurre che:

- Non ci sono vincoli di compatibilità verso il passato, non è necessaria nessuna virtualizzazione di particolari topologie di rete. I vantaggi offerti da una struttura Cloud Ibrida pertanto risultano non utili all'ottenimento di quanto desiderato per Iungo Collector;

- Se si volesse far fronte alla richiesta di High Availability realizzando una Cloud Privata, l'investimento iniziale sarebbe quasi sicuramente proibitivo relativamente alle problematiche di rischio imprenditoriale per la realtà di Smarten s.r.l. sopra descritte;

- Le risorse-uomo di Smarten s.r.l. non sono in numero sufficiente o non esistono le condizioni di carico di lavoro tali da permettere un percorso formativo di tipo sistemistico ad uno staff minimo e un allocamento di una quantità costante di risorse-uomo alla gestione di una Cloud Privata o Ibrida;

- Non necessitando di risorse hardware particolari o di sistemi software esotici su cui basarsi per una buona esecuzione di Iungo Collector è sufficiente disporre un sistema operativo comune per server (distribuzione

Linux, Mac Os X Server o Windows Server) configurabile e gestibile tramite accesso remoto.

Pertanto risulta evidente che il miglior compromesso capace di soddisfare la maggior parte delle tematiche affrontate in questa sezione risulta essere una soluzione di Cloud Pubblica. Fra le Cloud Pubbliche è consigliabile orientare la scelta verso le soluzioni che mettono a disposizione un sistema operativo virtualizzato su cui poter operare con maggiore controllo e libertà rispetto ad opzione di semplice hosting dell'applicativo in modo da poter far fronte alle necessità di espansione e gestione dinamica del sistema.

L'unico aspetto rilevante che risulta penalizzato nella scelta di una Cloud Pubblica è l'aspetto della Sicurezza dei dati gestiti: Iungo Collector si dovrà occupare della gestione di dati sensibili e strategici per gli utenti del sistema e pertanto la riservatezza e l'autenticità di tali dati sono caratteristiche critiche per la buona funzionalità del sistema. Dato che Cloud Private o Ibride avrebbero permesso una maggiore libertà nella definizione dei criteri di sicurezza dei software in esse eseguiti, è necessario aggiungere un ulteriore requisito non funzionale che permetta di far fronte a questa mancanza:

RNF03-b: la gestione delle politiche di riservatezza e autenticità dei dati trattati da Iungo Collector è demandata alla implementazione software del sistema (in particolare alle tecnologie e ai meccanismi di comunicazioni fra i vari end-point del sistema) piuttosto che alla piattaforma sistemistica su cui verrà installato.

2.4.2 RMI o SOAP?

Una ulteriore scelta sul piano tecnologico riguarda la tecnologia da utilizzare per l'infrastruttura di comunicazione fra gli end-point di Iungo Collector.

Nelle sezioni 1.3 e 1.4 è stato illustrato lo stato dell'arte delle due maggiori tecnologie che potrebbero più di altre fare al caso di Iungo Collector: Java RMI e Webservices su SOAP.

Vediamo come si posizionano le due tecnologie rispetto ai requisiti funzionali e non funzionali illustrati in sezione 2.2:

- *RF01*: sia Java RMI che SOAP permettono di realizzare un meccanismo di autenticazione arbitrariamente complesso: RMI può richiedere una autenticazione mediante la pubblicazione di un metodo remoto che richieda determinate credenziali e -in caso di autenticazione positiva- inserire il riferimento al client all'interno di una lista di client autenticati da cui accettare richieste di elaborazione; similmente un Webservice basato su SOAP può pubblicare una funzione di autenticazione che richieda determinate credenziali e -in caso di autenticazione positiva- può associare l'avvenuta autenticazione alla Sessione HTTP del client che la ha effettuata;

- *RF02*: RMI è appositamente strutturato per la creazione di un protocollo di comunicazione bidirezionale in cui i ruoli di client e server sono relativi e possono essere interpretati da entrambi gli end-point di comunicazione secondo la necessità del momento. Al contrario in una struttura di comunicazione basata su Webservices i ruoli di client e di server sono rigidamente definiti: l'end-point che pubblica un Webservice è il server che riceve richieste di elaborazione da parte del client. Nel caso di Iungo Collector è quindi necessario creare una struttura basata su due Webservice: uno sul sistema centrale che pubblichi -ad esempio- le funzionalità per l'autenticazione e uno sui vari nodi periferici che pubblichino le funzionalità per il recupero dei dati;

- *RF03*: anche in questo caso RMI è appositamente strutturato per prevedere il passaggio di logiche come il passaggio di dati tra due end-point della comunicazione. Con i Webservices non è possibile effettuare un passaggio di logiche a meno di ingegnerizzare ed implementare un meccanismo che trasferisca come vettore di byte il codice contenente la logica da caricare ed eseguire a run-time;

- *RF04*: sia RMI che Webservices possono realizzare una interfaccia grafica che renda trasparente all'utente finale la provenienza dei dati.

La trasparenza delle operazioni logiche da eseguire sui tali dati risente dei limiti dei Webservices accennati al punto precedente;

- *RF05*: come per il punto RF02 con RMI risulta più semplice invertire il verso della comunicazione per restituire i risultati di una operazione, con i Webservices è necessario che entrambi gli end-point di comunicazione pubblichino le funzionalità necessarie per lo scambio di dati bidirezionale;

- *RNF00*: non esistono particolari controindicazioni nè per l'utilizzo di RMI nè per l'utilizzo di Webservices su un paradigma di Cloud Computing;

- *RNF01*: l'efficienza del sistema nel suo complesso intesa come tempi di risposta e dimensione dei dati trasmessi su rete è in gran parte dipendente dallo studio del miglior algoritmo per il meccanismo di comunicazione fra gli end-point. Tuttavia dovendo valutare questo aspetto dal punto di vista tecnologico possiamo assumere di essere riusciti a realizzare l'algoritmo più efficiente che renda minimo il numero di messaggi scambiati su rete e fare riferimento allo studio realizzato in [18] di cui riportiamo i risultati nella tabella e nelle immagini seguenti.

  In queste immagini sono messi a confronto i tempi medi di istanziamento e di trasmissione di alcuni dati primitivi e di un oggetto Java di tipo String utilizzando RMI puro, RMI con tunnel HTTP sulla porta 80, RMI con tunnel HTTP verso una servlet (si veda la sezione 1.3) e Webservices su SOAP. Da questi risultati su può dedurre che la soluzione con RMI puro è la più efficiente con una differenza di almeno un ordine di grandezza rispetto alle altre con cui è stato messo a confronto. I Webservices risultano essere i secondi più efficienti con una distacco da RMI puro nell'ordine di 14 volte più lenti nella trasmissione di dati su rete. Le soluzioni di tunnelling HTTP che RMI mette a disposizione per aggirare gli eventuali problemi di firewall rimangono comunque le più inefficienti con risultati che vanno dalle 30 alle 40 volte più lente

Time in ms	RMI	HTTP-to-port	HTTP-to-servlet	Web services
Instantiation	0,686	19,070	19,483	0,616
Simple types average:	0,157	5,052	7,663	2,336
- int	0,157	5,044	7,659	2,330
- short	0,156	5,041	7,689	2,356
- long	0,156	5,050	7,622	2,319
- float	0,157	5,052	7,680	2,375
- double	0,161	5,066	7,689	2,342
- boolean	0,155	5,059	7,670	2,316
-byte	0,155	5,050	7,631	2,313
String	0,172	5,105	7,658	2,353

Table 3: Local performance results

Figure 3. Performance results without network overhead

Figure 4: Performance results with network overhead

Figura 2.3: Risultati dei test di performance su RMI e WebServices [18]

rispetto ad un RMI puro e dalle 2 alle 3 volte più lente rispetto ai Webservices;

- *RNF02*: Come accennato nel capitolo 1.4 il fatto che i Webservices possano utilizzare in modo nativo il protocollo HTTP e utilizzi messaggi XML permette di rendere meno soggetto all'intervento di firewall che ne possano bloccare o rallentare il funzionamento. Java RMI mette a disposizione soluzioni che permettono di ricondurre le comunicazioni ad un protocollo HTTP che incapsula messaggi RMI riuscendo nella maggior parte dei casi a superare firewall o restrizioni sull'utilizzo di porte esotiche, tuttavia -come mostrato nel punto precedente- tali soluzioni deteriorano le performance del sistema in modo molto significativo;

- *RNF03* e *RNF03-b*: sia Java RMI che i Webservices permettono di rendere sicuro il canale di comunicazioni basandosi su SSL e quindi garantiscono i criteri di segretezza, integrità e autenticità dei messaggi indispensabili per la garanzia di sicurezza che Iungo Collector deve fornire ai suoi utenti;

- *RNF04*: considerando il fatto che Java RMI eredita il paradigma di programmazione a oggetti tipico di Java che lo rende estremamente flessibile ed estensibile, il fatto che RMI gestisce nativamente il marshalling e l'un-marshalling dei dati e delle logiche da inviare su rete senza che lo sviluppatore debba intervenire in alcun modo ed il fatto che RMI mette a disposizione un meccanismo per il download automatico di classi aggiornate, mentre i Webservices non solo non offrono nessuna di queste funzionalità ma risultano anche molto rigidi nella loro implementazione, si può affermare con sufficiente certezza che un sistema Java basato su RMI risulta più flessibile e più facilmente espandibile e modificabile rispetto ad un suo omologo basato su Webservices;

- *RNF05*: sia RMI che Webservices non soffrono di particolari problemi riguardo al carico di lavoro e al numero di utenti. Ad esempio con en-

trambe le soluzioni è possibile approntare un approccio multi-threaded che lavori in parallelo minimizzando i tempi di risposta;

- *RFN06*: nè RMI nè i Webservices introducono problemi o limiti alla realizzazione di un sistema che sia disponibile 24/7 worldwide.

Per meglio illustrare i pesi che i diversi fattori hanno avuto nella scelta della tecnologia migliore tra RMI e SOAP, nella tabella seguente viene assegnato un coefficiente di peso (compreso fra 0 e 1 dove 1 è il massimo del peso) ad ogni requisito funzionale in base alla sua rilevanza nel contesto di Iungo Collector e un valore di soddisfazione (compreso fra 0 e 10 dove 10 è il massimo della soddisfazione e 6 è il minimo di sufficienza) per ciascuna delle due tecnologie rispetto a tale requisito.

Requisito	Peso	RMI	SOAP
RF01	*0,2*	6	6
RF02	*0,7*	8	6
RF03	*1,0*	10	5
RF04	*0,7*	7	6
RF05	*0,7*	8	6
RNF00	*1,0*	6	6
RNF01	*0,8*	6	6
RNF02	*0,8*	6	8
RNF03 e 03-b	*0,5*	6	6
RNF04	*1,0*	10	4
RNF05	*0,3*	6	6
RNF06	*0,3*	6	6
SOMMA PESATA		***59,5***	***46,6***

Tabella 2.2: Valutazione della soddisfazione dei requisiti da RMI e SOAP

Il calcolo illustrato in tabella 2.2 che rappresenta la somma pesata dei vari giudizi di soddisfazione rispetto ai singoli requisiti, e il fatto che RMI a dispetto di SOAP riesce a soddisfare con una valutazione almeno sufficiente (in alcuni casi ottima) tutti i requisti di Iungo Collector indicano senza ombra di dubbio che Java RMI è la tecnologia migliore in questo contesto e pertanto sarà quella utilizzata nella realizzazione del dimostratore di fattibilità e in Iungo Collector.

2.5 Casi d'uso

Questa sezione descrive l'analisi dei casi d'uso del sistema che verrà in seguito utilizzata per realizzare la struttura e il sistema stesso di Iungo Collector. In sezione 2.6.1 verranno scelti i casi d'uso di maggiore interesse e sulla base di questi verrà realizzata la progettazione e l'implementazione del dimostratore di fattibilità.

I casi d'uso sono una tecnica per descrivere i requisiti funzionali di un sistema basandosi sulle interazioni tipiche fra gli utenti di un sistema (gli attori) ed il sistema stesso. Ogni caso d'uso si concentra sulla descrizione dei singoli passi che devono essere compiuti dagli attori e dal sistema per permettere all'attore che ha iniziato la sequenza di portare a termine una precisa operazione.

Una sola descrizione grafica (allo stesso modo di una sola descrizione testuale) non permetterebbe di avere una visione chiara e univoca del processo completo con tutte le alternative possibili, quindi si è deciso di descrivere ogni caso d'uso tramite l'ausilio di una descrizione testuale e dei diagrammi di attività.

Descrizione testuale

Si è scelto di usare per ogni caso d'uso una descrizione testuale suddivisa in più sezioni:

Caso d'uso: una descrizione concisa dell'interazione attore/sistema

Attori: un elenco degli attori che interagiscono col sistema

Precondizioni: un elenco di condizioni che devono necessariamente essere verificate prima di compiere i passi descritti nel Flusso Principale

Flusso Principale: un elenco di passi che gli Attori e il sistema devo compiere per portare a completamento l'obiettivo descritto nel Caso d'Uso. è il corpo della descrizione del Caso d'Uso.

Flussi Alternativi: un elenco di possibili variazioni dal Flusso Principale che possono essere compiuti al verificarsi di determinate condizioni.

Diagrammi di attività

I vari diagrammi di attività rappresentano la logica di funzionamento del sistema tramite la descrizione dei flussi di elaborazione dei singoli casi d'uso. Riescono pertanto a fornire una panoramica di tutte le possibili sequenze di operazioni elementari eseguibili dal sistema a tempo di elaborazione analizzando, di ogni Caso d'Uso preso in considerazione, sia il Flusso Principale che tutti i Flussi Alternativi.

2.5.1 Accreditamento dei nodi al sistema

Caso d'Uso:

Quando un nodo periferico del sistema viene attivato si accredita al nodo centrale (Iungo Collector) passandogli un riferimento a se medesimo. Il nodo centrale mantiene una cache dei riferimenti dei nodi accreditati in un Database Nodi.

Attori:

- Nodo periferico;
- Nodo centrale;
- Database Nodi.

Precondizioni:

- Un nodo periferico si è appena attivato dopo un periodo di inattività.

Flusso principale:

1. Il nodo periferico chiama un metodo remoto del nodo centrale passandogli come parametro un riferimento alla sua interfaccia remota e una chiave di autenticazione;
2. Il nodo centrale verifica la validità della chiave di autenticazione;
3. Il nodo centrale aggiunge il riferimento al nodo periferico al Database Nodi;
4. Il nodo centrale restituisce al nodo periferico come risultato della chiamata al suo metodo remoto un valore di conferma dell'avvenuto accreditamento.

Flussi alternativi:

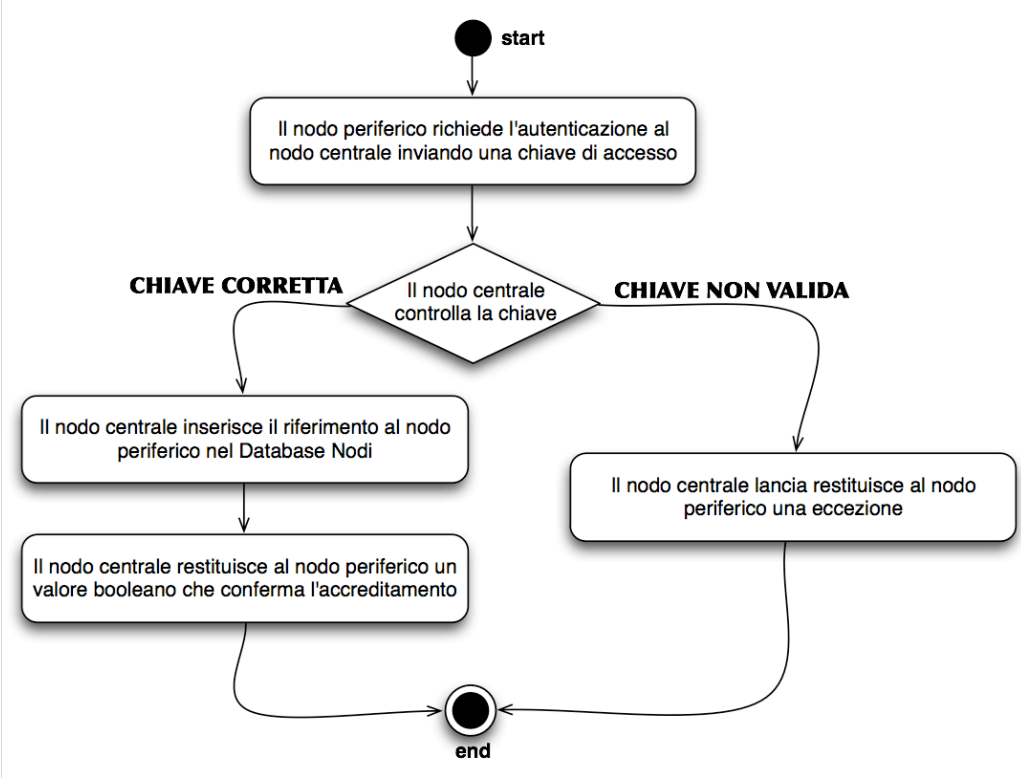

Figura 2.4: Diagramma di attività: Accreditamento dei nodi al sistema

- Il nodo centrale verifica che la chiave di autenticazione passata dal nodo periferico non è autentica : il metodo remoto di accreditamento lancia una eccezione remota che deve essere gestita dal nodo periferico e non aggiunge il riferimento al Database Nodi;

2.5.2 Accreditamento degli utenti al sistema

Caso d'Uso:

Affinchè un utente possa interagire con il sistema è necessario che si autentichi ad esso con una coppia di credenziali username/password.

Attori:

- Nodo centrale;
- Utente;
- Database Utenti.

Precondizioni:

- L'utente si è collegato al nodo centrale.

Flusso principale:

1. L'utente invia la coppia di credenziali username/password al nodo centrale tramite l'apposita interfaccia;
2. Il nodo centrale verifica l'autenticità delle credenziali con quelle conservate nel Database Utente;
3. Il nodo centrale avvisa l'utente dell'avvenuta autenticazione e gli presenta l'interfaccia per la richiesta e l'elaborazione dei dati.

Flussi alternativi:

- Il confronto tra le credenziali inserite dall'utente e quelle conservate nel Database Utenti non va a buon fine : il nodo centrale presenta un messaggio di errore e non autentica l'utente.

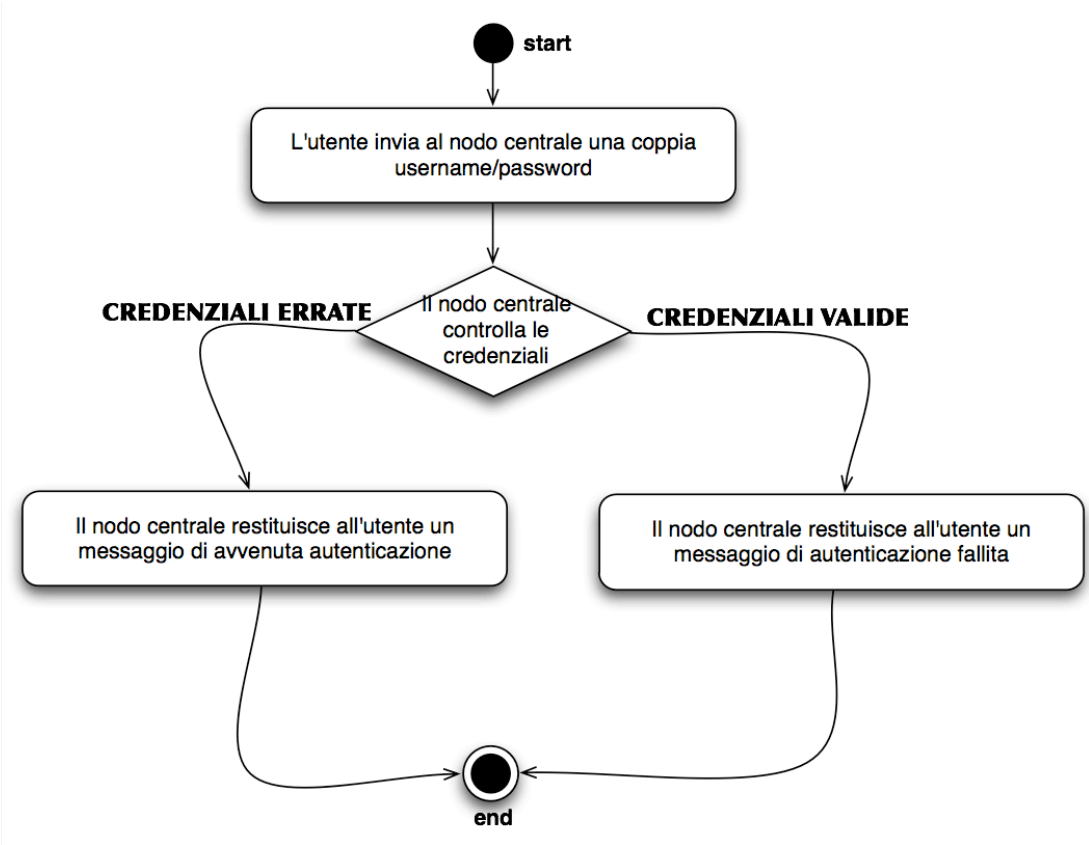

Figura 2.5: Diagramma di attività: Accreditamento degli utenti al sistema

2.5.3 Recupero dati

Caso d'Uso:

L'utente opera sul nodo centrale e richiede di visualizzare dei dati. Il nodo centrale deve recuperare i dati dai nodi periferici interessati dalla richiesta assieme alle logiche di operatività sui dati.

Attori:

- Utente;
- Nodo centrale;
- Nodi periferici;
- Database Nodi;
- Database Utenti.

Precondizioni:

- L'utente si è autenticato al sistema;
- Almeno un nodo periferico si è accreditato al nodo centrale.

Flusso principale:

1. L'utente invia una richiesta di visualizzazione di un determinato insieme di dati mediante l'interfaccia sul nodo centrale;
2. Il nodo centrale mette in relazione le informazioni contenute nel Database Utenti con quelle del Database Nodi per recuperare i riferimenti ai nodi periferici a cui l'utente autenticato può accedere;
3. Il nodo centrale invia la richiesta di dati utilizzando i metodi remoti dei riferimenti ai nodi periferici recuperati;
4. I nodi periferici recuperano i dati e li incapsulano in una struttura che li associa alle logiche di elaborazione ad essi relative e che contiene un riferimento all'interfaccia remota del nodo periferico;

5. I nodi periferici restituiscono i dati incapsulati assieme alle logiche come valore di ritorno del metodo remoto chiamato dal nodo centrale;
6. Il nodo centrale applica l'interfaccia di visualizzazione ai dati ricevuti dai nodi periferici e li presenta all'utente.

Flussi alternativi:

- Il nodo centrale non possiede le classi (o non sono aggiornate) che devono essere utilizzate per le logiche di modifica dei nodi : il nodo centrale sfrutta RMI per scaricare le classi necessarie dai nodi periferici;
- Un nodo periferico non ha dati da restituire al nodo centrale : restituisce un valore nullo senza sollevare eccezioni remote.

2.5.4 Modifica dei dati

Caso d'Uso:
L'utente modifica i dati presentati dal nodo centrale e salva le modifiche.
Attori:

- Utente;
- Nodo centrale;
- Nodi periferici;

Precondizioni:

- L'utente è autenticato al sistema;
- Almeno un nodo periferico è accreditato al nodo centrale;
- L'utente ha inviato una richiesta di recupero dati che ha restituito almeno un dato;

Flusso principale:

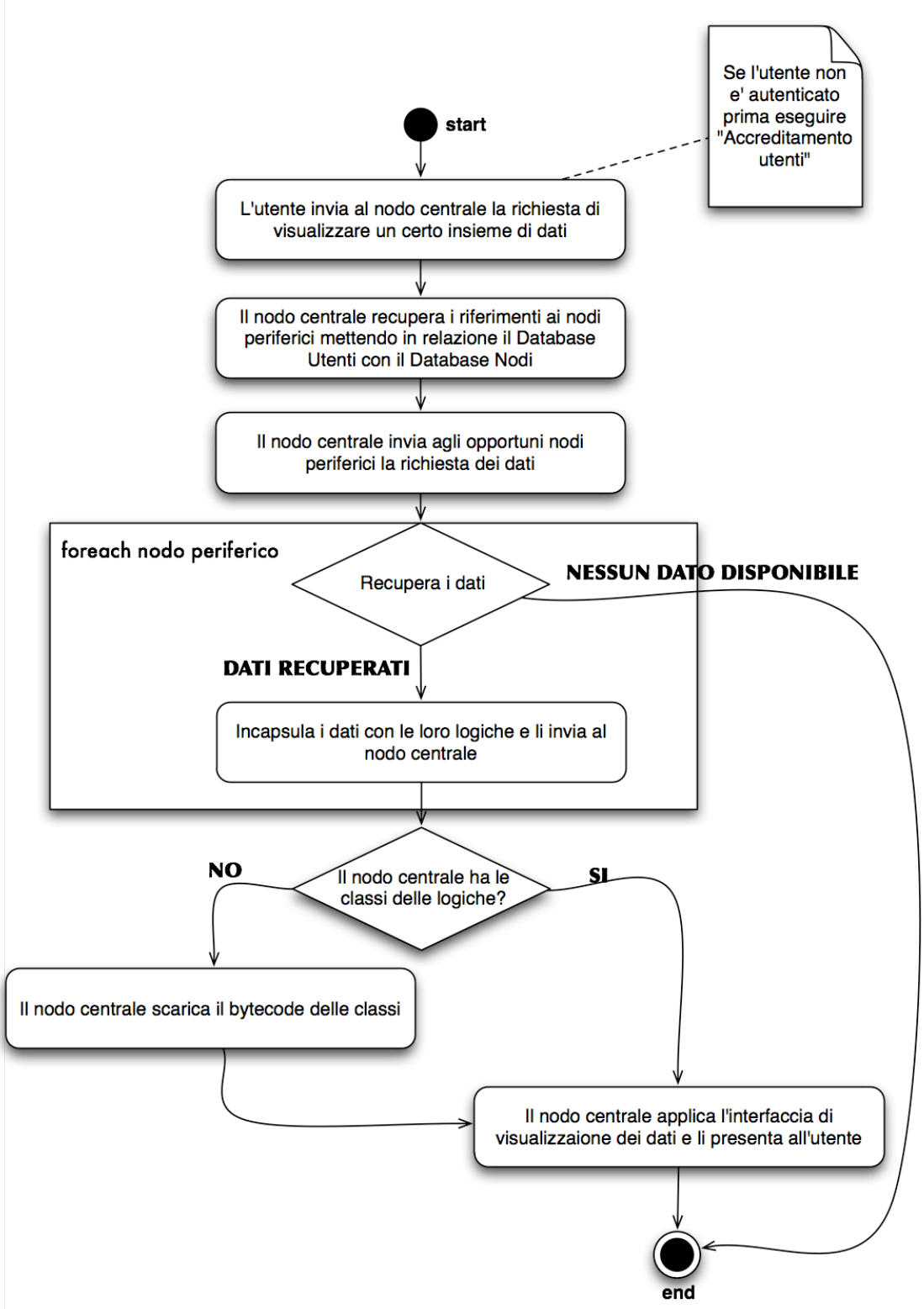

Figura 2.6: Diagramma di attività: Recupero dati

1. L'utente modifica i dati tramite l'interfaccia sul nodo centrale;
2. Il nodo centrale applica le logiche di modifica incapsulate dai nodi periferici assieme ai dati;
3. Il nodo centrale utilizza i riferimenti alle interfacce remote dei nodi periferici contenuti nelle strutture di incapsulamento dei dati per restituire i dati modificati dall'utente ed elaborati dalle logiche;
4. I nodi periferici ricevono i dati modificati e li memorizzano.

Flussi alternativi:

- Le logiche di modifica dei dati sollevano una eccezione durante l'elaborazione delle modifiche richieste dall'utente : il nodo centrale presenta un avviso all'utente e non trasmette i dati ai nodi periferici;
- I nodi periferici falliscono nel tentativo di salvare i dati modificati : durante l'esecuzione del metodo remoto invocato dal nodo centrale i nodi periferici lanciano una eccezione remota che dovrà essere gestita dal nodo centrale segnalando all'utente il fallimento dell'operazione di modifica.

2.6 Dimostratore

Oggetto di questa sezione è la descrizione della progettazione ed implementazione del dimostratore di fattibilità di Iungo Collector.

2.6.1 Requisiti e Caso d'Uso

Partendo dall'analisi dei principali casi d'uso di Iungo Collector svolta nella sezione precedente è necessario determinare un sottoinsieme di funzionalità per il dimostratore di fattibilità tali da rendere il dimostratore sufficientemente significativo da poter stabilire la fattibilità o la non fattibilità di Iungo Collector.

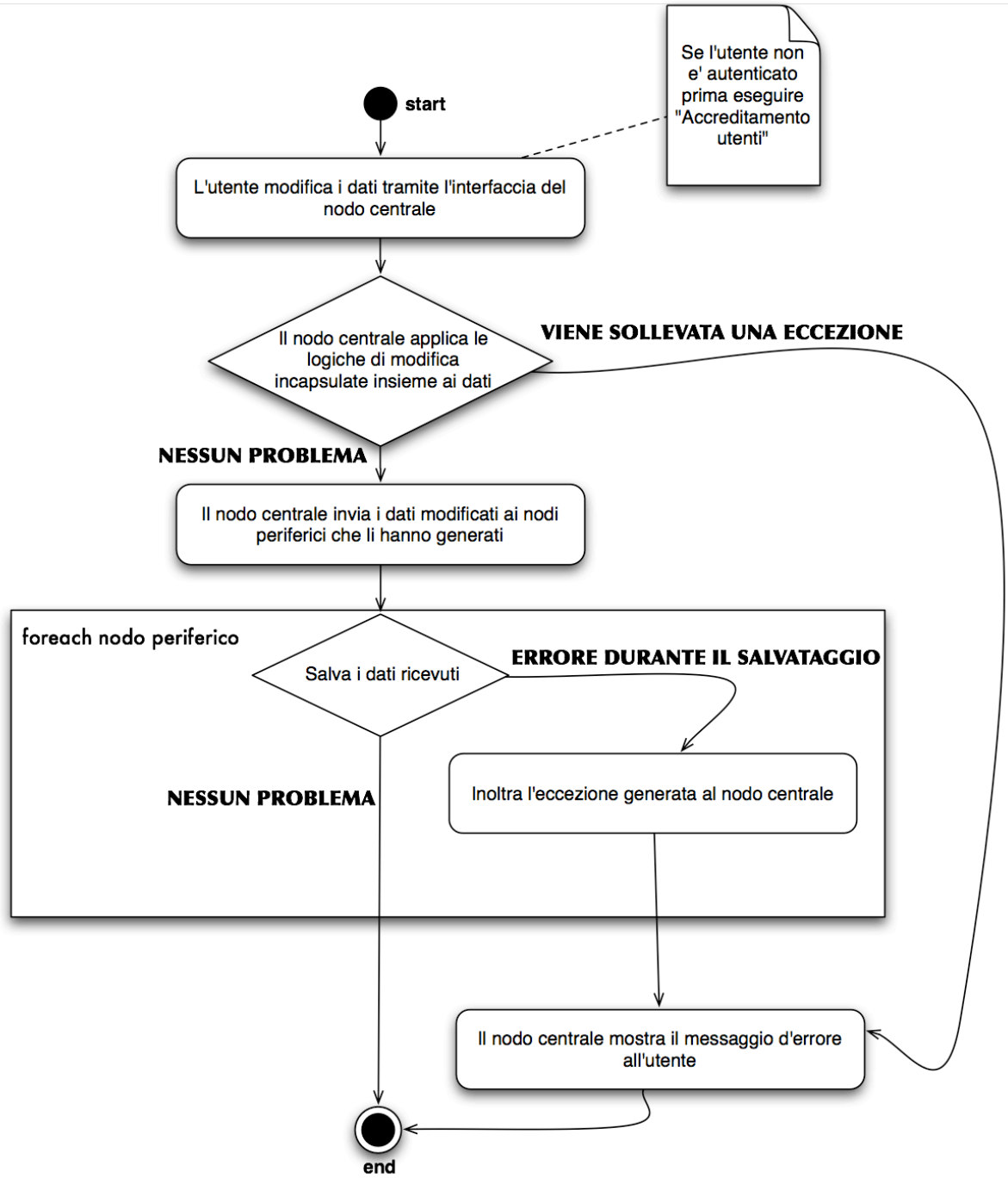

Figura 2.7: Diagramma di attività: Modifica dei dati

Di seguito vengono brevemente riportati i casi d'uso esaminati con una valutazione di rilevanza e criticità nei loro singoli aspetti:

1. **Accreditamento dei nodi al sistema**: l'implementazione delle funzionalità previste da questo caso d'uso è abbastanza triviale e fondamentalmente non presenta ostacoli o peculiarità che possano influire sulla fattibilità di Iungo Collector;

2. **Accreditamento degli utenti al sistema**: come per il caso d'uso precedente anche per l'accreditamento di utenti ad un sistema centrale mediante l'inserimento di username e password non sono rilevabili difficoltà particolari e sistemi simili sono presenti -ad esempio- in tutti gli online store;

3. **Recupero dei dati**: l'implementazione delle funzionalità previste in questo caso d'uso è sicuramente più complessa e più interessante di quella dei casi d'uso precedenti: in particolare sotto il profilo architetturale e prestazionale sono presenti le criticità relative all'incapsulamento dei dati con le logiche e il passaggio su rete di tali informazioni dai nodi periferici al nodo centrale;

4. **Modifica dei dati**: per le stesse motivazioni del il caso d'uso precedente anche le funzionalità previste in questo caso d'uso sono di sicuro interesse e non triviali.

Da queste osservazione se ne deduce che i primi due casi d'uso non siano critici per il processo decisionale che mira a stabilire la fattibilità di Iungo Collector e che invece gli ultimi due siano più interessanti e incisivi per tale decisione.

Come conseguenza di tale selezione di casi d'uso significativi è possibile redigere un elenco dei requisiti funzionali minimi che il dimostratore di fattibilità deve soddisfare:

- *dRF01*: il sistema deve essere in grado di incapsulare dati di tipo primitivo e le loro relative logiche di modifica;

- *dRF02*: il sistema deve essere in grado di trasmettere gli incapsulamenti definiti in dRF01 tra due end-point di comunicazione su rete;
- *dRF03*: il sistema deve essere in grado di far modificare i dati da un end-point di comunicazione secondo le logiche personalizzate dall'altro end-point di comunicazione;
- *dRF04*: ogni end-point deve essere in grado di segnalare all'altro end-point eventuali anomalie che possono verificarsi durante la manipolazione dei dati.

E i requisiti non funzionali:

- *dRNF00*: il sistema deve utilizzare Java come linguaggio di sviluppo e RMI come tecnologia di comunicazione fra end-point;
- *dRNF01*: il sistema deve ottimizzare al massimo le performance di comunicazioni in termini di numero di messaggi trasmessi su rete e dimensioni di tali messaggi.

Da questi requisiti e da quelli generali di Iungo Collector è possibile definire il Caso d'Uso critico per il dimostratore di fattibilità

Caso d'Uso:

Il nodo centrale richiede un dato ad un nodo periferico, lo manipola secondo le logiche specifiche del nodo periferico e lo restituisce modificato al nodo periferico.

Attori:

- Nodo periferico;
- Nodo centrale;

Precondizioni:

- Il nodo periferico è in esecuzione ed espone una interfaccia RMI raggiungibile dal nodo centrale.

Flusso principale:

1. Il nodo centrale richiede l'oggetto da modificare al nodo periferico;
2. Il nodo periferico restituisce l'oggetto incapsulato con le logiche di modifica e un riferimento a se medesimo;
3. Il nodo centrale modifica l'oggetto utilizzando le logiche in esso incapsulate;
4. Il nodo centrale restituisce al nodo periferico l'oggetto modificato.

Flussi alternativi:

- Il nodo centrale non ha le classi delle logiche di modifica : il nodo centrale le scarica dalla posizione indicata dal parametro java.rmi.server.codebase del nodo periferico;
- Le logiche di modifica sollevano eccezioni mentre il nodo centrale sta modificato l'oggetto : il nodo centrale stampa sullo standard output l'eccezione;
- Il nodo periferico solleva una eccezione in seguito alla restituzione dell'oggetto modificato : inoltra l'eccezione al nodo centrale che la stampa sul suo standard output.

2.6.2 Architettura

Per illustrare esaustivamente l'architettura del dimostratore di fattibilità realizzato in questa sezione vengono riportati di seguito i diagrammi delle classi corredati da una breve descrizione delle funzioni delle singole classi e il diagramma di sequenza della principale funzione operativa del dimostratore.

Il dimostratore di fattibilità è formato da due componenti software disposte in due JVM distinte che possono risiedere su macchine reali fisicamente distinte e connesse tra loro via LAN o via Internet. La componente che simula

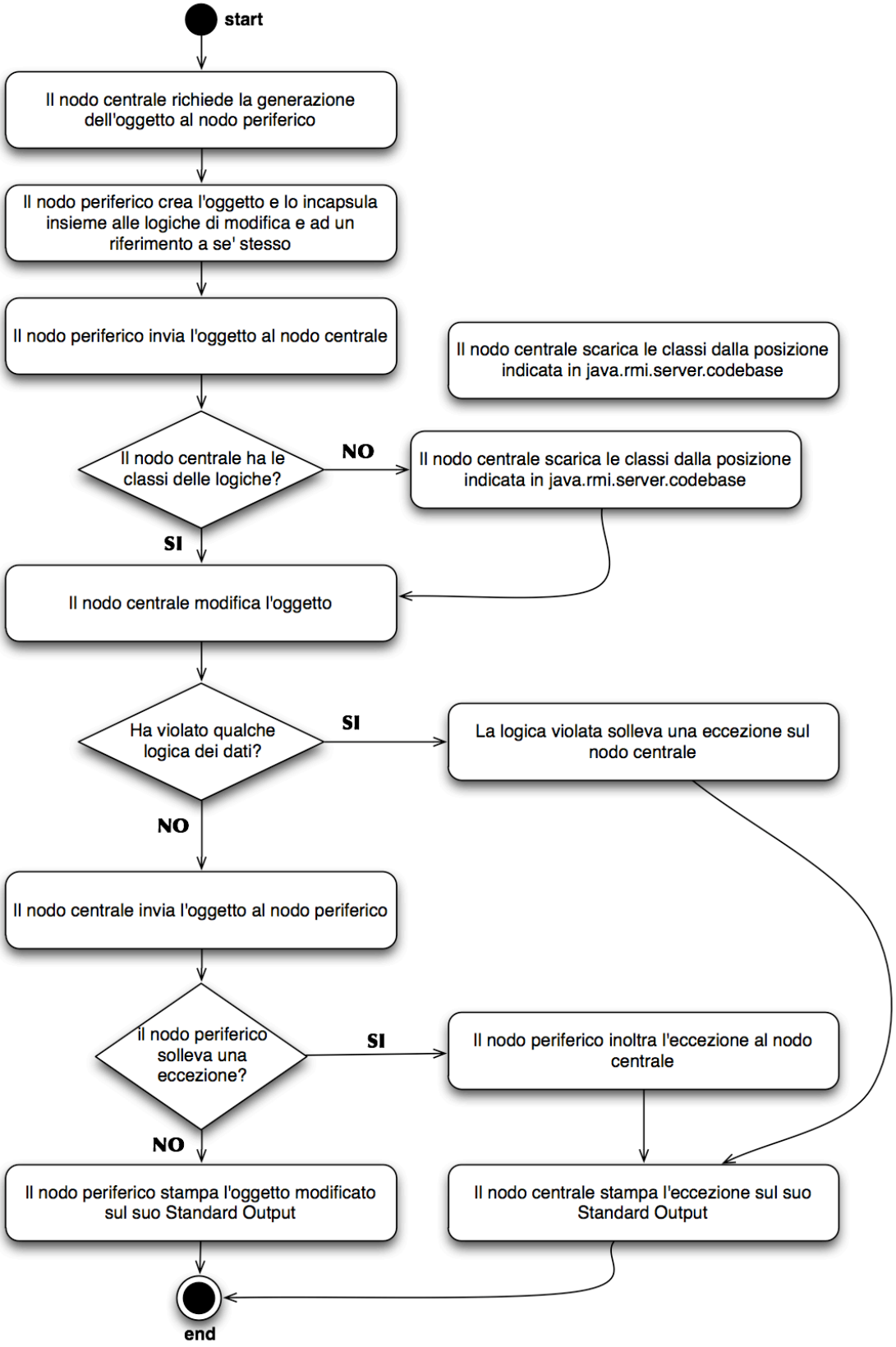

Figura 2.8: Diagramma di attività del dimostratore di fattibilità

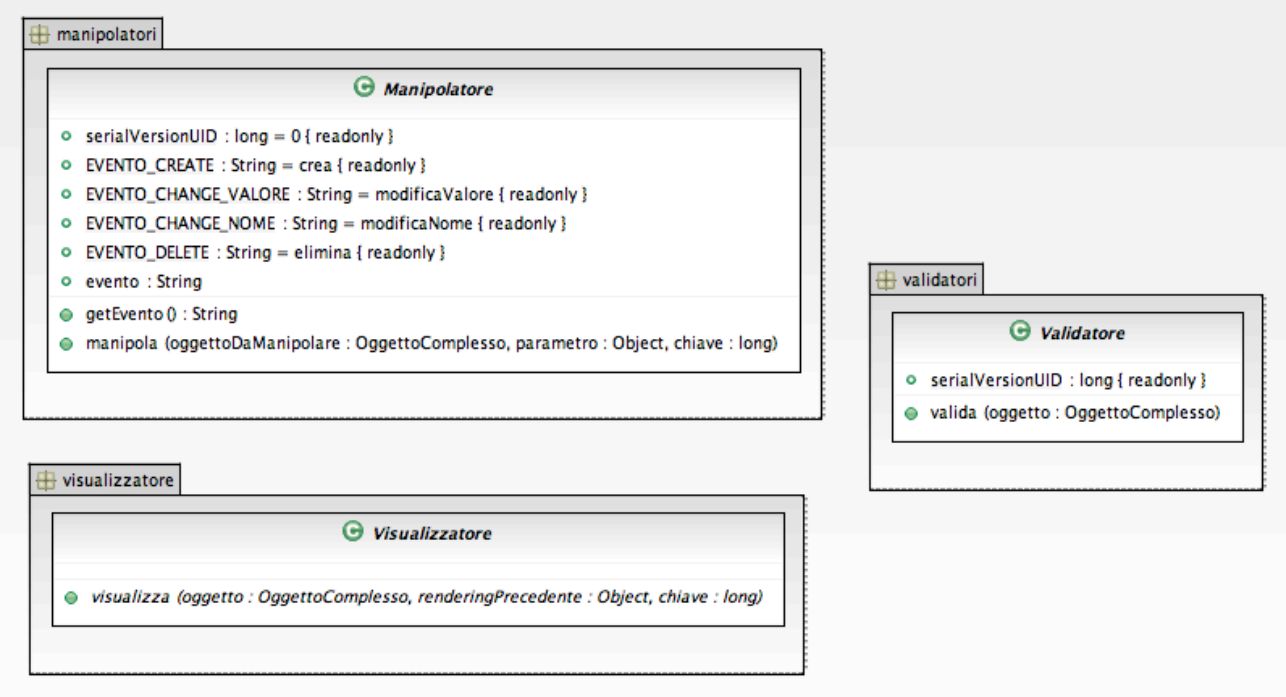

Figura 2.9: Manipolatori, Validatori e Visualizzatori

le funzionalità del nodo centrale di Iungo Collector è chiamata CentralCollector, la componente che invece rappresenta i nodi periferici della struttura di Iungo Collector è detta PeripheralCollector.

Dati e Logiche: per permettere l'operatività richiesta si sono definite una serie di interfacce che rappresentano lo standard di rappresentazione dei dati delle logiche nel dimostratore.

Le logiche di operatività sui dati sono state suddivise in tre macro-categorie a cui sono state fatte corrispondere tre classi astratte (fig 2.9): manipolatori, validatori e visualizzatori.

Ciascuna di queste classi presenta un metodo che ha come parametro l'OggettoComplesso su cui si devono applicare le logiche. Gli oggetti istanziati dalle classi che estendono queste classi astratte sono collegati agli OggettiComplessi seguendo il design pattern "Observer" e sulla base di eventi scatenati è lo stesso OggettoComplesso che si occupa di recuperare i manipo-

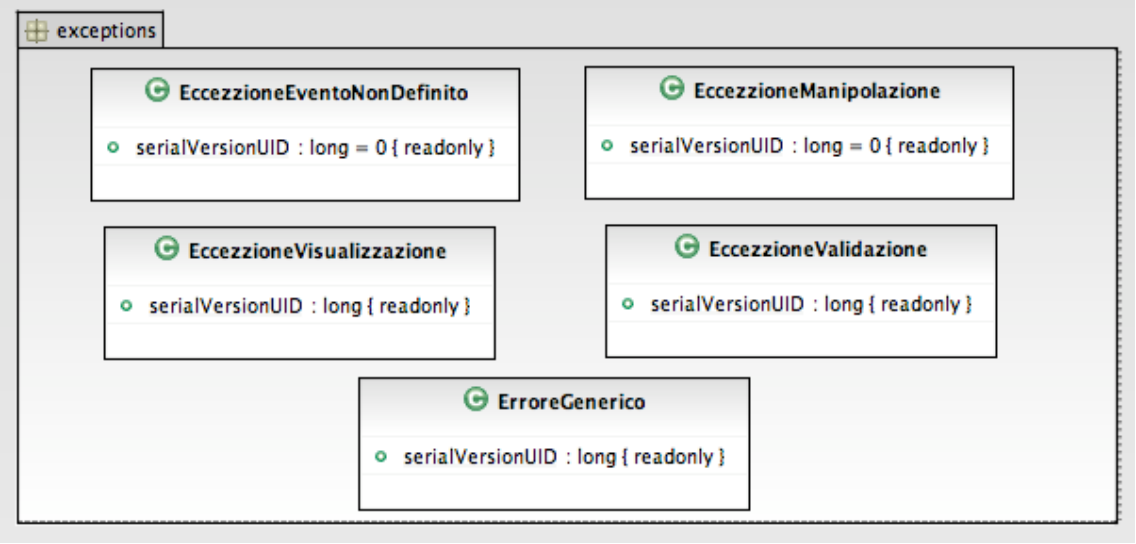

Figura 2.10: Tipi di eccezioni

latori, visualizzatori o validatori ad esso collegati e di chiamare il metodo di applicazione della logica passando sè stesso come parametro.

I metodi di queste classi astratte possono sollevare eccezioni di tipo personalizzato che vengono utilizzate per segnalare all'altro end-point di comunicazione un eventuale problema durante l'elaborazione delle logiche.

L'elemento che incapsula i dati primitivi e le logiche di modifica è definito dall'interfaccia OggettoComplesso (fig 2.11) e fornisce una interfaccia comune per tutti i tipi di dato tipicamente trattabili.

I metodi principali di questa interfaccia sono:

- *addValidatore, addManipolatore, setRendering* che permettono incapsulare nell'OggettoComplesso le logiche di validazione, modifica e rappresentazione grafica. Ogni Validatore, Manipolatore o Visualizzatore aggiunto tramite questi metodi viene memorizzato in un vettore all'interno dell'OggettoComplesso e viene utilizzato per applicare le validazione, manipolazioni o visualizzazione secondo l'ordine di inserimento;

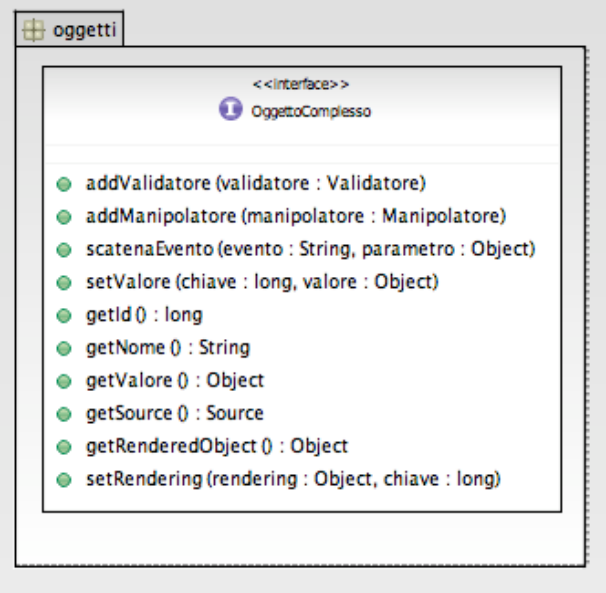

Figura 2.11: Interfaccia OggettoComplesso

- *scatenaEvento* è il metodo che viene chiamato quando è necessario compiere una azione sull'oggetto: i suoi parametri rappresentano il nome dell'evento scatenato e un parametro oggetto dell'evento;

- *getSource* è il metodo che permette di accedere all'interfaccia RemoteCollector del PeripheralCollector che ha generato l'OggettoComplesso e a cui andrà restituito dopo le elaborazioni utilizzando l'apposito metodo dell'interfaccia remota "restituisciOggetto".

Le varie classi di Eccezione, le classi astratte Manipolatore, Validatore, Visualizzatore e l'interfaccia OggettoComplesso implementano od estendono l'interfaccia Serializable rendendo così possibili le operazioni automatiche di Marshalling/UnMarshalling da parte di RMI.

Dal momento che tutte queste classi rappresentano lo standard degli elementi fondamentale del sistema, sono state impacchettate in un Jar collocato nei classpath di entrambi i componenti.

Figura 2.12: Interfaccia remota RemoteCollector

PeripheralCollector: il componente peripheral collector espone l'interfaccia remota RemoteCollector (fig 2.12) che fornisce un punto di accesso ai dati da parte del CentralCollector.

RemoteCollector estende l'interfaccia java.rmi.Remote ed tutti i suoi metodi possono sollevare una eccezione di tipo RemoteException per segnalare al CentralCollector un eventuale problema avvenuto in remoto.

I metodi principali di questa interfaccia sono:

- generaOggetto che permette di generare un OggettoComplesso avente il nome ed il valore primitivo passati come parametro;
- restituisciOggetto che permette di restituire al PeripheralCollector un oggetto precedentemente da lui generato e che abbia eventualmente subito modifiche.

Il PeripheralCollector ha anche le sue implementazioni delle logiche di modifica (fig 2.13).

Come già accennato in precedenza queste classi estendono le classi astratte Manipolatore, Validatore e Visualizzatore e sono responsabili dell'applicazione delle logiche di un PeripheralCollector sui dati da lui forniti.

Dal momento che ogni PeripheralCollector possiede le sue specifiche logiche eventualmente diverse da tutte quelle di altri PeripheralCollector, il codice oggetto generato dalla compilazione da queste classi (il bytecode contenuto nei file .class) deve essere collocato in una posizione raggiungibile da parte del CentralCollector (via HTTP o su file-system condiviso) e il parametro

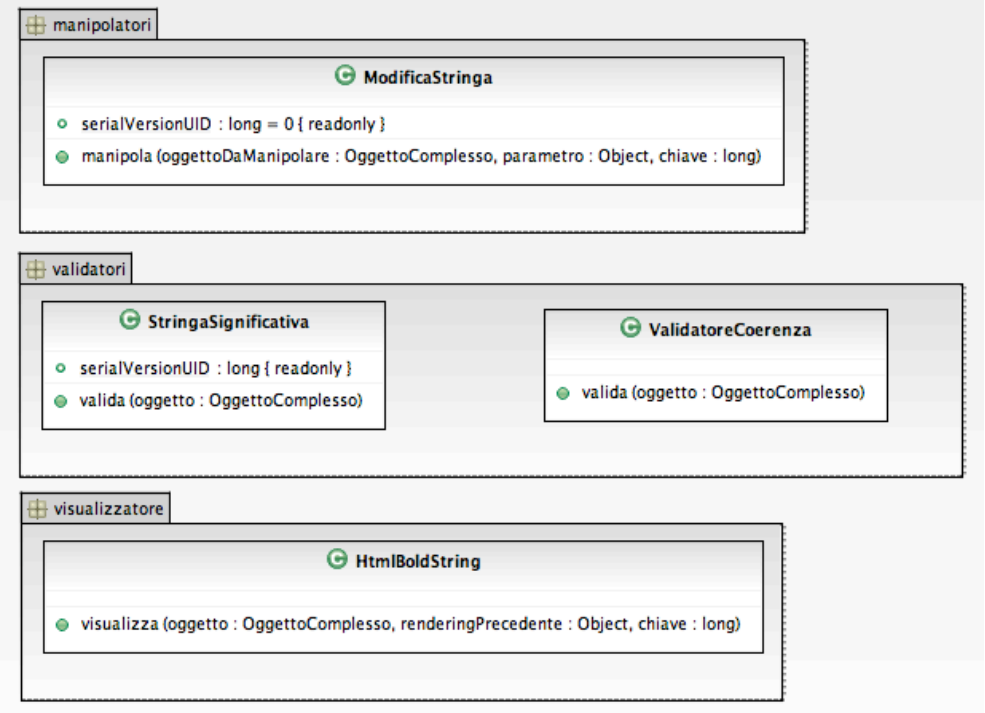

Figura 2.13: Implementazioni di Validatore, Manipolatore e Visualizzatore

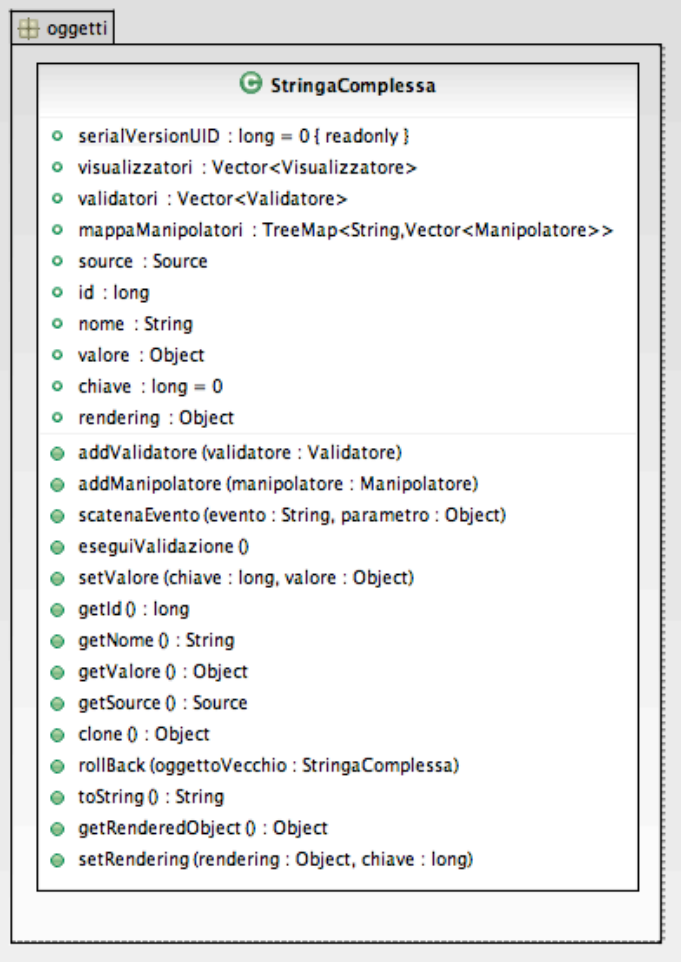

Figura 2.14: La classe StringaComplessa implementa OggettoComplesso

"java.rmi.server.codebase" della JVM su cui il PeripheralCollector esegue deve riportare tale posizione. In questo modo il CentralCollector che si collega ad uno specifico PeripheralCollector sa da dove scaricare in maniera automatizzata e trasparente al sistema il bytecode delle sue specifiche logiche.

Discorso identico vale anche per i dati: il PeripheralCollector implementa in StringaComplessa (fig 2.14) l'interfaccia OggettoComplesso per incapsulare il dato pseudo-primitivo di tipo String con le sue logiche specifiche di modifica, visualizzazione e validazione.

CentralCollector: il CentralCollector di fatto contiene soltanto la classe

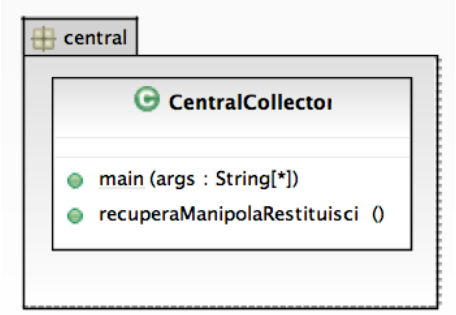

Figura 2.15: La classe Central Collector

"CentralCollector" (fig 2.15) che si occupa di recuperare, modificare in qualche modo e restituire i dati dei PeripheralCollector.

Diagramma di sequenza: il diagramma di sequenza riportato in figura 2.16 rappresenta la sequenza dei messaggi e delle operazioni durante una esecuzione del dimostratore di fattibilità.

Di seguito una breve descrizione di quanto si vede nel diagramma di sequenza:

1. Il CentralCollector richiede al PeripheralCollector un certo dato chiamando il metodo "generaOggetto" di RemoteCollector;

2. Il PeripheralCollector incapsula il dato, le logiche di modifica e il riferimento a sè medesimo in un nuovo OggettoComplesso;

3. Il PeripheralCollector restituisce l'OggettoComplesso creato come valore di ritorno del metodo chiamato dal CentralCollector;

4. Appena il CentralCollector riceve l'OggettoComplesso verifica di avere nel suo classpath tutte le classi necessarie all'elaborazione: se non le trova o non sono aggiornate le recupera dalla posizione indicata dal parametro java.rmi.server.codebase del PeripheralCollector;

5. Il CentralCollector modifica i dati dell'OggettoComplesso che a sua volta utilizza i Manipolatori, Validatori e Visualizzatori ad esso collegati per applicare le logiche del PeripheralCollector;

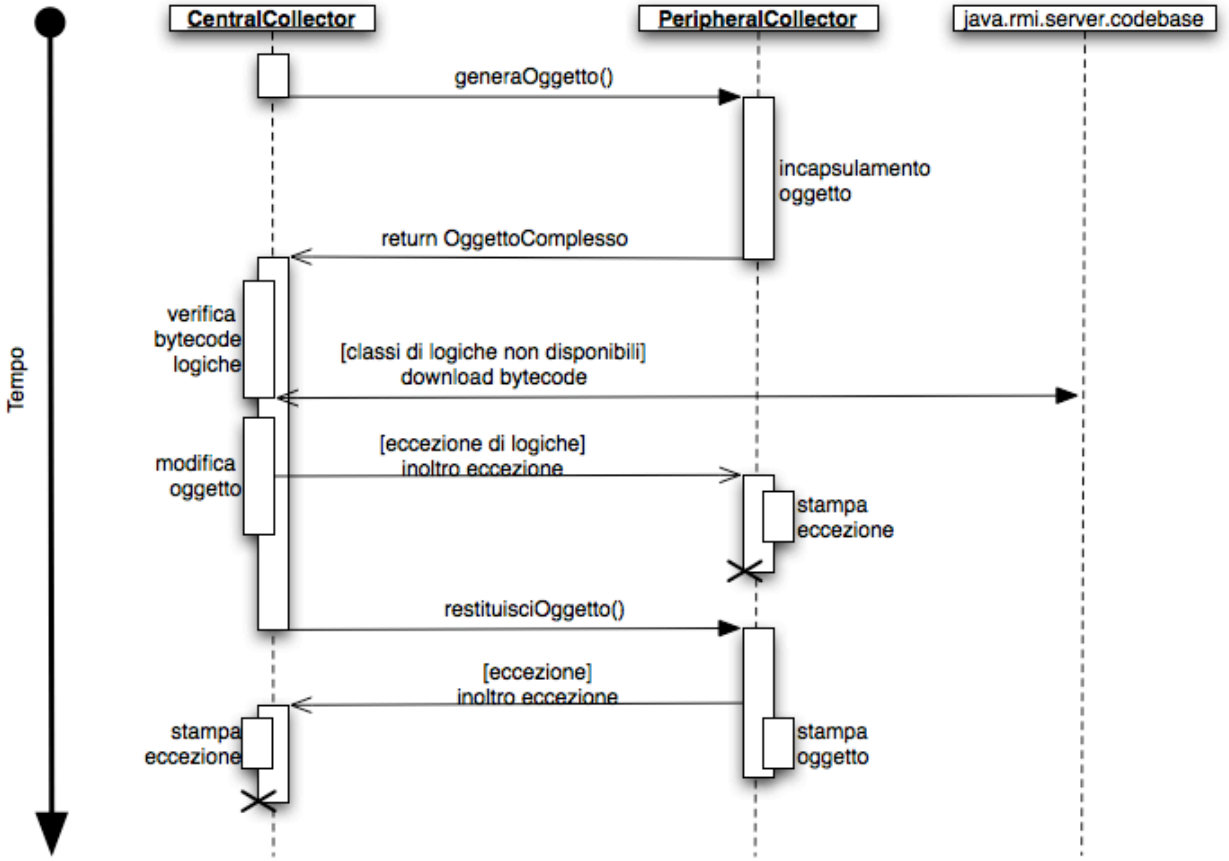

Figura 2.16: Diagramma di sequenza del dimostratore di fattibilità

6. Il CentralCollector chiama il metodo “restituisciOggetto” per far ritornare l'OggettoComplesso da lui modificato al PeripheralCollector.

2.6.3 Valutazione sperimentale

La valutazione sperimentale del dimostratore di fattibilità è stata suddivisa in tre parti:

1. Realizzazione di un ambiente di test e definizione dei tipi test significativi;
2. Valutazione della soddisfazione dei requisiti del dimostratore enunciati in sezione 2.6.1;
3. Misurazione delle performance in termini di numero di messaggi scambiati tra i componenti e dimensione di tali messaggi.

Per ottenere una valutazione precisa ed affidabile, si è utilizzato uno strumento di misurazione delle performance chiamato “RMI plugin for Eclipse 2.0” che -grazie alla sua funzionalità RMI Spy- ha permesso di creare un log real-time di tutte le comunicazioni effettuate tramite RMI. è comunque da tenere in considerazione che tale strumento, per quanto efficiente, ha introdotto un overhead di comunicazione difficilmente stimabile con precisione ma sicuramente presente.

Ambiente di test: per effettuare le varie valutazioni sopra menzionate si è predisposto un ambiente di test così formato:

- Le componenti CentralCollector e PeripheralCollector sono state disposte sulla stessa macchina fisica ma su due JVM differenti;
- PeripheralCollector è stato configurato per restare in ascolto sulla porta 7643 di localhost (127.0.0.1);
- Le classi di logica personalizzata di PeripheralCollector sono state raccolte in un Jar (collector.jar)che è stato posizionato nel webroot di un

Apache Web Server in esecuzione su localhost e in ascolto sulla porta 80;

- è stato creato uno script di esecuzione per PeripheralCollector che includesse il parametro java.rmi.server.codebase = `http://localhost/collector.jar`;
- è stato messo in esecuzione il servizio di rmiregistry sulla porta 1099.

L'esecuzione del test comporta che il CentralCollector richieda al PeripheralCollector la creazione di un oggetto di tipo StringaComplessa, modifichi tale oggetto nel suo contenuto e restituisca l'oggetto modificato al PeripheralCollector che provveda a stamparlo sul suo Standard Output. Il PeripheralCollector è stato configurato in modo che associ alla StringaComplessa delle logiche di modifica, di visualizzazione e di validazione tali per cui:

- Tutte le modifiche fatte alla stringa avranno il carattere "a" sostituito con il carattere "b";
- Non sono ammesse stringhe vuote o più lunghe di 20 caratteri;
- Se richiesta la visualizzazione del contenuto della StringaComplessa, questa viene formattata in HTML in modo che risulti un paragrafo HTML con contenuto in grassetto (bold).

Tali logiche sono contenute in classi (che implementano le interfacce standard di Visualizzatore, Manipolatore e Validatore) che non sono inizialmente disponibili al CentralController ma sono presenti in `http://localhost/collector.jar`.

Soddisfazione dei requisiti: per poter affermare che il dimostratore realizzato effettivamente dimostri o meno la fattibilità di Iungo Collector è necessario valutare quanto la sua implementazione sia risultato aderente ai requisiti illustrati in sezione 2.6.1.

Per ottenere un test BlackBox esaustivo sono stati realizzati tre tipi di esecuzione con input diversi:

1. Il CentralCollector modifica la StringaComplessa ottenuta dal PeripheralCollector assegnando come valore "bar bar bar bar". Ci si aspetta che alla fine della esecuzione il PeripheralCollector visualizzi sul suo Standard Output "bbr bbr bbr bbr";

2. Il CentralCollector modifica la StringaComplessa ottenuta dal PeripheralCollector assegnando come valore una stringa vuota. Ci si aspetta che al momento della modifica sul CentralCollector le logiche incapsulate con la StringaComplessa sollevino una eccezione e che il CentralCollector la visualizzi sul suo Standard Error;

3. Il CentralCollector modifica la StringaComplessa ottenuta dal PeripheralCollector assegnando come valore "bar bar bar bar" ma il PeripheralCollector solleva una eccezione di errore generico quando riceve la StringaComplessa modificata. Ci si aspetta che l'eccezione venga notificata anche al CentralCollector e che questo provveda a visualizzarla sul suo Standard Error.

Con il primo e il secondo tipo di test è possibile verificare che i requisiti funzionali (sezione 2.6.1) *dRF01, dRF02* e *dRF03* siano soddisfatti, mentre con il terzo è possibile verificare che anche *dRF04* sia soddisfatto.

Per verificare la soddisfazione di questi requisiti è opportuno utilizzare lo strumento RMI Spy per controllare che effettivamente lo schema di comunicazione tra i due componenti rispecchi il caso d'uso illustrato in sezione 2.6.1.

Di seguito vengono riportate le tabelle che mostrano la sequenza delle azioni compiute dai componenti come risultato del lancio dei tre test sopra descritti.

Da questa tabella si nota come durante il Test 1 il sistema si sia comportato esattamente come da requisiti e l'output finale risulta quello atteso:

- Il PeripheralCollector si ha registrato il suo nome nel rmiregistry;

TEST 1			
TIPO	**COMPONENTE**	**TIMESTAMP**	**AZIONE**
Outgoing Calls	PeripheralCollector	12:04:10,140	void rebind(String, java.rmi.Remote)
Loader Events	PeripheralCollector	12:04:10,171	Proxy[java.rmi.Remote, it.iungo.collector.RemoteCollector]
Incoming Calls	PeripheralCollector	12:04:10,171	void rebind(String, java.rmi.Remote)
Outgoing Calls	Central Collector	12:05:42,843	java.rmi.Remote.lookup(String)
Incoming Calls	PeripheralCollector	12:05:42,858	java.rmi.Remote.lookup(String)
Loader Events	Central Collector	12:05:42,890	Proxy[java.rmi.Remote, it.iungo.collector.RemoteCollector]
Outgoing Calls	Central Collector	12:05:42,921	OggettoComplesso generaOggetto(String, Object)
Incoming Calls	PeripheralCollector	12:05:42,921	OggettoComplesso generaOggetto(String, Object)
Loader Events	Central Collector	12:05:42,952	it.iungo.collector.oggetti.StringaComplessa
Loader Events	Central Collector	12:05:42,968	it.iungo.collector.manipolatori.ModificaStringa
Loader Events	Central Collector	12:05:42,968	it.iungo.collector.interfaces.manipolatori.Manipolatore
Loader Events	Central Collector	12:05:42,968	it.iungo.collector.Source
Loader Events	Central Collector	12:05:42,968	it.iungo.collector.validatori.StringaSignificativa
Loader Events	Central Collector	12:05:42,968	it.iungo.collector.interfaces.validatori.Validatore
Loader Events	Central Collector	12:05:42,968	it.iungo.collector.visualizzatore.HtmlBoldString
Loader Events	Central Collector	12:05:42,968	it.iungo.collector.visualizzatore.Visualizzatore
Outgoing Calls	Central Collector	12:05:42,983	void restituisciOggetto(OggettoComplesso)
Incoming Calls	PeripheralCollector	12:05:42,983	void restituisciOggetto(OggettoComplesso)
Output	PeripheralCollector	12:05:42,985	<p>bbr bbr bbr bbr/p>

Figura 2.17: Azioni occorse nella esecuzione del Test 1

- Il CentralCollector ha cercato il nome del PeripheralCollector nel registro;
- Il CentralCollector ha richiesto al PeripheralCollector di generare un OggettoComplesso;
- In seguito alla ricezione della stringa complessa come risultato del passo precedente il CentralCollector ha recuperato da `http://localhost/collector.jar` le implementazioni degli Oggetti di logica incapsulati nella StringaComplessa (anch'essa recuperata nella sua implementazione allo stesso modo);
- Il CentralCollector ha inviato la StringaComplessa modificata al PeripheralCollector;
- Il PeripheralCollector ha stampato sul suo StandardOutput “bbr bbr bbr bbr”.

Dalla tabella sopra riportata si nota come anche durante il Test 2 il sistema si sia comportato esattamente come da requisiti e l'output finale risulta quello atteso:

TEST 2			
TIPO	COMPONENTE	TIMESTAMP	AZIONE
Outgoing Calls	PeripheralCollector	16:34:26,968	void rebind(String, java.rmi.Remote)
Loader Events	PeripheralCollector	16:34:26,983	Proxy[java.rmi.Remote, it.iungo.collector.RemoteCollector]
Incoming Calls	PeripheralCollector	16:34:26,983	void rebind(String, java.rmi.Remote)
Outgoing Calls	Central Collector	16:34:32,233	java.rmi.Remote.lookup(String)
Incoming Calls	PeripheralCollector	16:34:32,249	java.rmi.Remote.lookup(String)
Loader Events	Central Collector	16:34:32,265	Proxy[java.rmi.Remote, it.iungo.collector.RemoteCollector]
Outgoing Calls	Central Collector	16:34:32,311	OggettoComplesso generaOggetto(String, Object)
Incoming Calls	PeripheralCollector	16:34:32,311	OggettoComplesso generaOggetto(String, Object)
Loader Events	Central Collector	16:34:32,327	it.iungo.collector.oggetti.StringaComplessa
Loader Events	Central Collector	16:34:32,358	it.iungo.collector.manipolatori.ModificaStringa
Loader Events	Central Collector	16:34:32,358	it.iungo.collector.interfaces.manipolatori.Manipolatore
Loader Events	Central Collector	16:34:32,358	it.iungo.collector.Source
Loader Events	Central Collector	16:34:32,358	it.iungo.collector.validatori.StringaSignificativa
Loader Events	Central Collector	16:34:32,358	it.iungo.collector.interfaces.validatori.Validatore
Loader Events	Central Collector	16:34:32,358	it.iungo.collector.visualizzatore.HtmlBoldString
Loader Events	Central Collector	16:34:32,358	it.iungo.collector.visualizzatore.Visualizzatore
Output	Central Collector	16:34:32,400	it.iungo.collector.EccezioneValidazione: la stringa non puo' e

Figura 2.18: Azioni occorse nella esecuzione del Test 2

- Il PeripheralCollector si ha registrato il suo nome nel rmiregistry;
- Il CentralCollector ha cercato il nome del PeripheralCollector nel registro;
- Il CentralCollector ha richiesto al PeripheralCollector di generare un OggettoComplesso;
- In seguito alla ricezione della stringa complessa come risultato del passo precedente il CentralCollector ha recuperato da `http://localhost/collector.jar` le implementazioni degli Oggetti di logica incapsulati nella StringaComplessa (anch'essa recuperata nella sua implementazione allo stesso modo);
- Il CentralCollector mostra in output una eccezione che riporta la violazione di una logica stabilita dal PeripheralCollector: il contenuto della stringa non può essere vuoto.

Dalla tabella sopra riportata si nota come anche durante il Test 3 il sistema si sia comportato esattamente come da requisiti e l'output finale risulta quello atteso:

TEST 3			
TIPO	COMPONENTE	TIMESTAMP	AZIONE
Outgoing Calls	PeripheralCollector	14:43:13,483	void rebind(String, java.rmi.Remote)
Loader Events	PeripheralCollector	14:43:13,515	Proxy[java.rmi.Remote, it.iungo.collector.RemoteCollector]
Incoming Calls	PeripheralCollector	14:43:13,515	void rebind(String, java.rmi.Remote)
Outgoing Calls	Central Collector	14:43:21,718	java.rmi.Remote.lookup(String)
Incoming Calls	PeripheralCollector	14:43:21,733	java.rmi.Remote.lookup(String)
Loader Events	Central Collector	14:43:21,765	Proxy[java.rmi.Remote, it.iungo.collector.RemoteCollector]
Outgoing Calls	Central Collector	14:43:21,811	OggettoComplesso generaOggetto(String, Object)
Incoming Calls	PeripheralCollector	14:43:21,811	OggettoComplesso generaOggetto(String, Object)
Loader Events	Central Collector	14:43:21,827	it.iungo.collector.oggetti.StringaComplessa
Loader Events	Central Collector	14:43:21,843	it.iungo.collector.manipolatori.ModificaStringa
Loader Events	Central Collector	14:43:21,843	it.iungo.collector.interfaces.manipolatori.Manipolatore
Loader Events	Central Collector	14:43:21,858	it.iungo.collector.Source
Loader Events	Central Collector	14:43:21,858	it.iungo.collector.validatori.StringaSignificativa
Loader Events	Central Collector	14:43:21,858	it.iungo.collector.interfaces.validatori.Validatore
Loader Events	Central Collector	14:43:21,858	it.iungo.collector.visualizzatore.HtmlBoldString
Loader Events	Central Collector	14:43:21,858	it.iungo.collector.visualizzatore.Visualizzatore
Outgoing Calls	Central Collector	14:43:21,858	void restituisciOggetto(OggettoComplesso)
Incoming Calls	PeripheralCollector	14:43:21,858	void restituisciOggetto(OggettoComplesso)
Loader Events	Central Collector	14:43:21,874	it.iungo.collector.exceptions.ErroreGenerico
Output	Central Collector	14:43:21,875	it.iungo.collector.exceptions.ErroreGenerico: errore nel salva

Figura 2.19: Azioni occorse nella esecuzione del Test 3

- Il PeripheralCollector si ha registrato il suo nome nel rmiregistry;
- Il CentralCollector ha cercato il nome del PeripheralCollector nel registro;
- Il CentralCollector ha richiesto al PeripheralCollector di generare un OggettoComplesso;
- In seguito alla ricezione della stringa complessa come risultato del passo precedente il CentralCollector ha recuperato da `http://localhost/collector.jar` le implementazioni degli Oggetti di logica incapsulati nella StringaComplessa (anch'essa recuperata nella sua implementazione allo stesso modo);
- Il CentralCollector ha inviato la StringaComplessa modificata al PeripheralCollector;
- Il PeripheralCollector sollevato una eccezione durante il salvataggio che ha portato il CentralCollector a caricare l'oggetto ErroreGenerico;

TIPO	COMPONENTE	TIMESTAMP	BYTES	AZIONE	PARAMETRI
Outgoing Calls	PeripheralCollector	14:43:13,483	406	void rebind(String, java.rmi.Remote)	arg0: [java.lang.String]='FornitoreOggetti', written ~55 bytes arg1: [it.iungo.collector.peripheral.Collector], written ~351 bytes
Incoming Calls	PeripheralCollector	14:43:13,515	0	void rebind(String, java.rmi.Remote)	
Outgoing Calls	Central Collector	14:43:21,718	434	java.rmi.Remote.lookup(String)	arg0: [java.lang.String]='FornitoreOggetti', written ~55 bytes returned: [$Proxy0], read ~379 bytes
Incoming Calls	PeripheralCollector	14:43:21,733	0	java.rmi.Remote.lookup(String)	

Figura 2.20: Messaggi scambiati durante il bind e lookup

- Il CentralCollector mostra in output l'eccezione sollevata dal PeripheralCollector che riporta un problema intervenuto durante il salvataggio dei dati.

Performance: durante l'esecuzione dei test descritti in precedenza RMI Spy ha permesso di conoscere il numero e la dimensione dei messaggi scambiati fra il Central e il PeripheralCollector:

- **Bind e lookup:** sebbene sia una operazione eseguita "una tantum" è corretto conteggiare anche i messaggi scambiati durante la fase connessione fra il CentralCollector e il PeripheralCollector. In tutti e tre i tipi di test sono transitati 4 messaggi messaggi per le operazioni di binding e lookup.

  Da questa tabella si può vedere come le operazioni di binding e di lookup abbiano richiesto il transito di 840 bytes sulla rete. Lavorando sulla dimensione parametri (arg0) si può ridurre questa dimensione fino ad minimo di 732bytes;

- **Recupero dati:** l'operazione di recupero dati per il dimostratore consiste nella chiamata del metodo "generaOggetto" e nel suo valore di ritorno. Per tutti e tre i test eseguiti questa operazione ha richiesto che esattamente due messaggi transitassero su rete: uno con la richiesta del metodo e i relativi parametri, l'altro con il valore di ritorno.

  Dalla tabella sopra riportata si può vedere come l'operazione di recupero dati in questo caso abbia comportato il transito su rete di

TIPO	COMPONENTE	BYTES	AZIONE	PARAMETRI
Outgoing Calls	Central Collector	1388	OggettoComplesso generaOggetto(String, Object)	arg0: [java.lang.String]='Test', written ~43 bytes arg1: [java.lang.String]='foo bar foo bar', written ~18 bytes returned: [it.iungo.collector.oggetti.StringaComplessa], read ~1327 bytes

Figura 2.21: Messaggi scambiati durante il recupero dati

TIPO	COMPONENTE	BYTES	AZIONE	PARAMETRI
Outgoing Calls	Central Collector	1327	void restituisciOggetto(OggettoComplesso)	arg0: [it.iungo.collector.oggetti.StringaComplessa], written ~1327 byte

Figura 2.22: Messaggi scambiati durante la restituzione dei dati

1388 bytes. Analizzando il peso dei singoli parametri si può vedere come in questo caso la StringaComplessia sia l'elemento che più incide sulla dimensione dei messaggi: oltre al peso della stringa primitiva e del suo nome (˜18 bytes e ˜43 bytes come da tabella) infatti l'OggettoComplesso contiene tutte le strutture dati che sono indispensabili all'esecuzione della sua logica.

- **Restituzione dati:** l'operazione di restituzione dei dati consiste nella chiamata del metodo "restituisciOggetto" del PeripheralCollector**.** Dato che il metodo "restituisciOggetto" presenta come valore di ritorno "null" è stato sufficiente il transito su rete di un solo messaggio che contenesse la richiesta del metodo e il relativo parametro.

  In questo caso il peso del messaggio ammonta a 1327 bytes e sono valide anche per questa situazione tutte le considerazioni sul peso di StringaComplesse fatte al punto precedente.

In tutti i casi presi in esame lo scambio dei messaggi coincide con il numero minimo di messaggi scambiabili per ottenere il risultato desiderato e risulta che la loro dimensione dipende essenzialmente dalla dimensione dei parametri passati.

La dimensione di tali parametri è determinata da due fattori: la dimensione effettiva del dato primitivo contenuto e -nel caso di OggettoComplesso

e relative implementazioni- dalle strutture che permettono l'applicazione di logiche generate dalla sorgente dei dati.

Lavorando quindi per una ottimizzazione della dimensione di tali parametri (in particolar modo sulla ottimizzazione della struttura di OggettoComplesso) è possibile ottenere risultati più che soddisfacenti.

Capitolo 3

Conclusioni

Come descritto nel capitolo introduttivo di questa tesi, gli obiettivi principali di questo documento cosistevano nel:

- Fornire una panoramica approfondita dello stato dell'arte delle maggiori tecnologie che possono interessare la realizzazione di un software per la raccolta ed analisi di dati su Cloud Computing;
- Affrontare le motivazioni tecniche, commerciali e culturali che possono influire sulle scelte progettuali per la realizzazione di Iungo Collector;
- Realizzare uno studio di fattibilità per Iungo Collector che ne determini la possibilità di realizzazione e le criticità di progetto.

Nel capitolo 1 è stato presentato lo stato dell'arte di Iungo: un software per la supply chain collaboration sviluppato da Smarten S.r.l. le cui potenzialità costiuiscono le principali motivazioni alla domanda di fattibilità di Iungo Collector. In seguito si e' fornito un panorama dello stato dell'arte del paradigma di Cloud Computing nelle sue varie declinazioni: cloud pubbliche, private, ibride o private virtuali ed inoltre sono stati illustriati i vari attori e le loro interazioni nel modello di buisness legato alle sue tecnologie. Quindi è stata fornita una visione ragionata dei punti di forza e delle debolezze delle tecnologie Java RMI e Webservices SOAP e le loro possibili applicazioni.

Questa analisi è stata utilizzata per la realizzazione dello studio di fattibilità di Iungo Collector: si sono messe a confronto le varie alternative tecnologiche del panorama attuale con i requisiti funzionali e non funzionali del sistema. In particolare è risultato che la soluzione migliore per Iungo Collector sia quella di utilizzare Java RMI come tecnologia di comunicazione fra le sue componenti distribuite e disporre il componente centrale (il tier di presentazione) su una Cloud Pubblica prestando particolare attenzione al SLA offerto dall'infrastructure provider e alla sua reputazione.

Per riuscire a dare una risposta precisa alla domanda di fattibilità di Iungo Collector si è realizzato un dimostratore di funzionalità che implementa le funzionalità più critiche fra quelle presenti nei requisiti funzionali e nei casi d'uso di Iungo Collector. Dai vari test di valutazione eseguiti sul dimostratore è risultato che non solo la Iungo Collector è fattibile con la struttura e secondo i casi d'uso presi in esame, ma è anche risultato che il meccanismo di comunicazione basato su Java RMI progettato è prestazionalmente più che accettabile: il numero di messaggi scambiati su rete sono al limite minimo ottenibile e il la loro dimensione risulta tutto sommato contenuta e con margini di ottimizzazione.

Si può quindi affermare non solo che gli obiettivi posti per questa tesi sono stati raggiunti, ma anche che le soluzioni proposte dallo studio di fattibilità qui descritto sono state accolte con soddisfazione da Smarten s.r.l. e partendo da quanto realizzato in questa tesi saranno fatte valutazioni commerciali approfondite per definire i termini e i tempi di sviluppo migliori per Iungo Collector. Risulta praticamente certo che Iungo Collector sia destinato a prendere forma e a concretizzarsi nel prossimo futuro entrando a far parte della soluzione di Iungo per l'integrazione della Supply Chain aziendale come elemento di spicco ed innovaitvo.

Bibliografia

[1] Coulouris G, Dollimore J, Kindberg T, *Distributed Systems: Concepts and Design* 4th ed., cap 5, 15 e 19, Addison Wesley 2005

[2] Ismail Ari, Bo Hong, Ethan L. Miller, Scott A. Brandt, Darrell D. E. Long, *Managing Flash Crowds on the Internet* mascots, pp.246, 11th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'03), 2003

[3] *Cost of Hard Drive Storage Space*, http://ns1758.ca/winch/winchest.html

[4] Parkhill D, *The challenge of the computer utility*, Addison-Wesley, Reading, 1966

[5] Vaquero L, Rodero-Merino L, Caceres J, Lindner M, *A break in the clouds: towards a cloud definition*, pp. 50-55 ACM SIG-COMM computer commuinications review, 2009

[6] *NIST Definition of Cloud Computing v15*, csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc

[7] XenSource Inc, *Xen*, www.xensource.com

[8] *Kernal Based Virtual Machine*, www.linux-kvm.org/page/Main_Page

[9] *VMWare ESX Server*, www.vmware.com/products/esx

[10]*Google App Engine*, code.google.com/appengine/docs/whatisgoogleappengine.html

[11] *Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model*, ISO/IEC International Organization for Standardization, 1994

[12] *The hybrid utility architecture*, Rough Type: Nicholas Carr's blog, http://www.roughtype.com May 31 2006

[13] *Virtualization Resource Chargeback*, http://www.vkernel.com/products/chargeback

[14] *Amazon EC2*, http://aws.amazon.com/ec2/

[14] *Windows Azure Platform*, http://www.microsoft.com/windowsazure/

[16] Chang F., Dean J. et al, *Bigtable: a distributed storage system for structured data*, ACM Transactions on Computer Systems (TOCS) Volume 26 Issue 2 Article 4, ACM New York, June 2008

[17] Eastlake D., Reagle J., Solo D., *(Extensible Markup Language) XML-Signature Syntax and Processing*, http://tools.ietf.org/html/rfc3275, March 2002

[18] Juric M.B. , Kezmah B., Hericko M., Rozman I., Vezocnik I., *Java RMI, RMI Tunneling and Web Services Comparison and Performance Analysis*, ACM SIGPLAN Notices Volume 39 Issue 5 pp. 58-65, May 2004

[19] Chang E., Dillon T., Hussain F.K., *Trust and reputation for service-oriented environments: technologies for building buisness intelligence and consumer confidence*, pag 53, John Wiley & Sons Ltd, 2006

[20] *OpSource Cloud Hosting*, http://www.opsource.net/Services/Cloud-Hosting

[21] *Microsoft Private Cloud*, http://www.microsoft.com/virtualization/en/us/private-cloud.aspx

[22] *Amazon Virtual Private Cloud*, http://aws.amazon.com/vpc/

[23] *Apache Turbine*, http://turbine.apache.org/

[24] *Apache Turbine*, http://velocity.apache.org/

[25] Gamma E., Helm R., Johnson R., Vlissides J.M., *Design Patterns: Elements of Reusable Object-Oriented Software*, cap 3,4 e 5, Addison-Wesley Professional, November 10, 1994

[26] *SOAP Version 1.2 W3C Recommendation*, http://www.w3.org/TR/2007/REC-soap12-part0-20070427/