

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

II Facoltà di Ingegneria

Corso di INGEGNERIA INFORMATICA

Laurea Triennale in SISTEMI DISTRIBUITI

Coordination as a Service (CaaS) in the Cloud

Relatore

Prof. Andrea Omicini

Candidato

Richiard Casadei

Correlatore

Ing. Stefano Mariani

Anno Accademico 2011/2012 - Sessione III

Ai miei genitori,
ai miei nonni,
ai miei zii e alla mia cuginetta,
alla mia ragazza
e a tutti quelli che mi hanno sempre sostenuto
e spronato a dare sempre di più.

Indice

Introduzione	9
1 Cloud computing	11
1 Attori coinvolti	12
2 Caratteristiche	12
3 Modelli di servizio	13
3.1 Infrastructure as a Service - IaaS	14
3.2 Platform as a Service - PaaS	15
3.3 Software as a Service - SaaS	15
4 Modelli di distribuzione	16
4.1 Public Cloud	16
4.2 Private Cloud	17
4.3 Community Cloud	17
4.4 Hybrid Cloud	17
2 Coordinazione come servizio (CaaS)	19
1 Sistemi coordinati	20
1.1 Architettura logica	21
2 Modello di coordinazione TuCSoN	23
2.1 Linguaggio di coordinazione	25
Primitive di base	26
Primitive Bulk	27
Primitive Uniform	28
Primitive di meta-coordinazione	28
Primitive Spawn	28
2.2 Architettura	29

	Spazio di coordinazione	29
	Mezzo di coordinazione	31
2.3	Agent Coordination Context	32
	ACC in TuCSoN	32
3	CaaS in the Cloud	35
1	TuCSoN in the Cloud	35
1.1	Scelte architetturali	36
	Welcome Cloud Entity	38
	Meta-Coordination Context	39
	Cloud Coordination Context	39
1.2	Naming dei centri di tuple	41
1.3	Naming delle risorse	42
1.4	Situatedness vs. Cloud	43
	Situatedness di TuCSoN in the Cloud	44
1.5	Primitive nel Cloud	45
1.6	Servizio pay-per-use	45
4	TuCSoN through Cloudify	47
1	Cloudify CLI	48
	Operazioni disponibili	48
2	Local Cloud	50
3	Cloudify Recipes	52
3.1	Application recipes	52
3.2	Service recipes	53
4	CLI di TuCSoN in the Cloud	54
4.1	Applet	54
4.2	Cloudify	55
	Recipe Nodo	56
	Recipe Tomcat	56
4.3	Installazione	56
4.4	Management	56
4.5	Problemi riscontrati	57
	Conclusioni e sviluppi futuri	59

Indice	7
A Sorgenti	63
Bibliografia	67

Introduzione

I modelli di coordinazione, come ad esempio il modello LINDA, sono stati concepiti nel contesto di sistemi chiusi, come applicazioni ad alte prestazioni parallele. In questo scenario tutti i soggetti coordinati sono conosciuti già in fase di progettazione, la coordinazione e il supporto sono concettualmente parte stessa dell'applicazione coordinata.

Con il progredire delle tecnologie, la crescente complessità degli attuali scenari applicativi, ha richiesto un nuovo approccio alla formalizzazione della coordinazione. I sistemi aperti e distribuiti, in genere ospitano una molteplicità di applicazioni che lavorano in modo concorrente, richiedendo una coordinazione basata su astrazioni che *(i)* persistano attraverso l'intero processo di progettazione e che *(ii)* forniscano servizi di coordinazione per applicazioni in un'infrastruttura condivisa sotto forma di supporti di coordinazione.

È naturale quindi pensare che, scenari come quelli che riguardano i sistemi distribuiti ed in particolar modo anche quelli che riguardano la loro nuova evoluzione e il modo di concepire l'erogazione di servizi IT e la gestione di risorse, chiamata Cloud Computing, richiedano una nozione diversa da quella di coordinazione come linguaggio, che nativamente è studiata e sviluppata principalmente nel contesto di sistemi paralleli. Infatti la facilità d'uso dei modelli di coordinazione, il disaccoppiamento temporale e territoriale che essi forniscono, sarebbero estremamente utili per fornire l'astrazione di strato middleware e mediatore per applicazioni basate su questo nuovo tipo di web services, il Cloud Computing, che necessitano di regole di coordinazione.

A tal proposito può essere quindi introdotta la nozione di coordinazione in-

tesa come servizio, denominabile anche sotto il nome di CaaS, che potrebbe essere a tutti gli effetti inserita tra le varie tipologie di servizi offerti dai provider di Cloud Computing e affiancherà quindi i suoi più diffusi e già noti servizi IaaS, PaaS e SaaS.

Il lavoro che segue è così organizzato:

- Nel primo capitolo si introducono le nozioni di base del Cloud computing, partendo dalla definizione, per poi trattare le principali caratteristiche, la suddivisione dei principali modelli e sistemi di distribuzione;
- Nel secondo capitolo si spiega cosa si intende per coordinazione come servizio, fornendone un contesto generale. Successivamente viene preso in considerazione il modello di coordinazione TuCSoN, del quale si vengono esaminate le principali caratteristiche e rappresentata l'architettura.
- Nel terzo capitolo si analizza la coordinazione come servizio nell'ambito del Cloud Computing e si spiega come il CaaS possa essere inserito tra i suoi vari servizi offerti e per concludere viene fornito un modello di TuCSoN in the Cloud con approfondimenti sulle entità coinvolte ed alcuni aspetti interessanti;
- Nel quarto capitolo viene proposto un piccolo caso di studio del CLI, il Command Line Interpreter, di TuCSoN in the Cloud realizzato come applet su pagina html e distribuito sul Cloud tramite la piattaforma PaaS Cloudify. Viene fornita una panoramica generale sulle caratteristiche di tale piattaforma e vengono spiegati in dettaglio i passaggi svolti per la realizzazione di tale caso di studio.

Infine sono delineate le conclusioni sul lavoro svolto e qualche possibile spunto per i prossimi tesisti che decideranno di approfondire questo argomento.

Capitolo 1

Cloud computing

Il NIST, National Institute of Standards and Technology, ovvero l'istituto di standardizzazione del settore delle tecnologie ha prodotto la seguente definizione di Cloud computing[1]:

Cloud computing is a model for enabling ubiquitous, 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 provisioned and released with minimal management effort or service provider interaction.

Da una certa prospettiva Il Cloud computing potrebbe essere visto come nulla di nuovo, in quanto alla base di esso troviamo nozioni, criteri e approcci già perfettamente stabiliti. Esso però cambia il modo di inventare, sviluppare, distribuire, scalare, aggiornare, gestire e pagare infrastrutture su cui vengono utilizzate le nostre applicazioni quindi, per questo, lo consideriamo nuovo. È un nuovo paradigma, un nuovo modo di concepire l'erogazione di servizi Internet-based e la relativa gestione delle risorse come server condivisi, software e dati su computer e altri dispositivi su richiesta. È un insieme di tecnologie informatiche che consentono l'utilizzo di risorse e servizi in remoto, erogati *on-demand*. Tali risorse e servizi possono essere forniti utilizzando server propri di un'azienda oppure da uno specifico provider, il quale offre e gestisce l'infrastruttura richiesta dall'utente. Proprietà molto importante è quella della virtualizzazione: possiamo addirittura considerare il Cloud come una sua naturale evoluzione e tramite la quale vogliamo rappresentare un

certo livello di astrazione nascondendo la complessità dell'infrastruttura alla base del sistema Cloud stesso.

Nel Cloud computing viene fornito un modello computazionale in cui l'hardware o il software sono erogati come servizi, permettendo all'utente di accedere ed usufruire delle funzionalità senza la necessità di possederne le conoscenze e le abilità operative. Questo modello è basato su un servizio di tipo *pay-per-use*, ovvero l'utente paga solamente l'effettivo utilizzo delle risorse e servizi che gli vengono messi a disposizione e resi accessibili tramite un browser oppure tramite delle API fornite dal provider.

1 Attori coinvolti

All'interno del contesto del Cloud computing distinguiamo tre principali tipi di attori coinvolti:

Cloud provider - è l'ente, l'organizzazione o l'ente che mette a disposizione le proprie risorse e fornisce una determinata tipologia di servizi Cloud all'utente finale;

Cloud consumer - è l'utente finale, la persona o l'organizzazione che richiede ad un Cloud provider uno specifico servizio in base alle proprie esigenze e con il quale mantiene delle relazioni di business;

Cloud carrier - è l'intermediario che fornisce la connettività tra il Cloud consumer e il Cloud provider.

2 Caratteristiche

Facendo riferimento al documento ufficiale del NIST[1] le principali caratteristiche del Cloud computing possono essere riassunte nei seguenti cinque punti:

On-demand self-service - L'utente può ottenere le risorse e i servizi richiesti come capacità di archiviazione o calcolo, in maniera automatica senza la richiesta di un intervento umano o un'interazione con ogni provider del servizio;

Broad network access - Le funzionalità sono disponibili in rete e vi si accede attraverso meccanismi standard che promuovono l'uso di piattaforme client eterogenee (ad esempio smartphone, tablet o computer);

Resource pooling - Le risorse di calcolo del provider sono raggruppate per servire più consumatori utilizzando un modello multi-tenant, con diverse risorse fisiche e virtuali assegnate dinamicamente e riassegnate in base alla domanda dei consumatori. Il cliente non ha alcun controllo o alcuna conoscenza sulla locazione esatta delle risorse messe a disposizione, ma può essere in grado di specificare la loro posizione ad un livello superiore di astrazione (ad esempio, paese, stato o data center);

Rapid elasticity - Le funzionalità possono essere fornite e rilasciate, in alcuni casi anche in modo automatico, per scalare rapidamente verso l'esterno o verso l'interno proporzionalmente alla domanda dell'utente. Per il consumatore, le funzionalità disponibili per il rilascio spesso sembrano essere illimitate e possono essere acquisite in qualsiasi quantità e in qualsiasi momento;

Measured service - I sistemi Cloud controllano automaticamente e ottimizzano l'utilizzo delle risorse, sfruttando un sistema di monitoraggio, tipicamente basato sul principio *pay-per-use*, e accounting ad livello di astrazione appropriato al tipo di servizio richiesto (ad esempio, la conservazione, l'elaborazione, la larghezza di banda, e gli account utenti attivi). L'utilizzo delle risorse può essere monitorato, controllato, e riportato, offrendo una garanzia di trasparenza sia nei confronti del provider, sia dell'utente utilizzatore del servizio.

3 Modelli di servizio

Il Cloud computing, e conseguentemente i suoi servizi, può essere offerto da un provider sotto forma di tre principali modelli (ne esistono altri ma non verranno trattati), ovvero tre diverse tipologie di infrastrutture:

- *Infrastructure as a Service* - IaaS;
- *Platform as a Service* - PaaS;
- *Software as a Service* - SaaS.

Queste tre tipologie hanno come modello base quello IaaS, quindi sono legati tra loro, ma possiedono comunque livelli di astrazione e complessità differenti, in termini sviluppo e realizzazione. Si distinguono anche per il livello di accesso fornito all'utente. Di seguito sono trattate in modo dettagliato le caratteristiche delle tre diverse infrastrutture.

3.1 Infrastructure as a Service - IaaS

È il servizio più vicino a ciò che forma il Cloud. Un servizio IaaS consiste in una virtualizzazione del hardware di un computer tradizionale. Attraverso questa infrastruttura il provider mette a disposizione dell'utente tutte le potenzialità e la flessibilità di un computer fisico che consistono in: capacità di calcolo, di archiviazione, network e tutte le altre risorse hardware sulle quali sarà possibile eseguire applicazioni utente o interi sistemi operativi. Il fornitore a tal proposito mette a disposizione la documentazione necessaria sull'infrastruttura, affinché la futura piattaforma software dell'utente possa utilizzarne le risorse.

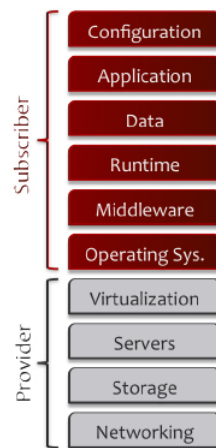


Figura 1.1: Diagramma dell'Infrastructure as a Service

Il consumatore non gestisce o controlla direttamente l'infrastruttura cloud di base ma ha il controllo su sistemi operativi, storage e applicazioni distribuite, ed eventualmente un controllo limitato delle componenti di rete (ad esempio firewall).

3.2 Platform as a Service - PaaS

Un servizio PaaS consiste in una virtualizzazione di una piattaforma completa, utile allo sviluppo di applicazioni. In questo caso l'utente non deve occuparsi dell'infrastruttura attraverso la quale è realizzata una piattaforma, in quanto tali problematiche sono già state affrontate dal provider del servizio PaaS, il quale ha messo a disposizione l'hardware, il sistema operativo e le relative librerie. L'utente avrà l'onere di preoccuparsi di creare e gestire le proprie applicazioni. Quest'ultimo, in questo caso, quindi ha meno oneri, ma nello stesso tempo anche meno flessibilità: infatti il fornitore ha sicuramente optato per delle scelte riguardo alla piattaforma su cui offrire il servizio PaaS, a cui l'utente si dovrà adattare per sviluppare la sua applicazione. Tipicamente sono piattaforme che rendono disponibili strumenti di sviluppo, di creazione di interfacce web e servizi web, storage e database integration.

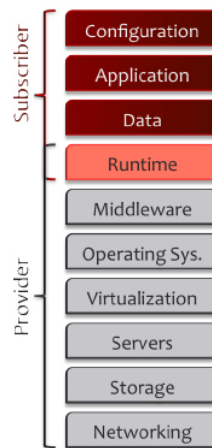


Figura 1.2: Diagramma del Platform as a Service

3.3 Software as a Service - SaaS

È un servizio software realizzato su una piattaforma che, a sua volta, poggia su una specifica infrastruttura. Tipicamente è un software completo realizzato sulle specifiche richieste dell'utente, e spesso rimane personalizzabile e configurabile dall'utente stesso. Il software in questo tipo di servizio Cloud è progettato seguendo un'architettura multi-tenant e dal punto di vista degli utenti si ha una completa trasparenza su dove il software è ospitato, in che

tipo di linguaggio di programmazione è stato scritto e su quale sistema operativo gira. È il servizio di più alto livello che possa essere offerto all'utente finale. L'accesso e l'utilizzo a tale servizio, possono essere effettuati da vari dispositivi tramite browser Web o da una specifica interfaccia del programma. Viene quindi eliminato il bisogno di installare ed eseguire il software sulle macchine fisiche dell'utente.

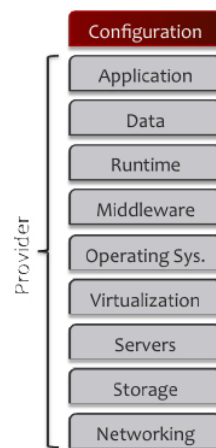


Figura 1.3: Diagramma del Software as a Service

4 Modelli di distribuzione

In base alla locazione fisica dell'infrastruttura con cui è offerto un servizio Cloud possiamo distinguere quattro tipologie di Cloud, definiti dal NIST deployment models[1]:

- *Public Cloud*
- *Private Cloud*
- *Community Cloud*
- *Hybrid Cloud*

4.1 Public Cloud

In questo modello, l'infrastruttura Cloud, è fornita per l'utilizzo aperto ad un generico pubblico. Essa e le relative applicazioni come dispositivi di storage e le varie risorse possono essere possedute, gestite e mantenute da un terzo,

ad esempio da un'azienda (cloud provider), da un'organizzazione accademica o governativa, o una combinazione di queste.

4.2 Private Cloud

In questo modello l'infrastruttura Cloud è fornita per un suo uso esclusivo ad un unico organismo che può comprendere molteplici utenti (ad esempio, unità di business). Essa può essere di proprietà, e gestito da un'organizzazione privata, un terzo, o una combinazione di essi.

4.3 Community Cloud

In questo modello l'infrastruttura Cloud è fornita per l'uso esclusivo da parte di una determinata comunità di consumatori o da parte di varie organizzazioni, facenti però parte della stessa comunità. Come nel caso del modello privato, essa può essere di proprietà, mantenuta e gestita da una o più delle organizzazioni della comunità, da un terzo, o una combinazione di essi.

4.4 Hybrid Cloud

In questo modello l'infrastruttura Cloud è una composizione di due o più infrastrutture Cloud distinte (private, community, o public) legate tra loro da tecnologie condivise o private, che consentono la condivisione di dati e la portabilità delle applicazioni.

Capitolo 2

Coordinazione come servizio (CaaS)

In contrasto con la nozione di coordinazione come linguaggio, naturalmente nata nel contesto di sistemi chiusi e paralleli, facendo riferimento all'articolo "*Coordination as a Service*"[4] scritto dai prof. Mirko Viroli e Andrea Omicini, in questa sezione verrà presentato un contesto formale e generale sulla percezione di coordinazione intesa come servizio.

In tale contesto, i mezzi di coordinazione sono promossi ad entità di prima classe dei sistemi di coordinazione, e quindi rappresentabili tramite una esplicita astrazione, con l'obiettivo di sostenere la progettazione di modelli di coordinazione fino alla distribuzione di infrastrutture di coordinazione. Un modello di coordinazione può essere costruttivamente definito descrivendo (i) le sue *entità di coordinazione*, tipi di agenti soggetti alla coordinazione, (ii) i *mezzi di coordinazione*, componenti e astrazioni che rendono possibili le comunicazioni tra gli agenti, e (iii) le *leggi di coordinazione*, che descrivono come gli agenti si coordinano attraverso i mezzi utilizzando alcune primitive di coordinazione[5]. Questa definizione è stata sviluppata intorno al concetto di *mezzo di coordinazione*, il quale rappresenta un'entità di prima classe dei sistemi che adottano modelli di coordinazione e che fornisce una netta separazione nei confronti degli agenti, entità attive ed indipendenti, ed identifica il luogo concettuale dove avviene la coordinazione. Un modello di coordinazione inoltre deve trattare gli aspetti che riguardano la creazione e la distruzione

degli agenti, la comunicazione tra di essi, la loro distribuzione spaziale e come avviene la loro sincronizzazione e la distribuzione delle loro azioni nel tempo. La fornitura di questi modelli di coordinazione può essere effettuata da un progettista software attraverso un linguaggio di coordinazione o tramite un'architettura di coordinazione, come l'architettura client-server, le pipeline software, o le blackboard: componenti software disposti in particolari e ben definiti insiemi di software che mettono in atto specifici protocolli di cooperazione[6]. All'interno di un'architettura di coordinazione la nozione di mezzo di coordinazione si piazza come modello per le astrazioni che costituiranno le architetture run-time a supporto della coordinazione stessa, più precisamente rappresenterà l'infrastruttura di coordinazione.

La comunicazione tra le entità coordinate e i mezzi di coordinazione ha luogo attraverso uno spazio di interazione, in cui gli eventi di comunicazione si verificano come un effetto di qualche entità generatrice di un output (fase di comunicazione) e sono consumati nel momento in cui qualche entità si apre allo spazio di interazione (fase di ascolto). Si suppone che i mezzi di coordinazione siano delle astrazioni reattive, dove le attività interne sono erogate solamente grazie alla fase di ascolto, e queste sono adatte per fornire una descrizione operativa in termini di transazioni di sistema, in cui le azioni osservabili sono eventi di consumo e produzione.

1 Sistemi coordinati

Un sistema che adotta un modello di coordinazione è chiamato sistema coordinato. Esso è costituito da tre spazi separati, lo *spazio coordinato*, lo *spazio di interazione* e lo *spazio di coordinazione*, che includono tutte le entità e le astrazioni che ricoprono ruoli differenti nel processo di coordinazione, ciascuna caratterizzata da uno specifico comportamento come astrazione di run-time.

1.1 Architettura logica

L'insieme delle tre parti che compongono un sistema coordinato ne rappresenta l'architettura logica.

Spazio coordinato - rappresenta la parte del sistema che include tutte le entità di coordinazione. Dalla comunicazione del ACM (Association for Computing Machinery)[8] che ha stilato la terminologia standard per i linguaggi di coordinazione, le entità coordinate sono quelle attività le cui interazioni sono governate e regolate da un modello di coordinazione. Il comportamento di ogni entità coordinata può essere inteso in termini di emissione di richieste di coordinazione, come l'apertura stessa alla ricezione delle risposte di coordinazione o l'esecuzione di calcoli interni. Ciò permette di affermare che la coordinazione si svolgerà come un servizio interattivo fatto di richieste e risposte, scambiate con l'ambiente circostante, e l'interazione tra le entità può essere solo indiretta in quanto deve essere eseguita e mediata dall'ambiente tramite un'attività di coordinazione. Lo spazio coordinato è quindi generalmente composto da un insieme di entità coordinate, ciascuna con la propria identità, che viene utilizzato dalle attività di coordinazione per tenere traccia degli agenti che hanno emesso delle richieste e di coloro che devono riceverne le risposte. Per affrontare l'apertura e la scalabilità del sistema, caratteristiche fondamentali dei sistemi distribuiti, tale insieme non deve essere fissato a priori[4].

Spazio di interazione - rappresenta l'ambiente per le entità coordinate e ha il ruolo di gestire le loro richieste e le relative risposte. Il suo obiettivo principale è quello di materializzare o reificare le azioni eseguite dalle entità e le astrazioni dei sistemi coordinati in eventi di comunicazione, i quali potranno esser successivamente consumati da altre entità o astrazioni. Sappiamo che gli eventi (azioni) da parte di un'entità possono essere di due tipi: eventi di richiesta ed eventi di risposta. Nel primo caso essi vengono emessi attraverso lo spazio di interazione, mentre nel secondo, gli eventi di risposta vengono consumati man mano che si verificano nello spazio di interazione e ogni evento è diretto esplicitamente ad una data entità di coordinazione, ad esempio la stessa entità che

ha emesso una determinata richiesta e successivamente si è aperta alla ricezione di tale risposta. Il processo di coordinazione quindi consiste in un consumo di richieste e una produzione di risposte tramite lo spazio di interazione[4].

Spazio di coordinazione - rappresenta lo spazio dove vengono effettuati il consumo di eventi di richiesta e la produzione di eventi di risposta, insieme alle attività di coordinazione. È costituito da un numero di mezzi di coordinazione, a loro volta concettualmente associabili ad un sottoinsieme di entità coordinate. Un evento può essere consumato da un mezzo di coordinazione solo secondo una certa condizione di matching, che verificherà la corrispondenza delle entità coordinate, del contenuto della richiesta e lo stato corrente del mezzo di coordinazione. Avvenuto il consumo dell'evento, il mezzo di coordinazione può produrre uno o più eventi, sia di risposta sia di richiesta, che si materializzano poi nello spazio di interazione. Quindi, come per le entità coordinate, il comportamento del mezzo di coordinazione è inteso in termini di consumo di richieste, di emissioni di risposte e di esecuzioni di azioni "silenziose" rappresentanti la computazione interna delle attività di coordinazione[4].

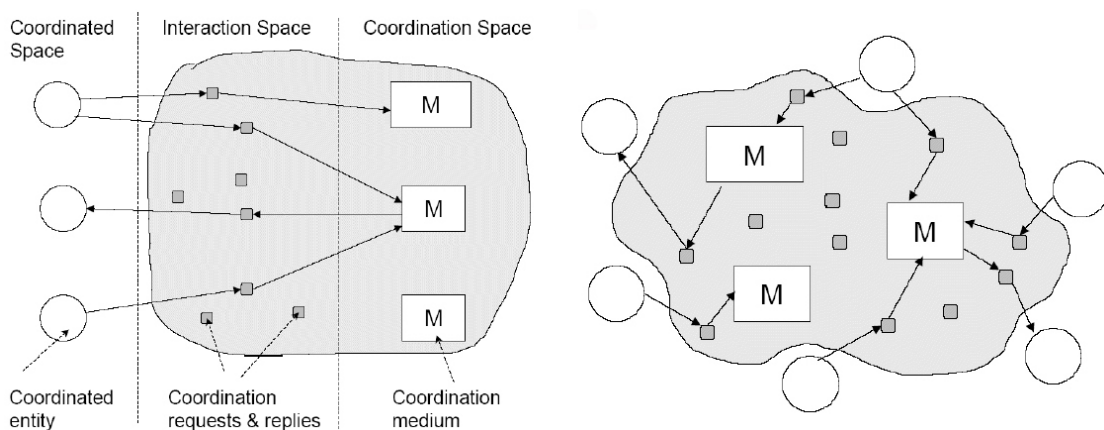


Figura 2.1: CaaS: Architettura logica (a sinistra) e topologica (a destra)

2 Modello di coordinazione TuCSoN

TuCSoN (Tuple Centres Spread over the Network)¹ è un modello e una tecnologia per la coordinazione di processi distribuiti e di agenti autonomi, intelligenti e mobili[9]. Alla base di esso troviamo delle astrazioni indipendenti di coordinazione e comunicazione dette centri di tuple, ovvero spazi di tuple avanzati il cui comportamento descrive leggi specifiche di coordinazione, programmabili tramite il linguaggio specifico per le reaction ReSpecT. In questo modello lo spazio di interazione è rappresentato dai centri di tuple distribuiti sui nodi attraverso la rete (Internet) ed utilizzati dagli agenti per interagire e coordinarsi con altri agenti remoti o con framework di esecuzione locali.

Le entità che caratterizzano un sistema TuCSoN sono[10]:

- *Agenti* TuCSoN - sono entità pro-attive ed intelligenti che interagiscono tra di loro scambiando tuple attraverso i centri di tuple mediante le primitive di coordinazione offerte da TuCSoN. Hanno anche la caratteristica di essere mobili quindi non sono associati in modo permanente ad un particolare device e sono distribuiti nella rete. All'interno di un sistema coordinato rappresentano le entità coordinabili;
- *Centri di tuple* ReSpecT - ricoprono il ruolo di mezzo di coordinazione, forniscono lo spazio condiviso per la comunicazione tuple-based (spazio di tuple), insieme allo spazio di comportamento programmabile per la coordinazione (spazio di specifica). Essi appartengono ai nodi TuCSoN e sono associati ad uno specifico device, quindi la mobilità dei centri di tuple dipende dal device che lo ospita;
- *Nodi* TuCSoN - rappresentano l'astrazione topologica di base che ospita al suo interno i centri di tuple, sono i loro "container".

Ad ogni nodo, centro di tuple e agente è associato un identificatore in maniera univoca all'interno di un sistema TuCSoN.

Ogni nodo è identificato dalla coppia $\langle NetworkId, PortNo \rangle$, dove *NetworkId* corrisponde all'indirizzo IP o alla entry DNS del device ospitante il nodo e *PortNo* è il numero di porta su cui il servizio di coordinazione TuCSoN

¹Sito: <http://tucson.apice.unibo.it>

è in ascolto di un'eventuale invocazione dell'esecuzione di un'operazione di coordinazione. In TuCSoN un singolo device può ospitare una molteplicità di nodi e nella sua sintassi astratta adottata per rappresentare l'identificatore un nodo sarà definito da `netid:portno`.

Un nome ammissibile per un centro di tuple è un qualsiasi termine della logica di prim'ordine senza variabili (*ground*) e viene identificato univocamente da `tname @ netid:portno` dove la coppia `netid:portno` rappresenta il nodo a cui appartiene il centro di tuple.

Ciascun agente, infine, nel momento in cui entra in un sistema TuCSoN è identificato da un nome comune `aname`, anch'esso un termine della logica *ground*, e da un UUID (*Universally Unique Identifier*) assegnatogli dal middleware in modo da distinguerlo da qualsiasi altro agente del sistema. Il nome completo è quindi formato da `aname:uuid`.

Facendo riferimento ai concetti e alla terminologia introdotti in [7], le principali e più rilevanti caratteristiche di TuCSoN riguardano:

Il *linguaggio di coordinazione* TuCSoN - che permette agli agenti di interagire con i centri di tuple mediante l'esecuzione di operazioni di coordinazione costituite da alcune primitive di comunicazione;

Lo *spazio di coordinazione* TuCSoN - con la sua duplice interpretazione sia come spazio di interazione globale costituito da astrazioni di comunicazione uniche e ben note, sia come collezione di spazi di interazione locali;

Il *mezzo di coordinazione* TuCSoN - che rappresenta le astrazioni di comunicazione il cui comportamento può essere definito in modo da incorporare le leggi di coordinamento globale.

Lo spazio di coordinazione sostiene efficacemente il duplice ruolo degli agenti Internet, come entità *networkaware* che individuano e accedono ai dati e alle risorse Internet, e come entità *roaming* che trasferiscono la loro esecuzione su un sito in cui essi interagiscono con le risorse locali. Il mezzo di coordinazione arricchisce il modello di coordinazione, fondamentalmente *data-oriented*, con la flessibilità e il controllo necessari per affrontare la complessità delle applicazioni Internet.

2.1 Linguaggio di coordinazione

TuCSoN fornisce un linguaggio di coordinazione, definito da un insieme di primitive di coordinazione, per permettere agli agenti di interagire con i centri di tuple. Tali primitive possiedono la stessa semantica delle primitive LINDA e consentono agli agenti di leggere, scrivere e consumare tuple nel centro di tuple e di sincronizzarsi con esso. Il linguaggio di comunicazione è costituito da linguaggi di tuple e template di tuple logic-based, in quanto il mezzo di coordinazione TuCSoN di default è il centro di tuple ReSpecT (anch'esso logic-based).

Ogni operazione di coordinazione è descritta da una fase di invocazione (*invocation*), dove avviene l'invio della richiesta di esecuzione da parte dell'agente contenente tutte le informazioni dell'invocazione verso il centro di tuple designato, e da una fase di completamento (*completion*), durante la quale il risultato dell'operazione richiesta dal centro di tuple ritorna all'agente richiedente includendo tutte le informazioni riguardante l'operazione eseguita. La sintassi astratta di un'operazione *op* invocata da un agente su un dato centro di tuple è:

$$\text{tcid ? op}$$

dove *tcid* è l'identificatore del centro di tuple. Siccome *tcid* può essere sia un nome assoluto sia un nome relativo del centro di tuple, gli agenti possono adottare due differenti forme di invocazione delle primitive, rispettivamente quella network e quella locale.

La forma di comunicazione network *tname @ netdid : portno ? op* è utilizzata dagli agenti quando essi agiscono come entità network-aware indicando il centro di tuple attraverso il suo nome assoluto nello spazio di interazione TuCSoN globale. La forma locale di comunicazione invece è rappresentata dalla forma generale *tcid ? op*, con riferimento all'implementazione locale del centro di tuple del nodo di esecuzione dello specifico agente, e viene utilizzata dagli agenti quando si comportano come componenti locali del loro ambiente di hosting.

Di seguito sono elencate le varie primitive di coordinazione presenti nella più recente formalizzazione del modello TuCSoN[10].

Primitive di base

Il linguaggio di coordinazione TuCSoN fornisce le seguenti primitive di coordinazione per costruire operazioni di coordinazione:

- `out`, `rd`, `in`
- `rdp`, `inp`
- `no`, `nop`
- `get`, `set`

`out(Tuple)` - scrive la tupla *Tuple* nello spazio di tuple specificato, dopo l'esecuzione con successo dell'operazione la tupla è ritornata come completamento.

`rd(TupleTemplate)` - ricerca se una tupla ha una corrispondenza col *TupleTemplate* nello spazio di tuple specificato, se una tupla *Tuple* fa il matching quando l'operazione è servita, il successo dell'esecuzione ritorna *Tuple*, altrimenti, l'esecuzione dell'operazione è sospesa, essa verrà ripresa e completata con successo quando sarà trovata una tupla con una corrispondenza nello spazio di tuple e poi la tupla ritornata.

`in(TupleTemplate)` - ricerca una tupla che abbia una corrispondenza col *TupleTemplate* nello spazio di tuple specificato, se una tupla *Tuple* fa il matching, il successo dell'esecuzione rimuove *Tuple* dallo spazio di tuple e poi la ritorna, in caso contrario, l'esecuzione dell'operazione viene sospesa, verrà ripresa e completata con successo quando sarà trovata una tupla con una corrispondenza nello spazio di tuple, la tupla verrà rimossa e poi ritornata.

`rdp(TupleTemplate)` - ricerca se una tupla ha una corrispondenza col *TupleTemplate* nello spazio di tuple specificato, se una tupla *Tuple* fa il matching quando l'operazione è servita, il successo dell'esecuzione ritorna *Tuple*, altrimenti, l'esecuzione dell'operazione fallisce e sarà ritornato il *TupleTemplate*.

`inp(TupleTemplate)` - ricerca una tupla che abbia una corrispondenza col *TupleTemplate* nello spazio di tuple specificato, se una tupla *Tuple* fa il matching, il successo dell'esecuzione rimuove *Tuple* dallo spazio di tuple e poi lo ritorna, in caso contrario, l'esecuzione dell'operazione fallisce e sarà ritornato il *TupleTemplate*.

no(TupleTemplate) - ricerca una tupla che abbia una corrispondenza col *TupleTemplate* nello spazio di tuple. Nel caso in cui non vi sia alcuna corrispondenza, il successo dell'operazione ritorna il *TupleTemplate*, altrimenti, l'esecuzione viene sospesa, verrà poi ripresa e terminata con successo nel momento in cui il matching non sarà trovato per nessuna tupla dello spazio di tuple, e successivamente il *TupleTemplate* verrà ritornato.

nop(TupleTemplate) - controlla se una tupla ha una corrispondenza col *TupleTemplate* nello spazio di tuple. Nel caso in cui non vi sia alcuna corrispondenza, l'operazione ha successo e il *TupleTemplate* viene ritornato, in caso contrario se una tupla *Tuple* fa il matching, l'esecuzione dell'operazione fallisce e sarà ritornato il *Tuple*.

get - legge tutte le *Tuple* dallo spazio di tuple e le ritorna sotto forma di lista, se nello spazio di tuple non è presente nessuna tupla nel momento dell'esecuzione, l'operazione ha comunque successo e viene ritornata una lista vuota;

set(Tuples) - riscrive lo spazio di tuple con la lista di *Tuples* passati e quando l'esecuzione dell'operazione è terminata, la lista di *Tuples* viene ritornata.

Primitive Bulk

Le primitive bulk sono state introdotte per ottenere un significativo aumento di efficienza nella gestione di operazioni di coordinazione con più di una tupla. Invece di ritornare una sola singola tupla, le operazioni bulk ritornano la lista delle tuple che posseggono la corrispondenza richiesta e nel caso in cui non fossero presenti, esse ritornano una lista di tuple vuota. Le primitive bulk quindi hanno sempre successo.

Il linguaggio di coordinazione TuCSon fornisce le seguenti primitive bulk per costruire operazioni di coordinazione:

- `out_all`
- `rd_all`
- `in_all`
- `no_all`

Primitive Uniform

Le primitive Uniform sono utilizzate per iniettare un meccanismo probabilistico all'interno della coordinazione, così da promuovere un comportamento stocastico nei sistemi coordinati. Tali primitive aggiungono alle primitive un uniforme e probabilistico non-determinismo. La tupla ritornata da una primitiva uniform è scelta in modo non-deterministico su tutte le tuple che hanno avuto una corrispondenza con il template ed è eseguita con una probabilità uniforme.

Il linguaggio di coordinazione TuCSoN fornisce le seguenti primitive bulk per costruire operazioni di coordinazione:

- `urd`, `uin`
- `urdp`, `uinp`
- `uno`, `unop`

Primitive di meta-coordinazione

TuCSoN definisce nove primitive di meta-coordinazione per permettere agli agenti di leggere, scrivere, consumare delle tuple di specifica ReSpecT, interagire e sincronizzarsi nello spazio di tuple. Le meta-primitive hanno una perfetta corrispondenza con le primitive di coordinazione di base presentate precedentemente, queste però permettono un accesso uniforme sia allo spazio di tuple ordinario (come quelle di base) sia allo spazio di specifica in un centro di tuple TuCSoN. Tali primitive sono:

- `out_s`, `rd_s`, `in_s`
- `rdp_s`, `inp_s`
- `no_s`, `nop_s`
- `get_s`, `set_s`

Primitive Spawn

TuCSoN fornisce le primitive spawn per permettere attività computazionali complesse relative alla coordinazione. L'esecuzione dello *spawn* è locale allo spazio di tuple dove esso è invocato. Corrispondentemente, il codice dell'agente dovrebbe essere locale allo stesso nodo ospitante il centro di tuple e il

codice può eseguire primitive di coordinazione TuCSoN ma solo nello stesso *spawning* centro di tuple. La semantica delle primitive spawn non è sospensiva: l'invocazione di una operazione di spawn innesca una attività computazionale concorrente e il risultato è ritornato al chiamante nel momento in cui questa attività inizia.

2.2 Architettura

Un sistema TuCSoN è caratterizzato prima di tutto dalla collezione (possibilmente distribuita) dei nodi ospitanti un servizio TuCSoN[10]. Un nodo offre i propri servizi tramite un host collegato alla rete e alla porta di rete in cui il servizio TuCSoN si mette in ascolto di possibili richieste in arrivo. Diversi nodi TuCSoN possono funzionare sullo stesso device, ognuno in ascolto su una differente porta di rete. In TuCSoN il numero di porta di default è 20504 quindi se un agente invoca un'operazione del tipo

$$\mathbf{tname} @ \mathbf{netid} ? \mathbf{op}$$

senza specificare il numero di porta `portno` ciò significa che l'agente intende invocare l'operazione `op` sul centro di tuple `tname` del nodo di default `node netid : 20504` ospitato dal networked device `netid`. Chiaramente vi è anche la possibilità di utilizzare altre porte per i nodi TuCSoN.

All'interno di ogni nodo TuCSoN è presente una collezione di centri di tuple resi ammissibili attraverso un nome `tname` valido. Ogni operazione di coordinazione può essere invocata su ogni centro di tuple appartenente ad un qualsiasi nodo TuCSoN, quindi agli agenti è fornito un completo spazio di coordinazione. Ogni nodo definisce inoltre un centro di tuple di default, chiamato `default`, il quale risponderà qualsiasi invocazione di un'operazione ricevuta dal nodo senza che non sia specificato il centro di tuple destinatario (omettendo quindi `tname` nel formalismo delle operazioni via network).

Spazio di coordinazione

Lo spazio di coordinazione TuCSoN è costituito da una moltitudine di astrazioni di comunicazione indipendenti chiamate centri di tuple. Come rappresentato in Figura 2.2 ogni nodo fornisce una propria versione del TuCSoN

media space, ovvero l'insieme degli identificatori dei centri di tuple ammissibili, implementando virtualmente ogni centro di tuple come un servizio internet.

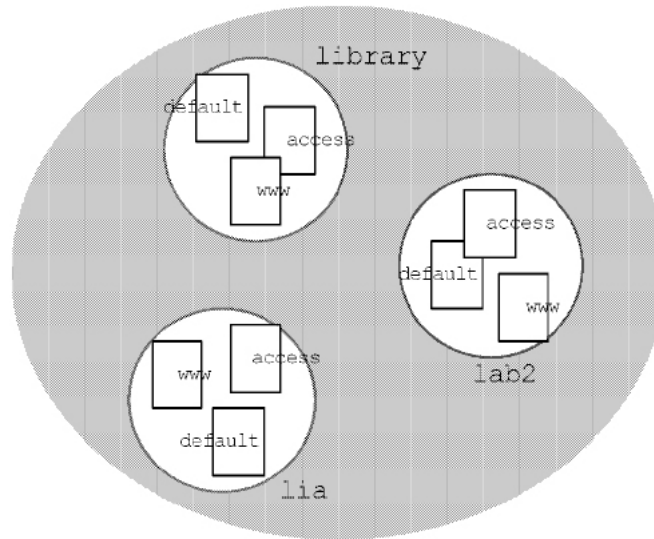


Figura 2.2: Tre nodi TuCSoS {lia, library, lab2}.deis.unibo.it, ognuno dei quali implementa la propria versione locale dello stesso media space {default, access, www}.

Lo spazio di coordinazione TuCSoS può essere visto sia come uno spazio di interazione globale, definito dalla collezione di tutti i centri di tuple disponibili nella rete, ospitati da un nodo ed indentificati dal loro nome assoluto, sia come una collezione di spazi di interazione locali, definiti dallo stesso insieme di identificatori dei nomi di un singolo hosting device[7] a cui si può fare riferimento evitando di specificare l'indirizzo di rete.

Un agente in esecuzione, tramite l'operazione

```
tname : portno ? op
```

invocherà un'operazione op sul centro di tuple tname localmente fornita dal nodo netid : portno, mentre per poter richiedere un'operazione a livello globale l'invocazione dovrà esser più specifica da parte dell'agente e dovrà essere della seguente forma:

```
tname @ netid : portno ? op
```

Mezzo di coordinazione

TuCSoN sfrutta i centri tuple come mezzi di coordinazione. Un centro di tuple è un'astrazione di comunicazione percepita dalle entità interagenti come uno spazio tupla standard, ma il cui comportamento in risposta ad eventi di comunicazione può essere definito in modo da incorporare le leggi di coordinazione.

Il comportamento di ogni singolo centro di tuple può essere definito separatamente e indipendentemente da qualsiasi altro centro di tuple in base ai specifici compiti di coordinazione scelti. Il comportamento di uno spazio di tuple è naturalmente definito come la transizione di stato osservabile dopo un evento di comunicazione. La definizione di un nuovo comportamento per un centro di tuple fondamentalmente equivale a specificare una nuova transizione di stato in risposta ad un evento di comunicazione standard. Ciò si ottiene consentendo la definizione di reazioni (*reaction*) di comunicazione attraverso un linguaggio di specifica di reazione[11].

Più precisamente, un linguaggio di reazione è associabile a qualsiasi delle primitive di base di TuCSoN (out, in, rd, inp, rdp) per specificare attività computazionali, queste saranno chiamate *reaction*. Esse sono definite come un insieme di operazioni non bloccanti. Una *reaction* eseguita con successo può atomicamente produrre effetti sullo stato del centro di tuple, mentre il fallimento di una *reaction* non produce alcun risultato. Ogni reazione può liberamente accedere e modificare le informazioni raccolte in una tupla di un centro di tuple, e può accedere a tutte le informazioni relative all'evento di comunicazione di attivazione. Ogni evento di comunicazione può in linea di principio innescare una molteplicità di reazioni, tuttavia, tutte le reazioni eseguite come conseguenza di un singolo evento di comunicazione sono tutte eseguite in una sola transizione di stato del centro di tuple prima di una qualsiasi altra esecuzione di eventi di comunicazione innescati da un agente. Di conseguenza, dal punto di vista degli agenti il risultato della chiamata di una primitiva è la somma degli effetti della stessa primitiva e di tutte le reazioni che essa ha innescato, complessivamente percepito come una unica transizione di stato del centro di tuple. Tali nozioni sono adottate nei centri di tuple di TuCSoN attraverso il modello ReSpecT.

2.3 Agent Coordination Context

La nozione di *Agent Coordination Context (ACC)* può essere vista come un mezzo per modellare e formare ambienti in sistemi ad agenti[12]. Ricopre lo stesso ruolo dell'interfaccia nei sistemi ad oggetti, fornendo una disciplina per l'interazione e qualche modello per la gestione di controllo nei sistemi ad agenti. Un ACC fornisce un disaccoppiamento tra gli agenti e il loro ambiente, in modo che un agente può essere progettato e sviluppato indipendentemente dagli altri agenti, risorse e servizi che popolano lo stesso multilayer system (MAS). Dal punto di vista dell'agente, esso dovrebbe fornire agli agenti stessi aspetti di interazione e comunicazione tramite un'astrazione che racchiuda anche nozioni di località nello spazio e nel tempo visti gli scenari applicativi di oggi e dovrebbe anche consentire agli agenti di percepire lo spazio in cui interagiscono in termini di effetti delle loro azioni e comunicazioni ed eventualmente, influenzare l'ambiente per raggiungere i propri obiettivi. Dal punto di vista del progettista un ACC dovrebbe fornire un quadro complessivo per esprimere l'interazione all'interno di un MAS, definendo l'insieme delle interazioni ammissibili, gli agenti coinvolti e l'ambiente MAS e dovrebbe anche incapsulare le regole per le applicazioni che disciplinano i sistemi ad agenti, e mediare le interazioni tra gli agenti e l'ambiente.

ACC in TuCSoN

Nel modello di coordinazione TuCSoN un *Agent Coordination Context* fornisce agli agenti una visione del loro spazio di coordinazione come collezione dei centri di tuple del sistema. Assieme ai centri di tuple, gli ACC sono le astrazioni che in fase di esecuzione permettono a TuCSoN di gestire in modo uniforme i problemi di coordinamento, organizzazione e sicurezza. L'ACC ricopre il ruolo di mediatore dell'interazione in quanto un agente non può invocare un'operazione direttamente sul nodo, ma esso dovrà utilizzare "l'interfaccia" delle operazioni disponibili fornitagli dall'ACC. Dualmente il sistema di nodi interagisce con gli agenti solo tramite l'ACC e nel momento in cui uno degli agenti rilascia il suo contesto, dal punto di vista del sistema esso smette di esistere. Essendo il mediatore dell'interazione tramite la sua configurazione un Agent Coordination Context permette di inserire un insieme di

regole di base per specificare e/o limitare le operazioni sul sistema. Infatti un ACC non è vincolato a fornire ad un agente una visione dell'intero spazio di interazione costituito da tutto l'insieme dei nodi disponibili e dei rispettivi centri di tuple, ma può bloccare totalmente l'accesso verso alcuni centri di tuple, impedire solo certe operazioni sugli altri centri o limitare l'accesso o la semplice lettura delle informazioni contenute in certi centri di tuple. Tutte queste regole possono essere iniettate nell'ACC attraverso una configurazione appropriata, a seguito di una negoziazione con il sistema. L'Agent Coordination Context rappresenta in questo senso il contratto tra l'agente e il sistema TuCSoS.

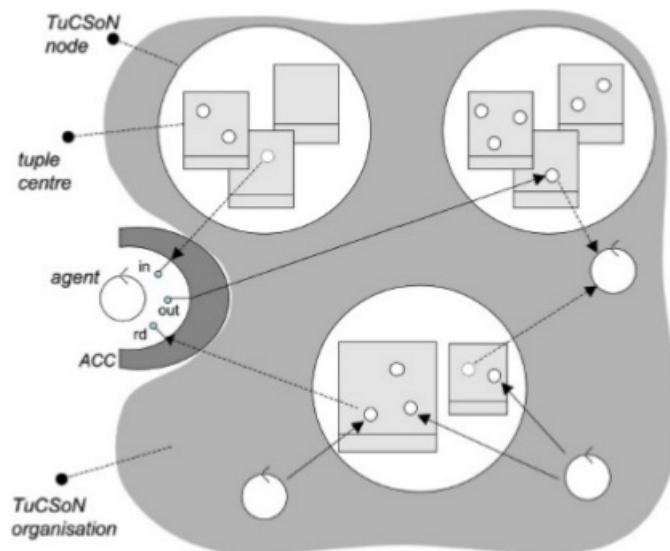


Figura 2.3: Architettura di un nodo TuCSoS in cui è presente anche l'ACC

Capitolo 3

CaaS in the Cloud

Il Coordination as a Service (CaaS) lo si può descrivere come un insieme di servizi che possono essere utilizzati attraverso il Cloud, con lo scopo di fornire coordinazione a utenti, processi o software senza dover installare piattaforme dedicate contenenti modelli di coordinazione sulle macchine fisiche dell'utente. La coordinazione sarà eseguita all'interno della "nuvola" sfruttando le risorse e i meccanismi di coordinazione forniti da un provider. Dualmente ai sistemi chiusi o ai sistemi distribuiti, anche nel Cloud Computing la coordinazione può essere eseguita attraverso diverse strategie e modelli, ma indipendentemente da ciò ogni servizio sarà del tipo *pay-per-use* e verrà erogato all'utente in modo *on-demand*. Quindi è lecito affermare che il CaaS può essere inserito tra i più noti servizi di Cloud Computing come l'IaaS, il PaaS e il SaaS.

1 TuCSoN in the Cloud

Nella seguente sezione verrà analizzato il modello di coordinazione TuCSoN nell'ambito del Cloud Computing. Si cercherà di analizzare ogni nozione principale come nodo, centro di tuple e Agent Coordination Context, saranno inoltre introdotte altre entità ed astrazioni che ci permetteranno di costruire un servizio CaaS in the Cloud che abbia TuCSoN come strumento centrale per la realizzazione della coordinazione. Tale servizio dovrà essere accessibile tramite un browser web nel quale l'utente dovrà, nel caso di un primo utilizzo del servizio, effettuare una registrazione specificando il tipo di contratto

che vorrà sottoscrivere con il provider, in caso di registrazione già effettuata, un login sul sito per poter accedere alla sezione in cui interagire col servizio stesso.

Di seguito è fornito un primo modello astratto comprendente tutte le nuove entità, che verranno poi spiegate nelle sotto sezioni, e da quello si partirà per realizzare un vero e proprio modello architetturale.

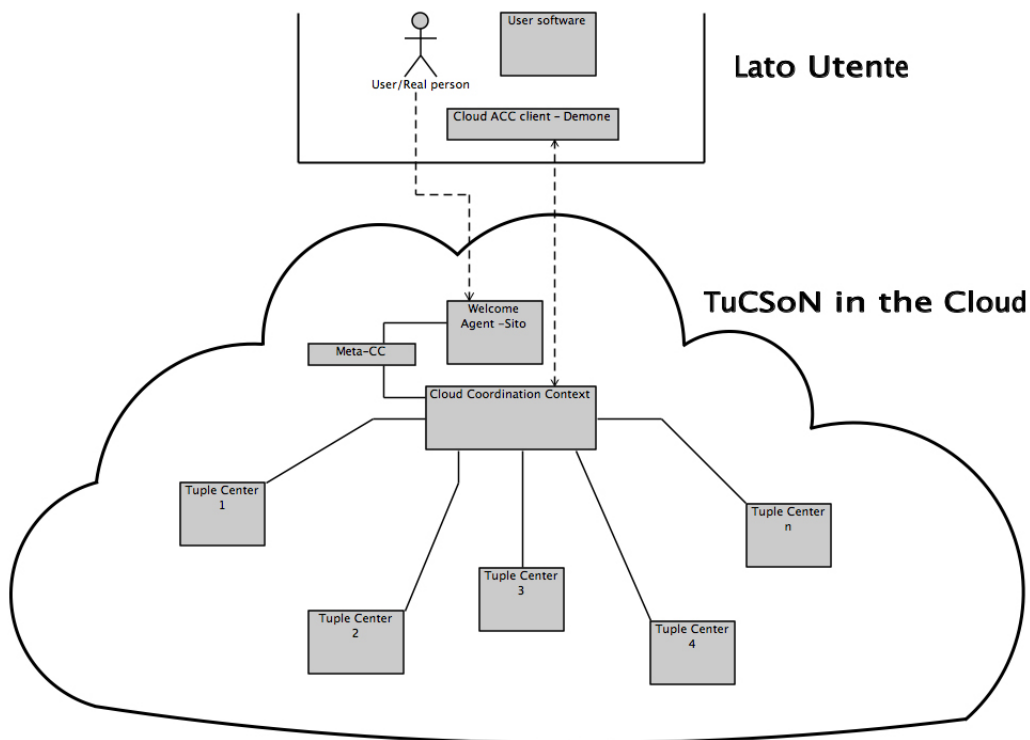


Figura 3.1: Modello astratto di *TuCSoN in the Cloud*

1.1 Scelte architetturali

Per realizzare un servizio CaaS con TuCSoN è stato necessario analizzare tutte le entità e nozioni esistenti attualmente nel modello di coordinazione e rivederle in ambito del Cloud Computing.

In TuCSoN le principali entità che caratterizzano un sistema coordinato sono i centri di tuple, i nodi e l'ACC. La nozione di centro di tuple rimane invariata, anche nel Cloud Computing essa rappresenta il mezzo di coordinazione. Dalla definizione di Cloud Computing si evince che la completa trasparenza

è un fondamento di base, per questo motivo si può affermare che la nozione di nodo TuCSoN dal punto di vista dell'utente non ha più senso di esistere in un servizio di tipo CaaS in the Cloud. L'utente infatti non conosce la reale locazione delle risorse fisiche (o virtuali) e la topologia con cui è realizzato il Cloud, quindi nel momento in cui esso invocherà un'operazione di coordinazione `op` non sarà necessario specificare il nodo a cui appartiene il centro di tuple che si vuole interrogare. Per l'utente tali informazioni non saranno più fondamentali e sarà quindi il provider a gestire come si concretizza il comportamento computazionale all'interno della "nuvola". Ciò comporterà una modifica dell'attuale sintassi dell'invocazione di un'operazione da parte dell'utente. Essa ora sarà solo del tipo:

`TupleCenterName ? op`

dove `TupleCenterName` rappresenta un nome logico assegnato dall'utente ad un centro di tuple acquistato. Esso sarà attribuito in modo dinamico alla prima invocazione su un centro di tuple non ancora identificato da quello stesso nome. All'interno del Cloud `TupleCenterName` sarà affiancato al nome logico della risorsa contenente, o che conterrà nel caso di allocazione dinamica, il tuple center identificato con quel nome.

Come per lo scenario dei sistemi distribuiti, anche nel Cloud Computing l'entità fondamentale di TuCSoN è l'ACC. È tramite di esso che avviene l'interazione, ricoprendo il ruolo di mediatore della coordinazione esso fornisce l'insieme di operazioni che l'utente può eseguire. Nel contesto del Cloud Computing esso potrà quindi essere denominato *Cloud Coordination Context (CCC)* e verrà configurato dal *Meta-Coordination Context*, il manager dei CCC, in base alla tipologia di servizio richiesto dall'utente. L'entità che prima raccoglie i dati di registrazione dal sito del servizio e poi li passa al Meta-Coordination Context utilizzandoli per la configurazione del CCC è la *Welcome Cloud Entity*. Un CCC è quindi associato ad ogni profilo registrato sul sito e, a seconda del servizio acquistato e alle informazioni specificate nel momento della registrazione dall'utente, il CCC dovrà fornire un numero limitato, o l'intero insieme di operazioni con le quali interagire con i centri di tuple assieme al numero massimo di centri di tuple acquistato. Tale configurazione deve però rimanere modificabile siccome il Meta-Coordination Context

dovrà eseguire l'aggiornamento del CCC nel momento in cui si verifichi un upgrade dell'abbonamento da parte dell'utente.

Il servizio CaaS che si vorrà offrire sarà costituito da un TuCSoN Service dove tutte queste entità-astrazioni interagiranno tra di loro e forniranno la coordinazione richiesta. Si è inoltre scelto di fornire tale servizio in modo da poterlo eseguire sia tramite browser web sia dalla postazione locale dell'utente. Quindi sarà previsto un eventuale download dallo stesso sito di un'applicazione (sotto forma di file .jar) o di un demone che risiederà nella macchina fisica dell'utente e che ricoprirà il ruolo di Cloud ACC lato client. Esso avrà lo scopo di non limitare al solo browser l'accesso all'interfaccia per l'utilizzo del servizio e di creare un canale di comunicazione con il CCC del Cloud al quale invierà l'operazione invocata. Spetterà poi all'utente decidere con che modalità accedere al servizio.

Di seguito saranno descritte in dettaglio le entità appena citate in modo da comprenderne meglio il reale funzionamento.

Welcome Cloud Entity

Rappresenta l'entità che risiede all'interno del sito e che, nel momento della registrazione, raccoglie i dati dell'utente e li passa al Meta-Coordination Context. Essa è anche responsabile di gestire gli upgrade degli account degli utenti segnalando al Meta-Coordination Context l'eventuale modifica di alcuni dati.

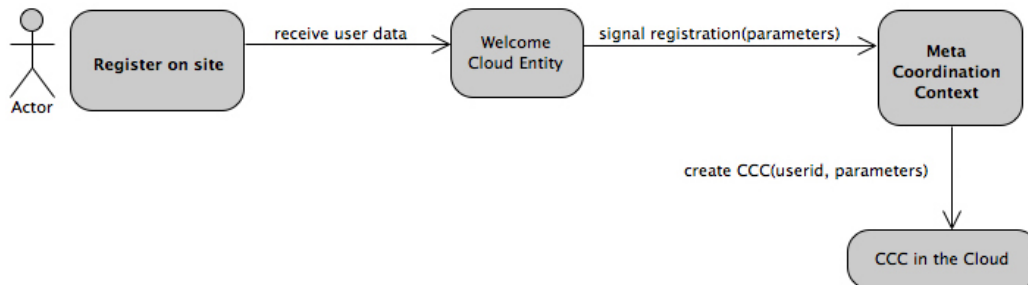


Figura 3.2: Registrazione dell'utente dal sito.

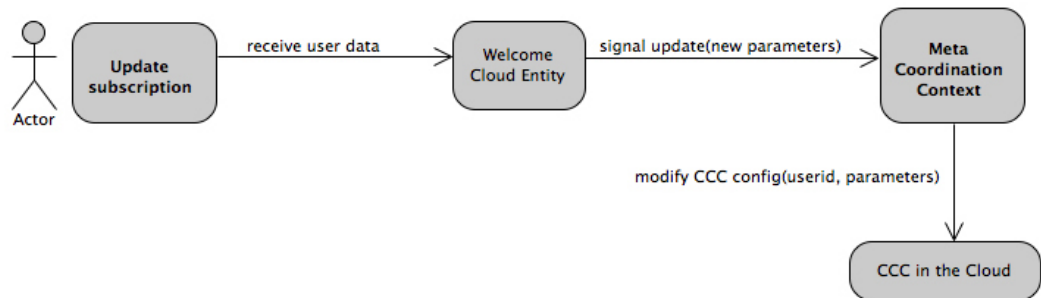


Figura 3.3: Update del contratto da parte dell'utente.

Meta-Coordination Context

È l'astrazione che ha il compito di creare e configurare il Cloud Coordination Context per l'utente registrato. Nel momento della registrazione dell'utente esso viene segnalato dal Welcome Cloud Entity, ne recupera i dati ed in base ad essi creerà un Cloud Coordination Context dedicato a quell'utente. Esso quindi invocherà il costruttore del CCC passandogli come parametri i dati fondamentali per la configurazione ad hoc, tra cui avremo un nome identificativo per l'utente che specificherà la relazione *Utente-CCC*, il numero di centri di tuple dichiarati dall'utente nel contratto con il provider e le operazioni che saranno messe a disposizione dell'utente stesso.

Il Meta-Coordination Context ricopre quindi il ruolo di manager dei CCC, esso infatti sarà anche incaricato di aggiornare la configurazione dello specifico CCC nel momento in cui il rispettivo utente eseguirà un upgrade dell'abbonamento sottoscritto con il provider. L'aggiornamento sarà segnalato dal Welcome Cloud Entity ed il Meta-Coordination Context provvederà alla nuova configurazione del CCC. La nuova configurazione setterà i nuovi parametri all'interno del Cloud Coordination Context andando a sostituire quelli derivanti dal vecchio contratto.

Cloud Coordination Context

Anche nel contesto del CaaS in the Cloud il Cloud Coordination Context permette all'utente di interfacciarsi al sistema TuCSoN ed è tramite di esso

che l'operazione di coordinazione viene effettivamente svolta. L'agente dualmente ai sistemi distribuiti non può interagire direttamente con i centri di tuple, ma può farlo solo attraverso lo specifico CCC.

Nella tipologia di servizio TuCSon in the Cloud si è voluto scegliere di fornire un'applicazione (sotto forma di file .jar) che conterrà il Cloud Agent Coordination Context lato client e avrà l'incarico di inviare l'invocazione di una operazione tramite la rete direttamente al rispettivo CCC nella "nuvola". Essa quindi ricoprirà il ruolo di applicazione proxy e nel momento in cui l'utente aprirà l'applicazione locale, essa richiamerà il CCC del Cloud (precedentemente configurato dal Meta-Coordination Context) e costruirà un canale di comunicazione, sarà quindi il tramite tra i due host ed inoltrerà le richieste e le risposte dall'uno all'altro. Se invece l'utente decidesse di scegliere di utilizzare il servizio direttamente da browser web egli interagirebbe con una GUI fornitagli dal suo CCC del Cloud, attraverso la quale potrebbe scrivere la sua richiesta e visualizzarne a video il risultato di ritorno.

Attualmente in nella più recente di TuCSon¹ è prevista solo la creazione e il rilascio di *ACC Enhanced*, ovvero un ACC completo di tutte le funzionalità disponibili all'interno del modello stesso. Questo aspetto nell'ambito di TuCSon in the Cloud potrebbe essere ammissibile solo nel caso in cui l'utente richieda e sottoscriva un contratto "*all inclusive*", pertanto sarà necessario fornire al Meta-Coordination Context la capacità di poter creare e far acquisire all'utente uno specifico CCC e non un CCC di tipo Enhanced. All'interno del modello TuCSon saranno previsti CCC specifici per i possibili abbonamenti predefiniti che l'utente potrà sottoscrivere e un CCC di tipo Custom (*CustomCCC*), che saranno configurati in base alle risorse e alle operazioni effettivamente richieste da parte dell'utente.

La figura che segue rappresenta il diagramma delle sequenze delle operazioni che vengono richiamate durante l'esecuzione di una operazione *op* da parte dell'utente sia che esso utilizzi l'applicazione client sia la GUI sul sito. Sono rappresentate entrambi i casi in cui il tuple center richiesto sia già presente tra quelli allocati e il caso in cui esso non sia ancora presente e quindi debba essere creato.

¹TuCSon v.1.10.3.0206

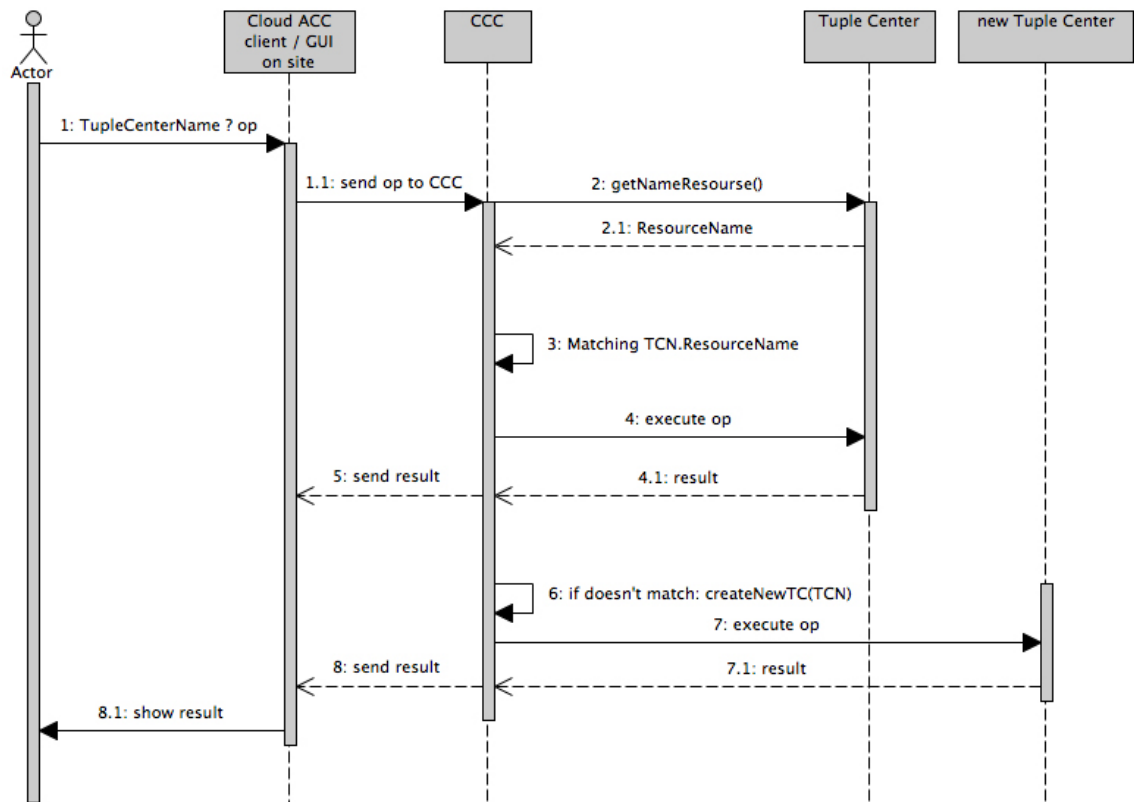


Figura 3.4: Architettura di un'operazione op

1.2 Naming dei centri di tuple

Essendo il Cloud Computing una metodologia di erogazione di servizi IT e gestione di risorse con alla base il principio di trasparenza totale per l'utente, quest'ultimo non avrà a disposizione le tutte informazioni sulla locazione fisica del centro di tuple (indirizzo IP e porta di ascolto) e conseguentemente sarà necessario poter assegnare a ciascun centro di tuple un nome **TupleCenter-Name** indicativo da utilizzare per l'invocazione di un'operazione. Dovrà essere possibile assegnare in modo dinamico un nome a piacere da parte dell'utente ad uno dei centri di tuple acquistati, quindi, vi dovrà essere un controllo da parte del CCC dell'effettiva presenza del centro di tuple chiamato tra quelli "in possesso" dall'utente. Nel caso in cui non vi sia una corrispondenza e il numero di centri di tuple già in uso non abbia già raggiunto il numero massimo previsto dal contratto con il provider, il CCC dovrà poterne tirarne

su uno ed assegnargli tale nome utilizzando le API fornite dai provider delle risorse fisiche o virtuali su cui poggia il servizio CaaS. Al contrario, se viene trovata una corrispondenza il CCC eseguirà l'operazione richiesta sullo specifico centro di tuple e ne restituirà successivamente il risultato (vedi Figura 3.4).

1.3 Naming delle risorse

Al fine di eseguire un'operazione il CCC dovrà conoscere dove il centro di tuple risiede, per tanto il nome identificativo completo del centro di tuple all'interno della "nuvola" sarà rappresentato da

`TupleCenterName.ResourceName`

rappresentante il nome assegnato dall'utente concatenato al nome logico della risorsa in cui si trova fisicamente il centro di tuple. `ResourceName` identifica un nome logico per la risorsa ospitante un tuple center, infatti come per i sistemi distribuiti, anche nel Cloud Computing le risorse fisiche e la loro locazione può cambiare nel tempo, ma tale spostamento deve rimanere trasparente all'utente, esso non dovrà percepire nessun cambiamento, per tanto dovranno essere identificabili da un nome che non verrà modificato.

Il nome della risorsa sarà recuperabile attraverso le API messe a disposizione dal provider dell'infrastruttura Cloud, come ad esempio Openstack², o della piattaforma Cloud, come ad esempio Cloudify³, infatti durante lo studio di tali infrastrutture e piattaforme si è notato che esse forniscono dei nomi e degli ID con cui identificano tutte le varie risorse.

Il servizio TuCSOn in the Cloud dovrà inglobare alcune delle funzionalità fornite attraverso le relative API dai servizi su cui poggerà per poter recuperare o settare tali nomi. Tali funzionalità dovranno essere rispettivamente utilizzate nella fase di verifica della presenza di un centro di tuple e nella creazione di un centro di tuple qualora non sia già presente(vedi Figura 3.4).

²Sito: <http://www.openstack.org/>

³Sito: <http://www.cloudifysource.org/>

1.4 Situatedness vs. Cloud

Con il concetto di *Situatedness* tecnicamente rappresentiamo l'abilità del sistema di percepire e reagire ai cambiamenti dell'ambiente[13]. Correlata alla nozione di Situatedness è presente l'astrazione di *Artefatto*, che consiste in una entità computazionale reattiva che ha il compito di mediare le reazioni ai cambiamenti tra il sistema stesso e l'ambiente delle risorse.

La nozione di *Situatedness*, introdotta nei sistemi MAS ed intesa come lo stretto accoppiamento con l'ambiente, può essere confrontata con il concetto base di trasparenza totale legato al Cloud Computing. Avendo introdotto TuCSoN all'interno del contesto Cloud, date le caratteristiche che contraddistinguono quest'ultimo, sarà ora opportuno porre una distinzione tra nodi situated (inter-cloud) e nodi nel Cloud (intra-cloud). Tale distinzione si può attuare in base al livello di trasparenza o, al contrario, al livello di dettaglio che il progettista e realizzatore del sistema vuole avere in un determinato nodo.

Nei sistemi distribuiti e nei sistemi MAS vi è la possibilità di raccogliere tutte le informazioni riguardanti un nodo, considerate dal punto di vista temporale, fisico e computazionale, tramite la nozione di situatedness. Essa ci permette di interagire con l'ambiente ospitante il nostro sistema e raccogliere, attraverso l'estensione del linguaggio di coordinazione ReSpecT nel caso di TuCSoN, tali informazioni e associarle ad uno specifico nodo. Nel modello attuale di TuCSoN alcune di esse sono necessarie per il corretto funzionamento ed il giusto indirizzamento delle operazioni al nodo specificato.

Al contrario nello scenario del Cloud Computing la nozione di situatedness "perde" di importanza, data l'elevata richiesta di trasparenza. Se per inoltrare un'operazione agli attuali nodi utilizzati in TuCSoN è necessario conoscerne l'indirizzo IP o la entry DNS del device ospitante ed il numero di porta in cui il servizio di coordinazione è in ascolto, tutto ciò non sarà necessario all'interno del Cloud. Pertanto la classificazione dei nodi in nodi situated e nodi cloud potrà basarsi sul livello di dettaglio con cui un singolo nodo è descrivibile e chiaramente il contesto in cui esso viene a trovarsi.

In un eventuale sistema ibrido, composto quindi sia da nodi situated e nodi cloud, la comunicazione tra i vari nodi dovrà essere adattata alle caratteristiche di entrambe le tipologie. Nello specifico, un nodo situated che riceve una

richiesta da un nodo cloud dovrà essere in grado di rispondere a tale richiesta nonostante quest'ultimo non gli abbia inviato tutte le informazioni (indirizzo IP e porta) a lui prima necessarie, parallelamente, un nodo cloud dovrà poter rispondere ad una determinata richiesta di un nodo situated selezionando solamente le informazioni a lui indispensabili per l'invio della risposta tra tutte quelle che caratterizzano il nodo situated.

Situatedness di TuCSoN in the Cloud

Nel contesto dello sviluppo di TuCSoN in the Cloud, l'ipotesi di un inserimento della nozione di situatedness in tale ambiente potrebbe rappresentare un punto delicato, in quanto essa comporta problemi di coordinazione. Essa sarebbe però utile al fine di estendere alcune funzionalità interne del sistema TuCSoN in the Cloud stesso. Attraverso di essa il nostro sistema diventerebbe sensibile ai cambiamenti che potrebbero avvenire nel Cloud e sarebbe in grado di produrre delle reazioni relative a questi cambiamenti.

Ad esempio si potrebbe estendere il sistema e il linguaggio di coordinazione ReSpecT, come nei sistemi MAS, in modo da tener traccia dei componenti che costituiscono le nostre risorse e di come esse variano nel tempo o delle istanze Cloud create. In questo caso saranno proprio i centri di tuple, basati proprio sul linguaggio ReSpecT, a ricoprire il ruolo di artefatti. Essi posseggono proprio i requisiti per catturare gli eventi dell'ambiente e mediare le interazioni tra l'ambiente stesso e gli agenti.

Prendendo come caso di studio la situazione in cui il nostro sistema TuCSoN in the Cloud possa tener traccia della memoria RAM installata ed utilizzata nelle risorse allocate, si potrebbe pensare ad un sensore (controllo tramite software) posto su ogni risorsa che capti le variazioni hardware di questo componente, ed in caso di cambiamenti avverta il tuple center specifico, il quale provvederà poi a rilasciare determinate reazioni in base ai dati che gli verranno passati. Altro caso di studio interessante potrebbe riguardare la localizzazione delle risorse, il nostro sistema attraverso un sensore di geolocalizzazione potrebbe essere informato di eventuali cambiamenti nella topologia del Cloud e aggiornare una teorica mappa delle risorse utilizzate in tutto il sistema.

Questi sono solo alcuni dei casi di studio che si potrebbero sviluppare per

TuCSoN in the Cloud, ed in base ad essi si dovranno progettare le nuove architetture, per rappresentare le entità coinvolte e si dovrà estendere il linguaggio di coordinazione ReSpecT, inserendo alcune reazioni adibite al controllo delle proprietà dell'ambiente in cui risiede e abilitando così il supporto alla Situatedness nel nostro sistema.

1.5 Primitive nel Cloud

Le primitive definiscono il comportamento con cui il modello di coordinazione TuCSoN opera sui centri di tuple, per tanto sia che esso si trovi nel contesto dei sistemi distribuiti, sia nell'ambito del Cloud Computing, queste non dovranno essere modificate in modo radicale, altrimenti si andrebbe a cambiare ciò che caratterizza TuCSoN stesso.

Alcune di esse potrebbero integrare alcune delle le funzionalità offerte dalle piattaforme Cloud su cui TuCSoN potrebbe appoggiarsi per fornire un servizio TuCSoN in the Cloud.

Ad esempio, se l'operazione `get()` viene invocata su un centro di tuple che risulta essere vuoto, dopo aver restituito una lista vuota potrebbe sospendere la risorsa su cui si trova il tuple center o addirittura rimuoverlo dalla risorsa stessa così da decrementare il numero di tuple center utilizzati dall'utente. Questa seconda opzione sarebbe molto vantaggiosa se l'utente avesse raggiunto il numero massimo di centri di tuple utilizzabili per contratto e quindi non potrebbe allocarne più. A volte però il centro di tuple, target dell'operazione `get()`, potrebbe servire nuovamente, quindi il CCC per evitare una eventuale cancellazione dopo il ritorno di una lista vuota, potrebbe fornire all'utente al posto di una semplice `get()`, una operazione chiamata ad esempio `get_p()` con la stessa semantica della `get()`, ma nella quale si possa specificare un parametro rappresentante la volontà dell'utente di cancellare o meno il tuple center se vuoto.

1.6 Servizio pay-per-use

Come tutti gli altri servizi di Cloud Computing, ad esclusione di quelli open-source, anche il servizio TuCSoN in the Cloud potrebbe essere offerto come servizio *pay-per-use*. In realtà non si possono definire delle regole precise sui

metodi di pagamento per tale servizio, in quanto è difficile quantificare realmente l'utilizzo ed il consumo di una risorsa all'interno del Cloud da parte del modello TuCSoN. Pertanto di seguito saranno fornite solo alcune ipotesi su possibili politiche di prezzo e non uno schema definitivo. Tali politiche potrebbero generalmente basarsi su un importo fisso periodico o in base al consumo. Nel caso di servizio a importo fisso periodico potremmo fare un'ulteriore divisione: un possibile servizio *enterprise* per aziende ed organizzazioni ed uno *consumer* per singoli utenti. Questo tipo di abbonamenti propongono delle fasce di prezzo nelle quali sono elencate le risorse messe a disposizione dell'utente, come numero di tuple center e di utenti per account, il cui numero è fissato a priori. Tipicamente i servizi enterprise sono più costosi rispetto a quelli consumer visto il numero maggiore di risorse messe a disposizione e, a volte, le capacità computazionali superiori rispetto a quelle consumer. Anche nel caso di servizi con pagamento in base al consumo vi potrebbe essere la stessa distinzione enterprise e consumer, ma questa volta il costo del servizio varierà in base alle risorse effettivamente consumate o richieste. Nel caso di TuCSoN in the Cloud tale costo potrebbe essere calcolato in base a:

- numero di centri di tuple richiesto
- numero e tipo di operazioni richieste
- numero di utenti che possono accedere allo stesso account (utilizzato maggiormente negli account enterprise)

Chiaramente sia nel caso di servizi in abbonamento sia in quello di costo in base al consumo vi sarà la possibilità di fare un upgrade, se necessario. Una buona strategia di marketing potrebbe essere inoltre quella di fornire un periodo di prova del servizio, ad esempio di 30 giorni, nel quale l'utente può utilizzare il servizio e alla fine del quale sarà libero di acquistare il prodotto se soddisfatto, altrimenti in caso contrario virare su un altro servizio.

Capitolo 4

TuCSoN through Cloudify

In questo capitolo viene spiegato il caso di studio del CLI di TuCSoN in the Cloud, che è stato implementato come applet attraverso un servizio Cloud basato sulla piattaforma Cloudify¹ distribuita da Gigaspaces².

Essa consiste in una piattaforma PaaS che si interpone tra l'applicazione e l'infrastruttura Cloud scelta, garantisce che le risorse necessarie siano disponibili indipendentemente dall'infrastruttura e dallo stack scelto ed è progettata per “spostare” le applicazioni nella nuvola senza apportare modifiche al codice o a qualsiasi altro componente middleware utilizzato.

Cloudify supporta sia infrastrutture cloud pubbliche (come Amazon EC2, Windows Azure ecc.) sia infrastrutture private (come OpenStack, cloud.com, VMWare vCenter ecc.) ed è anche in grado di creare cloud locali su cui testare le proprie applicazioni. Essa isola l'applicazione dell'utente dall'infrastruttura cloud di cui ne possiede l'accesso, in modo che un'applicazione possa essere spostata facilmente da una nuvola all'altra, fornendo così un alto grado di libertà di scelta per l'utente.

Cloudify, inoltre, fornisce un meccanismo per gestire l'intero ciclo di vita dell'applicazione ed utilizza le cosiddette *recipes* per descrivere un'applicazione, i suoi servizi e le loro interdipendenze, le modalità di monitoraggio, lo scaling dei suoi servizi e delle loro risorse. Di conseguenza, la distribuzione e la gestione l'applicazione diventa un processo relativamente semplice.

¹Sito: <http://www.cloudifysource.org/> - Versione: 2.3.1.1 GA

²Sito: <http://www.gigaspaces.com/>

1 Cloudify CLI

La piattaforma è accessibile tramite una Command Line Interface dedicata presente all'interno del file di distribuzione di Cloudify. La CLI richiede come requisiti minimi per la sua installazione la versione 1.6 o superiore del JDK di Java e necessita che la variabile d'ambiente `JAVA_HOME` sia settata ed esportata nella cartella di sviluppo di Cloudify. Una volta verificati tali requisiti essa è lanciabile tramite il comando da terminale `sh launch.sh` eseguito nella cartella `bin` all'interno di Cloudify.

```

richiard@richiard-VirtualBox: ~/Cloudify/bin
.o00000.  0000          .o8  o8o  .o88o.
d8P'  `Y8b  888          "888  `"'  888  `
888      888  .00000.  0000  0000  .0000888  0000  o8880o  0000  000
888      888  d88'  `88b  `888  `888  d88'  `888  `888  888  `88.  .8'
888      888  888  888  888  888  888  888  888  888  888  `88..8'
`88b  00o  888  888  888  888  888  888  888  888  888  `888'
`Y8bood8P'  o888o  `Y8bod8P'  `V88V"V8P'  `Y8bod88P"  o888o  o888o  .8'
                                                    .o..P'
                                                    `Y8P'

GigaSpaces Cloudify Shell.

Note for Windows Users:
The Cloudify shell does not currently support the back-slash character ('\')
as file separator. Instead, use the forward-slash character (/) when
specifying file paths.

Hit '<tab>' for a list of available commands.
Hit '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or 'exit' to exit the console.

Cloudify version: 2.3.1-RELEASE

cloudify@default>

```

Figura 4.1: CLI di Cloudify

Operazioni disponibili

Attraverso il pulsante `tab` è possibile visualizzare tutte le operazioni eseguibili una volta entrati nella CLI di Cloudify. Tali operazioni sono:

- `add-template`: comando per aggiungere un template al cloud riferendo ad un file template o ad una cartella template contenente una serie di file template;
- `bootstrap-...`: `cloud`, richiama il Cloudify Agent ed il processo di gestione Cloudify sul cloud specificato come argomento; `localcloud`,

richiama il Cloudify Agent ed il processo di gestione Cloudify sulla macchina locale;

- **bye/exit/quit**: tali comandi terminano la shell;
- **clear**: pulisce la consolle;
- **connect/disconnect**: permette di collegarsi/disconnettersi al/dal server REST amministratore specificato come argomento sottoforma di URL;
- **help**: visualizza la lista completa di tutti i comandi disponibili;
- **install-...**: **application**, installa l'applicazione specificata o dal file path o dalla cartella o dal archivio specificato come argomento; **service**, installa il servizio specificato o dal file path o dalla cartella o dal archivio specificato come argomento;
- **invoke**: invoca uno specifico comando personalizzato, ha come argomenti il servizio su cui vogliamo invocare tale comando e il nome del comando stesso;
- **list-...**: **applications**, visualizza la lista delle applicazioni inserite nel Cloud; **attributes**, visualizza la lista degli attributi dello store di controllo degli attributi di Cloudify; **instances**, visualizza tutte le istanze di uno specifico servizio passatogli come argomento; **services**, visualizza la lista di tutti i servizi inseriti nell'applicazione corrente; **templates**, visualizza la lista di tutti i templates Cloud;
- **login**: permette di riconnettersi al server amministratore REST;
- **remove-...**: **attributes**, rimuove gli attributi specificati come argomento dallo store di controllo Cloudify; **templates**, rimuove il template specificato dal Cloud;
- **set-...**: **attributes**, setta gli attributi specificati all'interno dello store di controllo degli attributi di Cloudify; **instances**, setta il numero di servizi di un servizio scalabile per un certo numero di istanze inserite come argomento;
- **shutdown-...**: **agent**, interrompe l'esecuzione dell'agente che risiede sulla macchina locale; **management**, interrompe l'esecuzione dell'agente di gestione che risiede sulla macchina locale, entrambi i comandi sono eseguibili solo per un uso interno;

- **start-...**: **agent**, mette in esecuzione l'agente Cloudify all'interno della specifica zona, esso comunica con l'agente generale di Cloudify e con le macchine di gestione; **management**, mette in esecuzione l'agente Cloudify all'interno della specifica zona di gestione e nel processo di gestione sulla macchina locale, esso comunica con l'agente generale di Cloudify e con le macchine di gestione, entrambi i comandi sono eseguibili solo per un uso interno;
- **tail**: recupera le ultime n righe di un log di uno specifico servizio
- **teardown-...**: **cloud**, termina le macchine di gestione del Cloud passatogli come argomento; **localcloud**, termina e disinstalla il Cloud locale installato sulla macchina;
- **test-recipes**: testa la correttezza della recipe passatagli come argomento;
- **uninstall-...**: **application**, disinstalla l'applicazione specificata come argomento; **service**, rimuove un servizio specificato;
- **use-application**: setta l'applicazione da utilizzare;
- **validate**: convalida un file DSL;
- **validate-...**: **application**, convalida un file applicazione DSL; **service**, convalida un file servizio DSL;
- **version**: visualizza la versione di Cloudify.

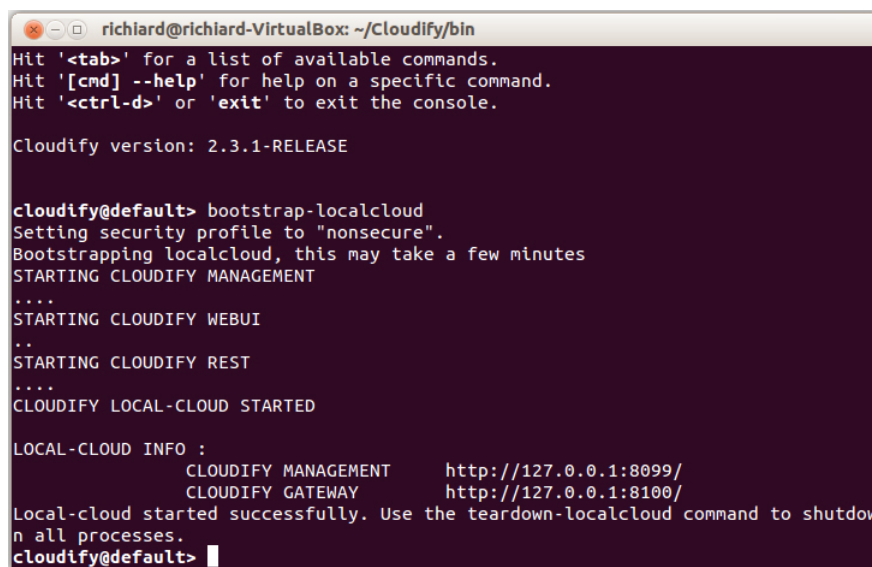
Tutte le operazioni permettono di specificare l'opzione **--verbose**, con la quale è possibile visualizzare nella shell il dettaglio delle operazioni eseguite dal comando richiamato.

2 Local Cloud

Per il caso di studio del CLI di TuCSoN in the Cloud si è deciso di sfruttare l'opportunità offerta da Cloudify di installare sulla propria macchina un Cloud locale tramite il comando **bootstrapping-localcloud**.

Un Cloud locale è un vero e proprio ambiente di emulazione di una “nuvola” che permette all'utente di eseguire tutta la gestione Cloudify e di servizi applicativi su una singola macchina. Per creare tale emulazione (ma anche per fare il deploy di un vero e proprio Cloud) Cloudify utilizza un processo

bootstrapping con il quale mette a disposizione le macchine per la gestione di Cloudify, e le macchine su cui pogeranno i servizi necessari per la distribuzione delle applicazioni. Il processo di bootstrap è quindi responsabile dell'assegnazione delle macchine (virtuali o fisiche) e dell'installazione del software necessario alla fornitura dei componenti Cloudify rilevanti.



```
richiard@richiard-VirtualBox: ~/Cloudify/bin
Hit '<tab>' for a list of available commands.
Hit '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or 'exit' to exit the console.

Cloudify version: 2.3.1-RELEASE

cloudify@default> bootstrap-localcloud
Setting security profile to "nonsecure".
Bootstrapping localcloud, this may take a few minutes
STARTING CLOUDIFY MANAGEMENT
****
STARTING CLOUDIFY WEBUI
**
STARTING CLOUDIFY REST
****
CLOUDIFY LOCAL-CLOUD STARTED

LOCAL-CLOUD INFO :
      CLOUDIFY MANAGEMENT      http://127.0.0.1:8099/
      CLOUDIFY GATEWAY          http://127.0.0.1:8100/
Local-cloud started successfully. Use the teardown-localcloud command to shutdown all processes.
cloudify@default>
```

Figura 4.2: Bootstrapping di un Cloud locale

Oltre ad installare il Cloud locale Cloudify inizializza e invoca anche i processi **MANAGEMENT**, **WEBUI** e **REST** istanziando i rispettivi servizi, i quali verranno messi a disposizione dell'utente. Tra questi i più importanti sono il **MANAGEMENT** e **WEBUI** perchè permettono all'utente di visualizzare lo stato delle risorse coinvolte dal Cloud ed al termine dell'installazione il CLI fornisce gli indirizzi IP con le rispettive porte di ascolto a cui accedere per visualizzare tutte le informazioni di management.

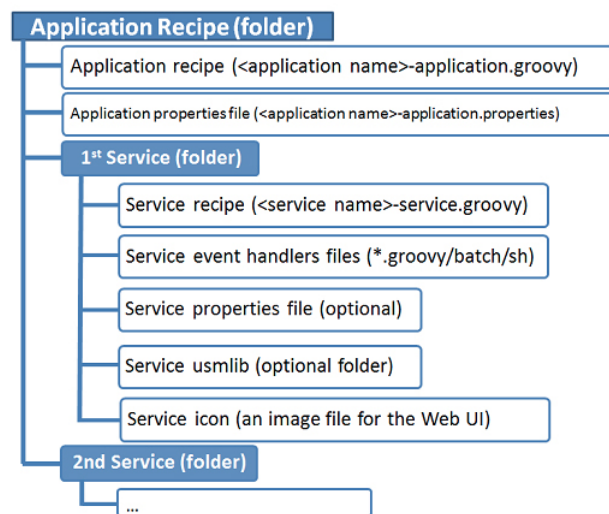
Infine utilizzando il comando `teardown-localcloud` potremmo terminare i processi alla base del Cloud locale.

3 Cloudify Recipes

Una recipe descrive le interdipendenze tra i diversi livelli di applicazione (servizi), l'intero processo di installazione per la fornitura di ogni livello, l'insieme dei compiti post-distribuzione, come gli indici di monitoraggio dell'andamento di un processo, la gestione delle fail-over, e le regole per gestire autonomamente la scalabilità e il corretto funzionamento del sistema. Essa rappresenta quindi una serie di istruzioni su come Cloudify dovrà prendere ed elaborare la nostra applicazione sul cloud.

Si distinguono due principali tipi di recipes, application recipes e service recipes. Al fine di distribuire un'applicazione è necessario creare un'unica application recipe che utilizza uno o più service recipe con le quali specifico i servizi di cui l'applicazione ha bisogno.

Tutti i file che compongono l'applicazione e la recipe application devono essere situati nella cartella `<Cloudify folder>/recipes/apps/<application folder>` e dovranno seguire l'anatomia rappresentata nella seguente figura.



3.1 Application recipes

Una application recipe è costituita dal file descriptor con il nome dell'applicazione, dai servizi di cui ha bisogno e dalle loro interdipendenze. Specificando le dipendenze tra i vari servizi dichiariamo alla piattaforma Cloudify di non attivare l'evento di inizio ciclo di vita del servizio dipendente fino a quando i

servizi da cui esso dipende non sono stati tutti avviati con successo. Inoltre il nome dell'applicazione viene utilizzato per identificare l'applicazione da parte dei vari strumenti di monitoraggio Cloudify, pertanto il nome dovrà essere univoco in tutte le applicazioni che verranno gestite dal controller di Cloudify.

3.2 Service recipes

Una service recipe è costituita dal file descriptor del servizio dove possono essere specificate le seguenti informazioni caratterizzanti il servizio stesso:

Caratteristiche generali - contengono il nome del servizio, il tipo di servizio, il numero di istanze previste ed il nome del file icona che si vuole assegnare al servizio nel WEBUI di management di Cloudify.

Specifiche - dichiarano il nome del modello da utilizzare per le macchine di distribuzione del servizio. Il nome del modello corrisponde a un nome di modello specificato nel driver Cloud che descrive i profili hardware di un insieme di macchine che possono essere utilizzate quando verranno distribuite le istanze di servizio. Un profilo hardware mappa a una specifica configurazione hardware e le dimensioni dell'immagine. Questo permette una netta separazione tra la application recipe e l'ambiente Cloud su cui viene eseguita.

Eventi del ciclo di vita - sono gli eventi utilizzati per gestire gli script o le chiusure del servizio e, come tali, sono la parte più importante della service recipe. Un gestore eventi può essere uno script esterno con estensione .groovy , uno script di shell, o un file batch (a seconda del sistema operativo). I gestori devono risiedere nella cartella di servizio.

Comandi personalizzati - specifica i comandi personalizzati per gestire gli script che possono essere invocati come comandi utilizzando la shell Cloudify.

Oggetti di monitoring - descrizione dei "sensori" e dei plugin per la configurazione del servizio di monitoraggio. Specifica anche il layout da utilizzare per la visualizzazione dei parametri raccolti nella WEBUI di gestione di Cloudify.

Non tutte queste sezioni devono obbligatoriamente essere specificate, sarà l'utente a decidere cosa volere implementare e con quanto dettaglio.

4 CLI di TuCSoN in the Cloud

Si è pensato alla realizzazione del CLI di TuCSoN, il Command Line Interpreter, tramite una semplice interfaccia realizzata come applet. Dato il contenuto dinamico che essa rappresenta e per la caratteristica di accesso tramite browser di qualsiasi servizio di Cloud Computing, si è deciso di inserire tale applet in una pagina html e di integrarla all'interno di un server Apache Tomcat³. quest'ultimo è stato poi distribuito sul Cloud grazie alla piattaforma Cloudify.

4.1 Applet

L'applet proposta consiste in una semplice interfaccia grafica i cui componenti principali sono:

- una JTextArea utilizzata per la visualizzazione dei log del CLI;
- un JTextField nel quale sarà possibile digitare l'operazione che si vorrà passare al CLI;
- un JButton che avrà il compito di inviare l'operazione inserita dall'utente al CLI.

All'interno dell'applet sono presenti inoltre due thread, uno generico che ha il compito di creare la GUI dell'applet con cui l'utente potrà interagire ed un ThreadApplet che gestirà tutte le operazioni inviate al CLI, esso quindi farà le veci dell'attuale CLI Agent all'interno di TuCSoN.

È stata creata poi una pagina html nella quale poter inserire l'applet utilizzando il tag `<applet>` e fornendogli i parametri necessari per il caricamento del file `.class`, tra cui il nome del file e la dimensione dello spazio in cui verrà disposta la nostra applet specificando altezza e larghezza. È stato necessario inserire altri parametri come ad esempio il `codebase`, l'URL del file, in quanto si è deciso per semplicità di posizionare il file all'interno di una

³Sito: <http://tomcat.apache.org/>

cartella `applet` divisa dal file `html`.

Tale pagina poi è stata tradotta per essere integrata con il servizio di Tomcat in una pagina JSP.



Figura 4.3: Interfaccia del CLI

Si indirizza il lettore all'appendice A per la visualizzazione dell'implementazione di tale interfaccia.

4.2 Cloudify

Per poter distribuire l'applet tramite Cloudify è stato necessario sviluppare una semplice application recipe, con la quale si è specificato che l'applicazione TuCSoN CLI richiede un servizio Tomcat, si è inoltre emulato anche un nodo TuCSoN creando una recipe dedicata all'avvio stesso del nodo. Questo è stato fatto per poter verificare una volta installata l'applicazione TuCSoN in the Cloud se il nostro CLI è in grado di comunicare con un semplice nodo TuCSoN distribuito sul Cloud o se queste semplici operazioni non sono sufficienti per realizzare tale comportamento. Sono state quindi utilizzate due service recipe, una per il nodo TuCSoN e l'altra per il CLI da noi realizzato.

Recipe Nodo

La recipe dedicata al nodo non è altro che una semplice invocazione del file `launch.sh` fornito attualmente con la distribuzione di TuCSoN, con l'inserimento dell'attributo `Node` che ci permetterà di avviare un processo sul Cloud con lo scopo di creare un nodo TuCSoN.

Si indirizza il lettore all'appendice A per la visione del codice della `node recipe`.

Recipe Tomcat

Cloudify fornisce degli esempi pratici in cui sono utilizzate recipe per creare servizi Tomcat, di conseguenza si è deciso di utilizzare una recipe già esistente modificandola in base alle specifiche della nostra applet nella pagina html.

4.3 Installazione

Una volta create le varie recipes con all'interno i relativi file `.groovy` e i file su cui essi si basano, si sono eseguite le seguenti operazioni:

- Lancio di Cloudify
- Creazione del Cloud locale tramite il comando `bootstrap-localcloud`
- Installazione dell'applicazione TuCSoN in the Cloud digitando il comando `install-application TuCSoNcloud`
- Verifica della corretta installazione dell'applicazione tramite l'accesso all'indirizzo su cui Cloudify fornisce il servizio di management

4.4 Management

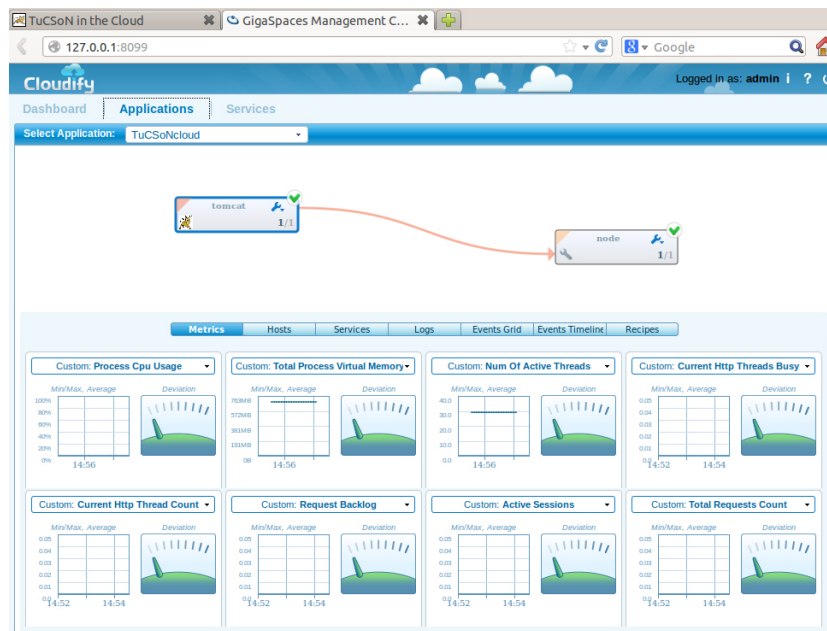
Cloudify permette di visionare lo stato della propria applicazione e dei relativi servizi accedendo all'indirizzo IP e porta indicati al momento del bootstrapping del Cloud sotto la voce `CLOUDIFY MANAGEMENT`. In questo caso di studio, una volta terminata l'operazione di bootstrap del local-cloud si è appreso che l'indirizzo di default al quale accedere per il management è `http://127.0.0.1:8099/`. Una volta inserito l'indirizzo, per poter usufruire

dell'interfaccia di management è necessario eseguire il login con le seguenti credenziali:

Username: *admin*

Password: *password*

e così si potrà utilizzare la seguente interfaccia per le operazioni di gestione e controllo:



4.5 Problemi riscontrati

Durante lo sviluppo dell'applet e l'utilizzo di Cloudify si è riscontrato che quest'ultimo prima della versione 2.3.1.1 GA, rilasciata il 12 Febbraio 2013, soffriva di due bug di java, uno riguardante il `logging.Level JDK error` che occorreva con l'utilizzo del Java Development Kit 1.7 U13 e 1.6 U39, l'altro riguardante un `OutOfMemory error` che si presentava nel caso in cui si volevano sviluppare servizi scritti su file di grandi dimensioni.

Tali bug non si presentavano nel momento in cui al posto del JDK distribuito

da ORACLE⁴ si utilizzava l'openJDK⁵.

Una volta ovviate queste problematiche si è proseguito con lo sviluppo dell'applet e del suo inserimento all'interno del Cloud. Si è potuto verificare, durante lo sviluppo dell'applet, la corretta visualizzazione della GUI del CLI solo attraverso il browser Firefox⁶, in quanto Google Chrome⁷ non riusciva ad inizializzare correttamente l'applet.

Nonostante l'installazione di TuCSoN in the Cloud tramite Cloudify avvenga con successo, nel momento in cui viene passata un'operazione al CLI, probabilmente per un motivo di comunicazione interno e di dipendenza tra i servizi di Cloudify viene lanciata un'eccezione per `UnreachableNodeException`.

⁴Sito: <http://www.oracle.com/us/technologies/java/overview/index.html>

⁵Sito: <http://openjdk.java.net/>

⁶Sito: <http://www.mozilla.org/it/firefox/new/>

⁷Sito: <https://www.google.com/intl/it/chrome/browser/>

Conclusioni e sviluppi futuri

In questa tesi si è cercato di progettare una prima architettura del concetto di *Coordination as a Service* nel Cloud prendendo in considerazione come riferimento il modello di coordinazione TuCSoN.

Per iniziare si è cercato di dare una panoramica su cosa consista il Cloud Computing e su tutte le sue caratteristiche principali analizzando anche gli attori coinvolti in tale scenario. Si sono sviluppati i punti ritenuti fondamentali, come le tipologie di servizio offerte dal Cloud Computing e i relativi modelli di distribuzione, che mi avrebbero aiutato a raggiungere l'obiettivo principale di progettare l'architettura di TuCSoN in the Cloud, un "porting di TuCSoN" all'interno della nuvola. Da questa analisi si sono apprese quindi le nozioni di base ed i vincoli che la mia futura architettura avrebbe dovuto rispettare.

Successivamente si è illustrato il concetto base di coordination as a service incentrato sull'astrazione di mezzo di coordinazione e la cui architettura logica è rappresentata da spazio coordinato, spazio di interazione e spazio di coordinazione. È stato studiato l'attuale modello di coordinazione TuCSoN, sua la rispettiva architettura per acquisirne i concetti fondamentali e rividerli in ottica del Cloud Computing ed è stato analizzato il suo linguaggio di coordinazione, con la suddivisione delle principali primitive, assimilando le nozioni di agenti, centri di tuple ReSpecT, nodi ed Agent Coordination Context.

Una volta terminato lo studio di TuCSoN, tutte le nozioni che lo caratterizzano sono state riviste, estese o addirittura eliminate (vedi la nozione di nodo nel Cloud dal punto di vista dell'utente, ma non da quello dello sviluppatore) in base alle caratteristiche principali ed alle funzionalità offerte dal Cloud Computing. In base a quest'ultime sono state fatte delle scelte

architetture che hanno portato all'introduzione delle astrazioni di *Welcome Entity*, *Meta-Coordination Context* e *Cloud Coordination Context*, caratterizzanti il modello TuCSoN in the Cloud e di come esse possano interagire tra di loro, fornendo anche una possibile architettura di una generica operazione op eseguita dall'utente e prendendo in considerazione il ruolo delle primitive anche nel Cloud. Inoltre, dato che il Cloud Computing si basa sul principio di trasparenza totale si è deciso di introdurre una forma di *naming* per attribuire ai centri di tuple ed alle risorse su cui esse risiedono un nome identificativo univoco così da poterle rappresentare indipendentemente dalle loro informazioni di rete, fornendo così la trasparenza richiesta. Questa scelta ha inoltre scaturito una possibile classificazione dei nodi di TuCSoN in base al livello di dettaglio con cui identificare un nodo stesso. La classificazione riguarda i nodi *situated* (tipici dei sistemi distribuiti attuali e sistemi MAS), i quali sono identificati da un numero elevato di informazioni e posseggono la capacità di interagire con l'ambiente, e i nodi *Cloud* che posseggono un minor numero di informazioni con cui identificarli. Si è inoltre affrontato un possibile aspetto monetario del servizio CaaS TuCSoN in the Cloud evidenziando come quest'ultimo possa essere distribuito ed utilizzato dopo la sottoscrizione di abbonamenti *consumer*, per i singoli utenti, ed *enterprise* per le aziende e la registrazione sul sito del provider.

L'obiettivo principale di questa tesi è stato quindi quello di ideare e formalizzare un semplice modello ed una architettura del servizio TuCSoN in the Cloud, senza andare a stravolgere l'attuale architettura del modello implementato per sistemi distribuiti e sistemi multi-agent.

Tale architettura deve essere considerata come un punto di partenza da cui incominciare per uno sviluppo vero e proprio del modello TuCSoN in the Cloud e della sua distribuzione sul Cloud e del quale potranno essere implementate concretamente le astrazioni introdotte in questa tesi e i relativi meccanismi di comunicazione.

È stato inoltre proposto un caso di studio in cui si è creato un servizio Cloud tramite la piattaforma Cloudify, che propone un'interfaccia utente per l'attuale Command Line Interpreter di TuCSoN, e l'avvio di un nodo TuCSoN all'interno della nuvola. L'interfaccia è stata concepita come applet

su pagina html nella quale l'utente ha la possibilità di inserire un'operazione ed inviarla al CLI stesso tramite un bottone dedicato all'invio. Con l'utilizzo dell'applet si è rispettato il requisito base del Cloud il quale prevede l'accesso di un suo servizio tramite browser Web, tale caso di studio però può essere considerato solo un semplice esempio e non deve essere inteso come unico metodo di implementazione di sviluppo di un servizio TuCSOn in the Cloud.

Appendice A

Sorgenti

Di seguito sono riportati i codici sorgente dell'interfaccia della CLI di TuC-SoN in the Cloud e dei file `.groovy` che hanno permesso l'inserimento di un nodo TuCSoN all'interno del Cloud.

GUI del CLI:

```
1 public class AppletCLI extends JApplet {
2
3     private JTextArea schermo;
4     private JButton send;
5     private JTextField campo;
6     private LinkedList<String> stringlist;
7
8     public void init(){
9
10        //Thread per la creazione della GUI dell'applet
11        try{
12            javax.swing.SwingUtilities.invokeLater(new Runnable(){
13                public void run(){
14                    createGUI();
15                }
16            });
17        }catch (Exception e){
18            throw new RuntimeException(e);
19        }
20    }
21
22    public void start(){
23
24        ThreadApplet t = new ThreadApplet(schermo, campo, stringlist);
25        t.start();
26
27    }
```

```
28
29 private void createGUI(){
30
31     //Container dell'applet
32     Container c = getContentPane();
33     c.setLayout(new BorderLayout(c, BorderLayout.Y_AXIS));
34
35     schermo = new JTextArea(500,200);
36     schermo.setAlignmentX(Component.CENTER_ALIGNMENT);
37     JScrollPane scroll = new JScrollPane();
38     scroll.add(schermo);
39     c.add(schermo);
40     campo = new JTextField(30);
41     campo.setAlignmentX(Component.CENTER_ALIGNMENT);
42     c.add(campo);
43     send = new JButton("Send operation to the CLI");
44     send.setAlignmentX(Component.CENTER_ALIGNMENT);
45     c.add(send);
46
47     //Creazione della LinkedList che conterrà il contenuto del JTextField
48     stringlist = new LinkedList<String>();
49
50     AppletListener invio = new AppletListener(send,campo,schermo,stringlist);
51     send.addActionListener(invio);
52
53 }
54
55 public class AppletListener implements ActionListener {
56
57     JTextArea area;
58     JTextField text;
59     JButton button;
60     String testo;
61     LinkedList<String> lista;
62
63     public AppletListener(JButton mybutton,JTextField mytext, JTextArea myarea,
64         LinkedList<String> mylist){
65
66         button = mybutton;
67         text = mytext;
68         area = myarea;
69         lista = mylist;
70
71     }
72
73     public void actionPerformed(ActionEvent e) {
74
75         testo = text.getText();
76         //area.append(testo + "\n");
77         lista.add(testo);
78         campo.setText("");
79
```



```
80     }
81   }
82 }
```


TuCSoN application recipes:

```
1 application {
2   name = "TuCSoNcloud"
3
4   service{
5     name = "node"
6   }
7
8   service{
9     name = "tomcat"
10    // Esprimo la dipendenza con il nodo
11    dependsOn = ["node"]
12  }
13 }
```


TuCSoN node recipes:

```
1 service {
2   name "node"
3
4   lifecycle{
5     start {
6
7         def cmdStr = "bash launch.sh Node"
8         def process = "${cmdStr}".execute()
9
10        // Itera il processo linea per linea
11        process.in.eachLine {line->
12          println line;
13        }
14      }
15    }
16 }
```


Bibliografia

- [1] Peter Mell, Timothy Grance - *“The NIST Definition of Cloud Computing”*, Special Publication 800-145, September 2011;
- [2] Fang Liu, Jin Tong, Jian Mao, Robert Bohn, John Messina, Lee Badger, Dawn Leaf - *“NIST Cloud Computing Reference Architecture”*, Special Publication 500-292, September 2011;
- [3] George Reese - *“Cloud computing. Architettura, infrastrutture, applicazioni”*, Hops-Internet e. Tecniche Nuove, 2010, ISBN 9788848124935;
- [4] Mirko Viroli, Andrea Omicini - *“Coordination as a Service”*, IOS Press, Fundamenta Informance 72, 2006, 1-28;
- [5] Paolo Ciancarini - *“Coordination models and languages as software integrators”*, ACM Computing Surveys, 28(2), June 1996, 300-302, ISSN 0360-0300.
- [6] Mary Shaw, Robert DeLine, Daniel V. Klein, Theodore L. Ross, David M. Young, Gregory Zelesnik - *“Abstraction for Software Architecture and Tools to support them.”*, IEEE Trans. Softw. Eng. 21, April 1995;
- [7] Andrea Omicini - *“On the Semantics of Tuple-based Coordination Models”*, ACM Symposium on Applied Computing (SAC '99), San Antonio (TX), 28 February - 2 March 1999, ISBN 1-58113-086-4
- [8] David Gelertner, Nicholas Carriero - *“Coordination languages and their significance”*, Communications of the ACM, 35(2), 1992, 96-107, ISSN 0001-0782;

-
- [9] Andrea Omicini, Franco Zambonelli - “*Coordination for Internet Application Development*”, *Autonomous Agents and Multi-Agent Systems* 2(3), September 1999;
- [10] Andrea Omicini, Stefano Mariani - “*The TuCSoN Coordination Model and Technology. A Guide*”, TuCSoN v. 1.10.3.0206, Guide v. 1.0.2, 9 January 2013;
<http://www.slideshare.net/andreaomicini/the-tucson-coordination-model-technology-a-guide>
- [11] Enrico Denti, Antonio Natali, Andrea Omicini - “*On the expressive power of a language for programming coordination media*”, in: *Proceedings of the 1998 ACM Symposium on Applied Computing (SAC '98)*, Atlanta (USA) 27 February - 1 March 1998;
- [12] Andrea Omicini - “*Towards a notion of agent coordination context*”, in: Dan C. Marinescu and Craig Lee, *Process Coordination and Ubiquitous Computing*, chapter 12 pages 187-200, USA, October 2002;
- [13] Matteo Casadei, Andrea Omicini - “*Situated Tuple Centres in Re-SpecT*”, in: *24th ACM Symposium on Applied Computing*, Waikiki Beach, Honolulu, HI, USA, 10 March 2009;