Alma Mater Studiorum · Università di Bologna

SECONDA FACOLTÁ DI INGEGNERIA - SEDE DI CESENA
Corso di Laurea in Ingegneria Informatica

Artefatti e Sistemi Coordinati: Esperimenti in ReSpecT

Elaborato in Sistemi Distribuiti LA

Relatore:
Chiar.mo Prof. Andrea Omicini

Presentata da:
Noemi Ciampelli

Correlatore:
Dott. Stefano Mariani

Sessione II
Anno Accademico 2011-2012

Indice

Introduzione

Gli agenti non sono soli in un sistema multi-agente (MAS): essi interagiscono con altri agenti e con l'ambiente circostante.
Gli agenti, la società e l'ambiente sono le categorie di base per interpretare e modellare la struttura e le dinamiche di un MAS.
Le attività umane in organizzazioni complesse non possono essere semplicemente modellate in termini di espressioni e parole: un paesaggio articolato di oggetti, strumenti e tools abilitano e vincolano il corso delle nostre azioni nella società umana; tali entità distribuite nello spazio influiscono profondamente sui nostri processi cognitivi.
Allo stesso modo, gli agenti vivono in un environment complesso popolato da entità che non sono agenti e che servono per gli scopi degli agenti: sorgenti della conoscenza come database o pagine web, risorse modellate come web services, tools fisici come sensori e attuatori e così via. Queste entità sono dette artefatti.
Un artefatto è una sorta di device, un'entità che risiede nell'environment indipendentemente dal ciclo di vita degli agenti e fornisce servizi per permettere agli agenti di interagire con le risorse software e hardware e per partecipare alle attività sociali. Tali servizi sono tipicamente legati alla comunicazione, coordinazione, interoperabilità, organizzazione e così via: essi sono generalmente legati alla mediazione tra gli agenti e le altre astrazioni (entità e attività) e sono utilizzati per raggiungere goal individuali o collettivi.

Quando si guarda alle scienze computazionali, nozioni generali di tools e artefatti sono ovunque e abilitano la comprensione delle attività sociali, la rappresentazione e la costruzione dell'environment strutturato e la modellazione di processi cognitivi non-trivial.
Secondo questa nuova prospettiva, i MAS sono modellati ed ingegnerizzati basandosi su due astrazioni computazionali fondamentali, agenti e artefatti. Gli agenti sono le entità pro-attive che incapsulano il controllo e devono realizzare i goal che tutti insieme definiscono e determinano il comportamento dell'intero MAS, mentre gli artefatti sono le entità passive, reattive che

forniscono i servizi e le funzioni che permettono ai singoli agenti di lavorare insieme nel MAS e di modellare il MAS environment in accordo alle esigenze del MAS.
Il meta-modello risultante agenti & artefatti (A&A) promuove la modellazione e l'ingegnerizzazione delle agent societies e del MAS environment come entità di prima-classe.

Scopo della tesi è portare il meta-modello A&A su un'astrazione generale come il centro di tuple TuCSoN, indicando le scelte di progettazione per implementare gli artefatti e progettando le reazioni ReSpecT del centro di tuple in risposta alle richieste che possono essere fatte agli artefatti.

La tesi, quindi, è strutturata in questo modo:

- nel capitolo 1 si analizza la nozione di artefatto nelle varie discipline scientifiche per portarla nell'ambito MAS; inoltre si analizza il meta-modello A&A ed, in particolare, gli artefatti cognitivi A&A;
- nel capitolo 2 si descrivono i centri di tuple TuCSoN e il loro linguaggio di reazione fondamentale ReSPecT e come essi sono reconcepiti in base al meta-modello A&A;
- nel capitolo 3 si discutono le scelte progettuali e si tenta l'implementazione in ReSpecT del comportamento del centro di tuple in risposta alle varie richieste rivolte agli artefatti

Capitolo 1

Artefatti e meta-modello A&A

In questo capitolo dapprima si analizza la nozione di artefatto nelle varie discipline scientifiche per riutilizzare i risultati ottenuti nel campo dei sistemi multi-agente (MAS).
La nozione di artefatto portata nei MAS fornisce una nuova prospettiva: i MAS sono basati su due astrazioni di base - agenti e artefatti - e modellati secondo il meta-modello A&A (agenti ed artefatti).
Nella seconda parte del capitolo, quindi, si definisce tale meta-modello, concentrandosi in particolare sugli artefatti cognitivi A&A e sulle loro proprietà principali.

1.1 Multidisciplinarietà

La nozione di sistemi complessi attraversa i confini di molte discipline scientifiche differenti, passando dalla fisica alla biologia, dall'economia alla sociologia e alle scienze organizzative – e ovviamente influiscono anche sui sistemi computazionali. La multi-disciplinarietà ci porta a riconoscere che esistono delle "Leggi di Complessità" che caratterizzano ogni sistema complesso, indipendentemente dalla sua specifica natura [1]. Non importa se stiamo modellando il comportamento di un'organizzazione umana, la vita di un intricato ecosistema o le dinamiche di un enorme mercato: ci aspettiamo di trovare alcuni pattern ripetuti, alcuni schemi condivisi, alcune leggi comuni che rendono questi sistemi simili quando guardati dal giusto livello di astrazione. Di conseguenza, quando ci si focalizza su un MAS, adottare una visione multi-disciplinare per capire la sua complessità è quasi obbligatorio oltre che utile: un MAS infatti potenzialmente esibisce tutte le caratteristiche tipiche di un sistema complesso cioè openess, situatedness e località in controllo ed interazione.

In questa sezione osserviamo la pervasività della nozione di artefatto attraverso un gran numero di diverse aree di ricerca e portiamo i risultati nel campo dei MAS.

1.1.1 Activity theory

L'activity theory è un framework molto generale per concettualizzare le attività umane – come le persone imparano, come la società evolve – basato sul concetto di attività umana come unità fondamentale di analisi. Ogni attività portata avanti da uno o più partecipanti non può essere capita senza considerare i tools (o artefatti) che abilitano le azioni e mediano le interazioni tra i componenti.
Tali artefatti da un lato mediano l'interazione sia tra gli individui sia tra gli individui e l'ambiente; dall'altro, inglobano la parte di ambiente che può essere progettata e controllata per supportare le attività dei partecipanti.
La nozione di artefatto quindi è centrale nella AT: è il mediatore per ogni tipo di interazione nelle attività umane. Gli artefatti possono essere sia fisici (mensole, porte, telefoni) sia cognitivi (procedure operative, esperienze individuali o collettive, linguaggi) o essere di entrambi i tipi, fisico e cognitivo (manuali operativi, computer).
Essendo tool di mediazione, gli artefatti hanno sia una funzione abilitativa che una vincolante: espandono la possibilità di manipolare e trasformare diversi oggetti, ma li fanno percepire e manipolare non così come sono, ma con le limitazioni imposte dal tool.
L'AT si focalizza sulle attività sociali che possono essere coordinate, cooperative e co-costruite [2, 3].

- L'aspetto della coordinazione cattura il normale flusso di interazione in un sistema, dove il significato e i goal delle attività di collaborazione sono dati e stabili.

- L'aspetto della cooperazione concerne la modalità di interazione in cui gli attori si focalizzano su un oggetto comune e condividono l'obiettivo dell'attività: in questo caso l'oggetto dell'attività è stabile e approvato, mentre i modi per realizzare l'attività non sono ancora definiti.

- L'aspetto di co-costruzione concerne le interazioni in cui gli attori si focalizzano sul reconcettualizzare la loro stessa organizzazione e l'interazione coi loro oggetti condivisi: né l'oggetto del lavoro né i mezzi sono stabili e devono essere costruiti collettivamente, cioè co-costruiti.

Adottando l'AT come framework concettuale per le attività sociali in un MAS, si giunge alla conclusione che l'agente non è l'unica astrazione basilare per pensare e costruire un MAS [4]: anche gli artefatti sono necessari per abilitare e vincolare le azioni degli agenti, per mediare l'interazione tra agenti e con l'environment, per modellare e configurare il MAS environment e, più in generale, per affinare le abilità degli agenti nel raggiungere i goal sociali e individuali [5].
I tre livelli identificati dall'AT per le attività sociali possono essere reinterpretati in un contesto MAS in termini di relazioni tra agenti e artefatti.

- Co-costruzione: gli agenti esaminano e capiscono gli obiettivi sociali di un MAS e costruiscono un modello di attività sociali da svolgere per realizzare i goal. Ciò porta a interdipendenze e interazioni che vanno affrontare e gestite.

- Cooperazione: gli agenti progettano e costruiscono gli artefatti più adatti a portare avanti le attività sociali e a gestire le interdipendenze e le interazioni concepite al precedente stage.

- Coordinazione: gli agenti usano gli artefatti. Le attività volte al gestire le interdipendenze e le interazioni (progettate a priori o programmate nel livello di cooperazione) sono rinforzate o automatizzate.

1.1.2 Cognizione distribuita

La cognizione distribuita [6] è una branca delle scienze cognitive e propone che le cognizioni umane e la rappresentazione della conoscenza, invece di essere confinate all'interno dell'individuo, siano distribuite tra gli individui, i tool e gli artefatti nell'environment.
Secondo la cognizione distribuita, i processi intelligenti nelle attività umane vanno oltre i confini degli attori individuali e la conoscenza non è confinata nelle menti individuali. In particolare, è riconosciuto che la rappresentazione della conoscenza non appartiene solo agli individui umani: essa è distribuita in parte nello spazio mentale degli umani, in parte come rappresentazione esterna di memorie, fatti e informazioni di ogni tipo distribuiti su oggetti, tools e strumenti che costituiscono l'environment.
La cognizione distribuita quindi si concentra sui sistemi distribuiti cognitivi, dove gli umani interagiscono con altri umani e con artefatti cognitivi esterni. Gli artefatti cognitivi sono prodotti dei progetti e del lavoro umano, volti ad aiutare e accrescere le nostre capacità cognitive: ad esempio, post-it, agende, calendari, computer, ecc. Comunque, gli artefatti cognitivi non amplificano

semplicemente le nostre capacità cognitive: modificano anche il modo in cui eseguiamo le attività.
Ci sono due possibili modi per guardare ai sistemi distribuiti cognitivi: la visione di sistema e quella personale.
Nella visione di sistema, individui ed artefatti sono presi insieme come unità di osservazione per capire attività ed azioni; nella visione personale, invece, gli individui sono presi come unità di osservazione per capire come gli artefatti influiscono su di essi – il loro modo di ragionare, il modo in cui essi rappresentano le azioni, piani, aspettative, intenzioni, ecc.
Nei sistemi distribuiti cognitivi l'environment ha un ruolo chiave. La natura dell'environment – che dipende dagli artefatti e dai tools che lo configurano – determina l'efficacia del lavoro e delle attività degli attori che sono immersi in esso.
Come modellare uno work environment e come definire i suoi confini e la sua struttura dipende in gran parte dalle attività che dovranno essere portate avanti al suo interno; inoltre progettare uno work environment richiede una profonda understanding delle attività sociali ed individuali che vengono portate avanti in esso, degli individui in carica delle azioni e i loro corrispondenti processi cognitivi, della interazione sociale e così via.
L'obiettivo di un'attività non è sempre e solo quello cambiare l'environment in modo che presumibilmente porterà alla soddisfazione dei goal: le persone fanno azioni anche per presentare attenzione o ricordare certe cose e usano elementi esterni per ridurre l'utilizzo di sforzi cognitivi. Ciò è possibile solo se le persone e le loro attività sono situated cioè strettamente accoppiate con l'environment. Quindi il progetto dell'environment è volto ad aiutare le persone a soddisfare i loro goal, ma anche a rendere altre azioni facili.
Portando i risultati della cognizione distribuita all'ambito MAS, riconosciamo subito che la cognizione e la rappresentazione della conoscenza non deve appartenere solo agli agenti: i processi cognitivi che avvengono in un MAS non devono appartenere agli agenti individuali. Il MAS environment deve partecipare a questi processi, abilitando e mediando le azioni individuali degli agenti e l'interazione sociale. Quindi la cognizione e la rappresentazione della conoscenza sono distribuite nell'environment e influiscono su qualunque processo cognitivo. Gli artefatti sono parti essenziali dei processi cognitivi nei MAS: in particolare gli artefatti cognitivi incapsulano conoscenza rappresentata esplicitamente e la rendono disponibile agli agenti cognitivi.

Le visioni proposte dalla cognizioni distribuita (visione personale e visione di sistema) suggeriscono due modi per guardare alle relazioni tra agenti e artefatti.

La visione dell'agente è la visione dell'agente individuale circa gli artefatti e le azioni in un MAS: una volta che gli artefatti sono utilizzati, essi cambiano il modo in cui gli agenti agiscono e ragionano sulle azioni. La visione MAS, invece, è la visione sociale/globale circa le azioni degli agenti e l'interazione in un MAS: per capire e valutare le azioni sociali degli agenti in un MAS, gli agenti e gli artefatti vanno considerati insieme, come sottosistemi funzionali. Quindi, in accordo con la cognizione distribuita, gli artefatti configurano il MAS environment, la conoscenza è distribuita nell'environment e incapsulata negli artefatti cognitivi e la struttura dell'environment e la conoscenza che esso contiene influisce sulle attività degli agenti nel MAS. Inoltre, gli artefatti devono essere progettati non solo per aiutare le azioni degli agenti per raggiungere i goal, ma devono anche rendere più facili/efficaci le azioni epistemiche, complementari e coordinative.

1.1.3 Sociologia

Gli individui in un sistema sociale possono essere generalmente concepiti come sottosistemi goal-governed o goal-oriented [7].
I sistemi goal-governed presentano forme di capacità cognitive: rappresentano esplicitamente i loro goal e guidano la selezione delle attività degli agenti.
I sistemi goal-oriented, invece, sono direttamente progettati e programmati per raggiungere determinati goal, che non sono rappresentati esplicitamente.
In entrambi i sistemi i goal sono interni; i goal esterni, invece, tipicamente appartengono al contesto sociale o all'environment in cui si trovano gli agenti.
I goal esterni condizionano il comportamento degli agenti: i sistemi goal-governed seguono i goal esterni aggiustando quelli interni.
Il punto fondamentale è che non tutte le entità coinvolte nelle azioni sociali hanno goal interni: è il caso dei sistemi passivi (oggetti) che sono caratterizzati dal concetto di uso: non hanno goal interni però possono essere utilizzati dalle entità goal-driven per raggiungere i loro goal.
Gli oggetti sono esplicitamente progettati (o in seguito modificati) per fornire una determinata funzione: la funzione di un oggetto è un goal esterno ed influenza la progettazione della sua struttura e del suo comportamento, che dipende dall'uso che il progettista ha immaginato per quell'oggetto.
Il concetto di destinazione è legato ma non uguale al concetto di funzione: la destinazione riguarda il vero uso dell'oggetto. Quindi un oggetto può essere utilizzato con una destinazione che è diversa dalla sua funzione. Anche la destinazione è un goal esterno ma viene associato all'oggetto dall'utente al momento dell'utilizzo, invece la funzione è associata all'oggetto nel momento in cui esso è creato (o modificato).

Diversi tipi di goal esterni quindi sono associati ad un oggetto da un utente:

- Il goal use-value (la funzione): esprime le potenzialità dell'oggetto e quindi guida la scelta dell'oggetto
- Il goal use (la destinazione): corrisponde direttamente al goal interno dell'utente e indica il vero uso dell'oggetto

Un oggetto all'inizio è creato con una certa funzione in mente, poi viene scelto per un uso futuro da un utente se ha il goal use-value che fa match col suo goal interno. Quando infine è usato, la destinazione dell'oggetto può coincidere o meno con la sua funzione, ma comunque avrà il goal use che fa match con il goal interno dell'utente.

La nozione di goal e le sue diverse accezioni sono molto utili per caratterizzare gli artefatti nei MAS. In questa prospettiva, il MAS è costituito da due tipi di entità: gli agenti, che sono sistemi goal-oriented o goal-governed e gli artefatti, sistemi che non hanno goal interni.
Gli agenti hanno goal interni, la cui rappresentazione definisce la nozione di agency: gli agenti forti sono goal-governed e i goal sono rappresentati esplicitamente, gli agenti deboli sono goal-oriented e i goal sono rappresentati implicitamente.
Gli artefatti non hanno goal interni ma sono caratterizzati da una funzione, che è determinata a tempo di progettazione e dipende dal possibile uso dell'artefatto immaginato dal progettista. Quando l'artefatto è usato, gli viene associato un altro goal esterno: la sua destinazione.
La relazione tra agenti e artefatti assume tre distinte forme: uso, selezione e costruzione.
Quando usa un artefatto, un agente è guidato dal goal use e associa veramente un artefatto a una destinazione che fa match con uno dei suoi goal interni, che può essere diversa dalla funzione designata dell'artefatto.
Quando seleziona un artefatto per uso futuro, un agente immagina la sua possibile destinazione considerando il goal use-value.
Infine, quando costruisce un artefatto – sia che modifichi uno esistente o che ne costruisca uno nuovo – un agente immagina il suo possibile uso futuro e in base a ciò progetta la sua struttura e il suo comportamento.
Quindi, gli agenti associano diversi tipi di goal esterni agli artefatti: il goal use-value, che guida la selezione dell'artefatto da parte dell'agente, e il goal use, che guida il reale uso dell'artefatto da parte dell'agente. La costruzione invece è una conseguenza del fallimento nella selezione dell'artefatto o nell'uso dell'artefatto selezionato: un nuovo artefatto deve essere creato o ottenuto manipolandone uno esistente, così da permettere all'agente di raggiungere i suoi goal interni.

1.1.4 Computer Supported Cooperative Work (CSCW)

La CSCW [8] riguarda il modo in cui le tecnologie computazionali e gli artefatti abilitano e promuovono le attività collaborative delle persone.
La CSCW sembra voler seguire due strategie divergenti: la prima include un approccio che viene dai Workflow Management Systems e tende a privilegiare l'automatizzazione della coordinazione; la seconda si riferisce al contributo classico della CSCW e ha come goal principale la flessibilità dell'interazione. Quindi, l'approccio classico enfatizza il ruolo delle entità computazionali stabilendo le regole di coordinazione, l'approccio WfMS invece conta sulle capacità di coordinazione delle entità collaborative (come gli umani o gli agenti intelligenti). Quindi c'è un gap tra due diverse strategie e due diverse linee di ricerca ed è richiesta una convergenza.
I due punti chiave della soluzione del problema sono la mutua consapevolezza e gli artefatti coordinativi.
La mutua consapevolezza significa che gli attori di un'attività di collaborazione influenzano e mutuamente percepiscono le attività degli altri attori attraverso lo spazio di lavoro comune – il workspace condiviso – che manifesta o cela ai partecipanti porzioni delle attività di collaborazione. Essa quindi è la base per assicurare la flessibilità delle attività di collaborazione.
Gli artefatti coordinativi, invece, sono le regole della coordinazione: incapsulano quelle porzioni di coordinazione che è meglio automatizzare per garantire l'efficienza della cooperazione. Quindi da una parte gli artefatti coordinativi definiscono e governano lo spazio delle attività ammesse, dall'altra non impongono un predefinito corso delle azioni che può causare rigidità non necessarie e ridurre la flessibilità richiesta.

La CSCW ci fornisce le linee guida su come un artefatto può essere usato in modo fruttifero in un MAS , promuovendo l'automatizzazione e la flessibilità delle azioni di coordinazione.
L'automatizzazione è delegata agli artefatti coordinativi, detti artefatti di coordinazione nella tipica terminologia MAS. Essi compongono e danno struttura al campo di lavoro comune di un MAS, automatizzando la collaborazione e rendendola efficiente. Come nella CSCW, da una parte essi definiscono e governano lo spazio delle attività di collaborazione del MAS ammesse, dall'altra non impongono un predefinito corso delle azioni degli agenti in un MAS, per non impedire la flessibilità della coordinazione degli agenti intelligenti e per rispettare l'autonomia degli agenti.
Come nella CSCW,la flessibilità è promossa anche dalla mutua consapevolezza. Il MAS environment condiviso può essere strutturato come campo di lavoro comune MAS per permettere agli agenti di percepire mutuamente

ogni altra attività: il campo di lavoro comune MAS manifesta o cela agli agenti porzioni delle attività di collaborazione MAS. Essa quindi assicura la flessibilità delle attività di collaborazione MAS.

1.1.5 Antropologia ed etologia

L'antropologia occidentale per lungo tempo ha vissuto con l'idea che l'uso di un linguaggio simbolico fosse il maggior segno di intelligenza umana. Questo provocò una profonda deviazione filosofica [9] da parte degli studiosi occidentali, che li condusse a dimenticare la relazione tra intelligenza ed uso degli artefatti: infatti essi pensavano che la facoltà di costruire e usare tools fosse un attributo prevedibile e quindi non richiedesse esami scientifici.
Dopo molte decadi di attività di ricerca in molte aree diverse ma legate (biologia, antropologia sociale, archeologia, linguistica, psicologia, neurologia ed etologia), solo in tempi molto recenti il punto fondamentale della relazione tra linguaggi, tools ed evoluzione delle capacità cognitive umane è stato visto come un unico, coerente problema [10].
Al giorno d'oggi, è di larga conoscenza che la capacità umana di sviluppare e usate tools è un segno fondamentale di intelligenza, almeno quanto usare un linguaggio.
La prima caratteristica dell'Homo Abilis è infatti la sua abilità di usare e forgiare tools. Questo è un chiaro segno di intelligenza e l'evidenza della co-evoluzione dell'uso dei linguaggi e dei tools e l'umana intelligenza è schiacciante negli studi antropologici moderni [10].
I tools non sono una caratteristica esclusivamente umana: i castori costruiscono dighe, le api costruiscono celle perfettamente esagonali e gli uccelli si costruiscono il nido.
Comunque, ciò che è spesso preso come caratteristica distintiva dei tools umani rispetto a quelli degli animali è il livello cognitivo al quale i tools sono concepiti, progettati e usati: apparentemente i tools non sono parte del comportamento umano, come lo sono per le api o gli uccelli, ma sono invece il risultato di un'elaborazione razionale riguardante la relazione tra l'essere umano e il suo habitat – il suo environment.
Quindi, progettazione e uso sistematico e sociale di tools sembra tipico della specie umana e spesso è presa come misura dell'intelligenza umana rispetto a quella degli animali.
Più in generale, la nostra conoscenza della stretta relazione tra tools e intelligenza (umana e non) è tale che noi interpretiamo la facoltà di usare e costruire tools come sintomo rilevante di intelligenza. Per esempio, gli etologi comunemente misurano l'intelligenza degli animali ponendogli dei problemi che necessitano di tools per essere risolti [11].

I tools, quindi, sono allo stesso tempo la prima e la più distintiva espressione dell'intelligenza umana, insieme al linguaggio; inoltre, sono i più potenti amplificatori dell'abilità umana di influire sull'ambiente – prima di tutto per sopravvivere ai cambiamenti dell'ambiente e successivamente per cambiare l'ambiente per gli scopi umani.

Guardando i sistemi ad agenti dal punto di vista dell' antropologia e dell'etologia, in particolare l'intelligenza degli agenti, si nota che il lavoro finora svolto si basa sull'elaborazione di atti e concetti linguistici ed è praticamente ignorata, almeno esplicitamente, la materia dei tools. Quindi per evitare la deviazione filosofica che avevamo riscontrato nell'antropologia anche nel campo MAS, una nozione di agent tool o artefatto è richiesta in modo che la teoria delle azioni fisiche degli agenti cresca allo stesso livello della teoria delle azioni di comunicazione. Tale teoria deve permettere una più larga definizione di intelligenza degli agenti , che includa non solo l'uso del linguaggio simbolico ma anche l'uso dei tools.

1.2 Il meta-modello A&A

In questa sezione viene introdotto il meta-modello A&A e si definiscono le nozioni di agenti, artefatti e sistemi multi-agente. Esse non sono nuove definizioni che vanno ad aggiungersi o a modificare le molte esistenti; piuttosto forniscono la precisa accettazione di questi termini nel contesto A&A.

Il meta-modello A&A [12] è caratterizzato da tre astrazioni di base:

- Agenti: componenti pro-attivi del sistema, che incapsulano l'esecuzione autonoma di attività all'interno di un ambiente
- Artefatti: componenti passivi del sistema come risorse e media che sono intenzionalmente costruiti, condivisi, manipolati e usati dagli agenti per supportare le proprie attività, sia cooperativamente che competitivamente.
- Sistemi multi-agente: sistemi computazionali il cui ambiente è organizzato in workspaces, contenitori concettuali di agenti ed artefatti.

1.2.1 Agenti A&A

Secondo la definizione formale data dal meta-modello A&A, un agente è un'entità computazionale autonoma.

Il meta-modello A&A, quindi, prima di tutto, riconosce l'autonomia come caratteristica fondamentale per gli agenti.
Da un punto di vista computazionale, autonomia significa che gli agenti incapsulano il controllo. Il controllo, quindi, non esce dai confini dell'agente: gli agenti non cedono mai il controllo né sono mai controllati – a meno che non lo decidano essi stessi, ovviamente.
Di conseguenza gli agenti non forniscono interfacce per essere usati e non possono essere invocati o controllati. Solo dati (conoscenza, informazioni) attraversano i confini degli agenti.
Come risultato, i MAS sono visti come una molteplicità di luoghi distinti di controllo, che interagiscono con gli altri scambiando informazioni.
Dalla nozione di autonomia emerge anche la nozione di agent sociality. Da un punto di vista filosofico, infatti, l'autonomia ha senso solo quando un individuo è immerso in una società – un individuo da solo non si può dire autonomo, questo termine non ha senso per un individuo isolato.
Letteralmente, l'etimologia della parola agente – dal latino "agens" – significa "colui che agisce". Quindi, gli agenti autonomi sono intrinsecamente entità attive: più precisamente, essi sono pro-attivi, dato che incapsulano il controllo e auto-governano il proprio corso delle azioni.
Proattivo significa "fare succedere qualcosa" invece di "aspettare che succeda qualcosa".
Per dare una coerente nozione di agente, è necessario un modello concettuale che definisca le azioni degli agenti. Qualunque sia il modello, ogni nozione di azione è intrinsecamente connessa alla nozione di cambiamento: un agente agisce per cambiare qualcosa.
Nel contesto dei sistemi multi-agente l'azione di un agente ha potenzialmente due target: o un altro agente o l'ambiente. L'azione di un agente, quindi, punta a cambiare o un altro agente o l'ambiente del MAS, cioè il luogo concettuale dove gli agenti di un MAS vivono e interagiscono.
Dato che gli agenti sono autonomi e solo dati possono attraversare i loro confini, l'unico modo per influenzare lo stato di un altro agente è fornirgli delle informazioni: queste azioni sono dette speech act o communication action.
I cambiamenti nell'ambiente, invece, possono essere fatti più facilmente con azioni fisiche.
Il modello base delle azione è strettamente accoppiato al luogo in cui le azioni avvengono, dipende dal contesto in cui gli agenti agiscono. In questo senso, gli agenti autonomi sono essenzialmente delle entità "situated" dato che ogni agente è strettamente accoppiato con l'ambiente in cui vive e interagisce.
Altre caratteristiche fondamentali sono spesso attribuiti agli agenti come l'intelligenza, la mobilità o la capacità di imparare. Comunque, anche se

queste sono proprietà desiderabili, non sono essenziali per gli agenti nel meta-modello A&A.

1.2.2 Artefatti A&A

Secondo la definizione formale data dal meta-modello A&A, un artefatto è un'entità computazionale volta all'uso da parte degli agenti.
Prima di tutto, quindi, gli artefatti sono fatti per essere usati dagli agenti. A partire dall'uso emergono altre caratteristiche, essenziali o desiderabili.
Gli artefatti non sono autonomi, a differenza degli agenti. Dato che essi sono progettati per servire gli scopi degli agenti, non possono seguire il proprio corso delle azioni.
Un artefatto è un tool nelle mani degli agenti e non ha quindi bisogno di autogovernarsi; piuttosto è governato dall'uso dell'agente: gli artefatti sono computazionalmente reattivi, sono reattivi in termini di controllo. Essi agiscono in risposta all'uso degli agenti e il loro comportamento ha bisogno di emergere solo quando sono usati dagli agenti.
Ogni artefatto ha una funzione, che gli viene associata a tempo di progettazione. Non è poi detto che l'artefatto verrà veramente usato dagli agenti per quella funzione, però incorpora lo scopo del progettista, che ha previsto che quell'artefatto potesse potenzialmente servire a raggiungere determinati scopi.
Essendo volti all'uso da parte degli agenti, gli artefatti sono il target primario delle azioni degli agenti. Anche le loro funzioni sono espresse in termini di cambiamenti dell'ambiente, cioè cosa devono veramente fare gli artefatti quando sono usati dagli agenti. Quindi il modello, la struttura e il comportamento degli artefatti sono espressi in termini di azioni degli agenti e del loro effetto sull'ambiente: ciò rende gli artefatti intrinsecamente situated.
Quindi, dato che sono situated e strutturalmente reattivi in termini computazionali, è facile pensare agli artefatti come reattivi ai cambiamenti nell'ambiente. Per essere usati, gli artefatti dovrebbero rendere disponibili per gli agenti delle operazioni. Le operazioni cambiano lo stato di un artefatto, fanno sì che esso reagisca e produca gli effetti desiderati sull'ambiente.
Esplicitamente o implicitamente, quindi, un artefatto esibisce la sua interfaccia, come collezione di operazioni disponibili per l'uso da parte degli agenti.
Infine, per promuovere l'uso da parte di agenti intelligenti, la funzione dell'artefatto dovrebbe essere disponibile per gli agenti e comprensibile da parte loro.
Altresì, il comportamento di un artefatto dovrebbe essere prevedibile, in modo da essere usato efficientemente dagli agenti.

Trasparenza e prevedibilità del comportamento, quindi, sono caratteristiche desiderabili per gli artefatti A&A.

1.2.3 Sistemi multi-agente A&A

Secondo la definizione formale data dal meta-modello A&A, un MAS è un sistema computazionale fatto di agenti e artefatti, organizzati in workspaces. Nel complesso, il comportamento del MAS A&A risulta dall'interazione tra entità autonome e auto-gestite (agenti) ed entità reattive e funzionali (artefatti). Dato che sia gli agenti sia gli artefatti sono entità computazionali situated, i MAS A&A sono sistemi computazionali situated: un MAS è sempre immerso in un ambiente e non può concepire/modellare/progettare in un modo separato rispetto al suo ambiente.
L'ambiente del MAS è detto working environment ed è concepito come un set dinamico di artefatti e agenti organizzati in workspaces. I workspaces sono contenitori logici, utili per definire la topologia dello working environment. Inoltre, forniscono una nozione di località per gli agenti: gli agenti possono lavorare solo con artefatti appartenenti al loro stesso workspaces, però concettualmente possono essere situtati su più workspaces allo stesso tempo.
Lo spazio delle interazioni ammissibili in un MAS è generato dai 4 tipi di interazione che possono potenzialmente avvenire tra le due entità fondamentali che compongono il MAS:

- comunicazione: gli agenti parlano con gli agenti
- operazione: gli agenti usano gli artefatti
- composizione: gli artefatti sono collegati ad altri artefatti
- presentazione: gli artefatti si manifestano agli agenti

Le possibili interazioni nel meta-modello A&A sono queste, più l'interazione con l'ambiente del MAS: questa interazione è attribuita ai singoli agenti/artefatti o al sistema nel suo insieme, dipende dal livello di astrazione desiderato/richiesto.

1.3 Artefatti cognitivi A&A

Una volta che la nozione di uso è stata introdotta come fondamentale nella definizione di artefatto e nella relazione tra agenti e artefatti, la nozione di uso cognitivo è la questione più rilevante che deve essere indirizzata.

Un agente equipaggiato con abilità cognitive (intelligenza, razionalità) può valutare gli artefatti disponibili, selezionare quello più adatto alle sue esigenze ed usarlo efficacemente ed efficientemente per raggiungere i suoi goal.
Per capire effettivamente gli outcomes che si possono ottenere usando un artefatto, un agente deve essere in qualche modo conscio della struttura e del comportamento dell'artefatto, del modo in cui va usato e degli effetti che otterrà.
A questo fine, conoscenza riguardo alla progettazione, struttura e comportamento dell'artefatto dovrebbe essere immessa nell'artefatto stesso, in modo da renderla disponibile all'ispezione degli agenti.
La definizione di artefatto nel meta-modello A&A - il cui target erano anche gli agenti non cognitivi oltre a quelli cognitivi - non prevedeva questa conoscenza quindi è richiesta una nozione di artefatto più ricca, con caratteristiche addizionali che possono elevare gli artefatti al livello cognitivo degli agenti: l'artefatto cognitivo [12].

In questa sezione quindi si analizza la nozione di artefatto cognitivo A&A e le sue proprietà.

Un artefatto cognitivo A&A è un artefatto che aspira ad un uso cognitivo da parte degli agenti.
Un uso cognitivo significa che gli agenti lo usano razionalmente – agendo in modo conforme alle task che devono eseguire o ai goal che devono perseguire – e con la consapevolezza dell'esistenza e della disponibilità all'accesso dell'artefatto, dei benefici derivanti dall'uso e della sua struttura e comportamento.

1.3.1 Proprietà degli artefatti cognitivi A&A

Le proprietà basilari di un artefatto cognitivo nel meta-modello A&A [12] sono:

- Function description: gli outcomes attesi quando viene usato l'artefatto, cioè la funzione che l'artefatto fornisce agli agenti, secondo le intenzione del progettista

- Operating instructions: il comportamento che deve caratterizzare l'interazione dell'agente con l'artefatto, cioè le procedure ammissibili per usare un artefatto per un determinato scopo

- Usage interface: i dettagli delle operazioni fornite dall'artefatto, cioè la struttura esterna e osservabile dell'artefatto.

Queste tre proprietà vengono usate dagli agenti in ordine: prima per selezionare l'artefatto adatto, poi per pianificare lo sfruttamento dell'artefatto selezionato e infine per interagire in modo appropriato con l'artefatto per eseguire un'operazione.

1.3.1.1 Selezionare un artefatto mediante le function description

In un sistema multi-agente A&A, un agente vive in un ambiente popolato da artefatti e da altri agenti: agendo all'interno e su tale ambiente, vengono eseguite le task e ottenuti i goal. In particolare, dato che gli artefatti possono servire a diversi scopi, un agente cognitivo dovrebbe essere capace di capire completamente tutte le opportunità offerte da ogni artefatto disponibile, in modo da scegliere quello adatto. Ogni artefatto ha una funzione determinata durante la progettazione, che rappresenta il motivo originario per cui l'artefatto è stato costruito: essa determina il cosiddetto use-value dell'artefatto, che rappresenta la ragione per cui un agente dovrebbe scegliere tale artefatto tra tutto l'insieme degli artefatti disponibili. In accordo a ciò, nel metamodello A&A un artefatto cognitivo è corredato da una function description che definisce la funzione intenzionale che l'artefatto rende disponibile per l'uso degli agenti. Essenzialmente, la function description contiene informazioni riguardo i possibili usi che il progettista aveva immaginato per l'artefatto. Il modo in cui la function description di un artefatto cognitivo è veramente fornita agli agenti può variare, per esempio in sintassi e semantica – lo stesso vale per le operating instructions e l'usage interface. Questo perché la scelta del modo più appropriato per rappresentare la function description è influenzata da molto fattori, come lo scenario applicativo, l'esistenza di un'ontologia comune a tutti gli agenti, il grado di apertura del sistema multi-agente e il livello di cognizione degli agenti. Infatti, lo stesso sistema multi-agente può ospitare artefatti con diverse forme (modelli, linguaggi, tecnologie) per la function description. Per esempio, la function description di un artefatto cognitivo può essere data in termini di goal che si possono ottenere utilizzandolo o di una lista di task che si possono eseguire con esso. Queste informazioni dovrebbero essere accoppiate con delle pre-condizioni mentali dell'agente in modo da funzionare come filtro per quegli agenti che non hanno ragione di usare l'artefatto.

1.3.1.2 Utilizzare un artefatto mediante le operating instruction

Una volta che l'artefatto è stato scelto, l'agente dovrebbe essere al corrente di come debba essere usato per lo scopo selezionato. In modo similare ad un manuale per i device fisici, le operating instructions descrivono il com-

portamento che un agente dovrebbe seguire per interagire significativamente con l'artefatto, cioè per usarlo efficacemente in base alla destinazione scelta dall'agente, in altri termini lo use-goal selezionato.
Le operating instructions quindi devono essere scritte in un linguaggio che gli agenti possano capire: la semantica di tale linguaggio definisce precisamente le sequenze di azioni degli agenti sull' l'artefatto ammesse/richieste o, equivalentemente, un set di piani sintetici per usare efficacemente l'artefatto.
Il goal delle operating instructions è quindi quello di assistere gli agenti quando pianificano il loro corso delle azioni quindi, una volta che l'artefatto è stato selezionato, le operating instructions sono controllate e interiorizzate dagli agenti per pianificare le loro attività.
L'outcome è un piano che incorpora delle sequenze di operazioni definite come ammissibili dalle operating instructions dell'artefatto.
Ancora una volta, possono essere utilizzati diversi linguaggi.
I linguaggi per le operating instructions possono essere completamente operazionali (essi mirano a descrivere il comportamento passo-passo e tipicamente indicano le pre-condizioni e gli effetti delle operazioni degli artefatti); oppure può essere utilizzato un approccio più dichiarativo o un approccio logic-based (adatto per agenti con capacità deduttive intrinseche).

1.3.1.3 Agire su un artefatto mediante l'usage interface

Data la funzione a cui l'agente è interessato, le operating instructions descrivono il comportamento ammissibile/rilevante, in termini di sequenze di operazioni dell'artefatto da eseguire – azioni sull'artefatto e percezioni di eventi rilevanti. Per questo fine, ogni artefatto è corredato da una descrizione della sua interfaccia d'uso, cioè il set di tutte le operazioni rese disponibili dall'artefatto per l'uso degli agenti.
Per ogni operazione l'interfaccia d'uso descrive come deve essere invocata e come i suoi risultati possono essere percepiti dagli agenti.
Ogni operazione è descritta in termini del suo nome, dei suoi argomenti (nome e tipo) e dei suoi outcomes.
Un agente esegue un'azione sull'artefatto specificando il nome dell'operazione e fornendo gli argomenti adatti – questi sono i dati che fluiscono all'esterno dell'agente. Gli outcomes descrivono gli eventi generati dall'artefatto mentre serve l'operazione, che gli agenti possono percepire con il sensing – questi sono i dati che fluiscono all'interno dell'agente. Questi eventi possono segnalare quando l'operazione è veramente attivata, se è completata con successo e quali sono i risultati, se e perché è fallita e se l'artefatto ha raggiunto uno stato a cui l'agente potrebbe essere interessato.

Le interfacce d'uso possono essere sia statiche che dinamiche: il set e la configurazione delle operazioni ammissibili può essere stabile o può variare in base allo stato interno dell'artefatto. Inoltre, essa può organizzare le operazioni in classi distinte come una specie di interfacce specializzate: ad esempio, classi usate per controllare direttamente lo stato interno dell'artefatto, classi per amministrare il ciclo di vita dell'artefatto (accensione, cambiamento del comportamento, stopping e spegnimento) e così via [13].

1.3.2 Caratteristiche degli artefatti cognitivi ed intelligenza

In realtà, le tre caratteristiche distintive degli artefatti cognitivi non sono tutte obbligatorie per l'uso cognitivo di un artefatto.
Mentre la disponibilità della function description, delle operating instructions e dell'usage interface insieme supportano completamente gli agenti cognitivi nel selezionare razionalmente un artefatto, pianificare il suo uso ed interagire con esso, in altri scenari diversi un sottoinsieme di esse permette comunque altri livelli desiderabili di uso cognitivo.
Per esempio, la function description potrebbe non essere disponibile all'ispezione diretta dell'agente: un agente però potrebbe comunque intuire la funzione di un artefatto attraverso un processo trial-and-error o osservando le azioni degli altri agenti nel sistema multi-agente – e lo stesso per le operating instructions e l'usage interface.
Infatti, tutte le caratteristiche distintive di un artefatto cognitivo potrebbero essere definite implicitamente in un artefatto non cognitivo e gli agenti potrebbero venire a conoscenza di esse imparando o perché sono progettati per fare questo. Ciò che realmente caratterizza un artefatto cognitivo è che tali proprietà sono concepite quando l'artefatto è progettato, sono esplicitamente rappresentate in esso e quindi disponibili all'ispezione degli agenti cognitivi.
Come risultato, gli artefatti cognitivi supportano la costruzione di sistemi multi-agente realmente aperti , dove agenti cognitivi di ogni tipo possono entrare nel workspace del MAS, ispezionare gli artefatti disponibili e decidere di conseguenza , ottenere le function descriptions e selezionare degli artefatti per un uso futuro, ispezionare le operating instructions e pianificare il suo corso delle azioni e infine ottenere l'usage interface ed interagire efficacemente ed efficientemente con l'artefatto.

L'esistenza degli artefatti, e ancor più, degli artefatti cognitivi e le opportunità che forniscono agli agenti, introducono un nuovo modello di intelligenza degli agenti, dove l'intelligenza di un agente è misurata sia in termini della

sua abilità nel parlare e capire linguaggi simbolici di alto livello, sia in termini della sua abilità nell'usare artefatti – e possibilmente nel forgiarne di nuovi in base ai suoi bisogni.

1.3.3 Altre proprietà

Oltre alle caratteristiche basilari, gli artefatti cognitivi esibiscono ulteriori caratteristiche rilevanti [14]:

- Ispezionabilità: lo stato di un artefatto, il suo contenuto, la sua usage interface, le operating instructions e la function description dovrebbero essere totalmente o parzialmente disponibili agli agenti attraverso l'ispezionabilità. In particolare nei MAS aperti, gli agenti devono essere in grado di entrare dinamicamente nella società e di venire a conoscenza a run-time delle informazioni necessarie riguardo agli artefatti disponibili.

- Malleabilità: chiaramente legata all'ispezionabilità, la malleabilità è una caratteristica chiave negli scenari dinamici dei MAS, quando il comportamento dell'artefatto necessita di essere modificato dinamicamente per adattarsi ai cambiamenti delle necessità o alle mutevoli condizioni esterne del MAS. La malleabilità, come capacità di cambiare il comportamento dell'artefatto a tempo di esecuzione, è un punto chiave per i MAS auto-organizzanti.

- Predicibilità: differentemente dagli agenti (che come entità autonome hanno la libertà di comportarsi anche in modo sbagliato), l'usage inteface, le operating instructions e la function description possono essere usate dagli agenti come contratti con gli artefatti. In particolare, la function description fornisce dettagli precisi sugli outcomes che si otterranno usando l'artefatto, mentre le operating instructions rendono il comportamento dell'artefatto predicibile per un agente.

- Formalizzabilità: la predicibilità può essere facilmente relazionata alla formalizzabilità. Data la precisa caratterizzazione che si può dare al comportamento degli artefatti, è automaticamente possibile verificare anche le proprietà e il comportamento dei servizi da essi forniti.

- Linkabilità: aumentando la complessità dell'environment, può essere interessante comporre gli artefatti. Per esempio, si può costruire un nuovo servizio sulla base di un altro: si crea un nuovo artefatto che realizza i suoi servizi interagendo con un artefatto già esistente. A tale scopo, gli artefatti devono essere in grado di invocare le operazioni di

altri artefatti: la risposta a tale invocazione verrà trasmessa dal receiver invocando un'altra operazione del sender. La linkabilità permette la costruzione di articolati environment di artefatti interagenti. Tale complessità , comunque, non necessariamente impatta sulla percezione che hanno gli agenti dell'ambiente: molto probabilmente un agente sarà conscio di pochi artefatti, usati come front-end per un management possibilmente complesso di servizi.

- Distribuzione: differentemente dagli agenti, che tipicamente sono concettualmente allocati su un singolo nodo della rete, gli artefatti possono essere distribuiti. In particolare, un singolo artefatto può modellare un servizio distribuito, a cui si può accedere da più nodi della rete. Usando la linkabilità, un artefatto distribuito può essere concepito e implementato come una composizione di artefatti collegati possibilmente non distribuiti o, viceversa, un gran numero di artefatti collegati, sparsi attraverso un gran numero di luoghi fisici differenti, possono essere visti nell'insieme come un singolo artefatto distribuito.

Tutte le proprietà citate giocano un ruolo diverso a seconda del punto di vista: quello degli agenti o quello degli ingegneri del MAS.
Per esempio, le operating instructions sono viste come un tool di progettazione dagli ingegneri del MAS, mentre sono viste come un supporto a run-time dagli agenti razionali.
Invece, caratteristiche come ispezionabilità e malleabilità diventano rilevanti quando i due punti di vista diventano uno solo: quando un agente intelligente è capace (e gli è permesso) di giocare nel ruolo dell'ingegnere MAS, può capire lo stato e le dinamiche del MAS osservando gli artefatti e possibilmente modificare gli artefatti per cambiare il comportamento dell'intero MAS.

Capitolo 2

TuCSoN e ReSpecT nella prospettiva A&A

In questo capitolo viene introdotto il modello di coordinazione tuple-based e si osservano i motivi dell'evoluzione del suo medium di coordinazione da spazio di tuple a centro di tuple. Successivamento si descrivono il modello di coordinazione TuCSoN e il suo linguaggio di specifica ReSpecT e come essi vengono reconcepiti secondo la prospettiva A&A.
In particolare verrà analizzato A&A ReSpecT: le novità portate dalla prospettiva A&A e la sua semantica.

2.1 I modelli di coordinazione tuple-based

La coordinazione è una questione fondamentale nei sistemi distribuiti.
In tali sistemi vi sono componenti/flussi di controllo indipendenti attivi simultaneamente su contesti di esecuzione che sono dei più svariati che interagiscono tra di loro e con l'ambiente (risorse esterne e struttura time-space).
Le caratteristiche dei sistemi distribuiti sonol'apertura, la distribuzione e l'eterogeneità perciò, per quanto riguarda i componenti del sistema, non ci sono ipotesi sul loro tempo di vita e comportamento, sulla loro allocazione e mobilità e sulla loro natura e struttura. Per costruire il sistema, quindi, non è più possibile concentrarsi sui singoli componenti, bisogna focalizzare l'attenzione sullo spazio di interazione, dove i componenti verranno messi insieme ed interagiranno tra di loro e con lo spazio stesso.
È perciò necessario un modello di coordinazione da applicare a questo spazio di interazione che si occupi dell'entrata/uscita delle entità, della loro comunicazione e distribuzione spaziale e della sincronizzazione e distribuzione delle loro attività.

In questa sezione viene introdotto un tipo di modello di coordinazione: il modello tuple-based. Vengono osservate tutte le sue caratteristiche e si osserva l'evoluzione del suo medium di coordinazione da spazio di tuple a centro di tuple.

2.1.1 Introduzione del modello tuple-based

I modelli di coordinazione tuple-based sono basati su un meta-modello [15] secondo il quale i componenti di un sistema sono:

- i coordinabili, ovvero le entità da coordinare
- il medium di coordinazione, ossia lo spazio in cui i coordinabili vivono ed interagiscono
- le leggi di coordinazione, cioè le regole che definiscono il comportamento del medium in risposta all'interazione. Tali leggi sono espresse in termini di linguaggio di comunicazione - la sintassi usata per esprimere e scambiare strutture dati - e di linguaggio di coordinazione - set di primitive di interazione ammissibili e loro semantica.

Seguendo questo meta-modello viene definito il modello tuple-based: in esso il medium di coordinazione è lo spazio di tuple, ovvero un multiset di elementi/aggregati di informazione eterogenei detti tuple; il linguaggio di comunicazione è rappresentato dalle tuple e il linguaggio di coordinazione è costituito da primitive per inserire e recuperare tuple nello/dallo spazio di tuple. Questi due linguaggi fanno parte di un modello chiamato Linda [16].

Il linguaggio di comunicazione [16] del modello Linda ha tre punti fondamentali:

- le tuple: aggregati di informazione possibilmente eterogenei
- i tuple template: classi di tuple
- il meccanismo di matching: associazione tra un tuple template e una tupla

Il linguaggio di coordinazione [16], invece, è costituito dalle primitive a disposizione delle tuple per interagire col medium di coordinazione. Le primitive di base sono:

- *out(T)*: inserisce la tupla T nello spazio di tuple

- *in (TT)*: dato un tuple template TT, recupera dallo spazio di tuple una tupla che fa match con questo.

La primitiva *in* ha due proprietà importanti: la *lettura distruttiva* e la *semantica sospensiva*. Lettura distruttiva significa che la tupla recuperata è rimossa dallo spazio di tuple. Semantica sospensiva significa che se nessuna tupla fa match con il template, l'esecuzione dell'operazione è sospesa e viene risvegliata quando arriva una tupla che fa match.

Esistono altre primitive, invece, che hanno lettura non distruttiva o semantica non sospensiva, ad esempio:

- *rd(TT)*: questa primitiva fa la stessa cosa dell'*in(TT)* con la differenza che fa una lettura non distruttiva, cioè legge solo la tupla ma non la rimuove dallo spazio delle tuple.

- *inp(TT)* - *rdp(TT)*: sono primitive predicative, cioè sono vere o false, invece la *rd* e la *in* all'infinito avranno sempre successo, saranno sempre vere. *inp* e *rdp*, quindi, sono uguali a *in* e *rd* se nello spazio di tuple vi è una tupla che fa match con il template e restituiscono la tupla in questione; se invece la tupla non è presente nello spazio, la primitiva fallisce e restituisce falso. Le primitive predicative, quindi, hanno semantica non sospensiva.

- *in_ all(TT)* - *rd_ all(TT)*: restituiscono una lista di tuple, cioè tutte quelle che fanno match con il template TT. Non si ha semantica sospensiva perché, se non vengono trovate tuple, viene restituita la lista vuota. Per quanto riguarda la lettura distruttiva/non distruttiva *in_ all* rimuove dallo spazio tutte le tuple che fanno match, invece *rd_ all* no.

Esiste una proprietà comune a tutte le primitive cioè il *non determinismo*: se ci sono n tuple che fanno match con il tuple template, la tupla viene scelta a caso, in modo non deterministico.

Un'altra primitiva interessante è *ts@node ? primitive(p)* denota l'invocazione dell'operazione *primitive(p)* nello spazio di tuple *ts* sul nodo *node*.

Questa primitiva è legata all'introduzione degli spazi di tuple multipli: avere un unico centro che coordina milioni di entità in un sistema con una grande estensione geografica poteva essere un limite allora si sono introdotti gli spazi di tuple multipli e per gestire ciò serviva una primitiva che permettesse di specificare su quale spazio di tuple eseguire la primitiva e su quale nodo esso si trovasse.

Le caratteristiche principali del modello Linda sono le *tuple*, la *comunicazione generativa*, l'*accesso associativo alla conoscenza* e la *semantica sospensiva* [17].

Tuple

Le tuple sono collezioni ordinate di elementi di conoscenza, possibilmente eterogenei. Hanno struttura a record senza bisogno di nomi per i campi e rappresentano un facile modo per aggregare conoscenza.
Dalla sintassi non arriva nessun vincolo se non quello di essere delle liste di elementi eterogenei; l'interpretazione semantica, invece, dice che una tupla è un'unità concettuale che contiene tutte le informazioni riguardanti un dato elemento. La struttura di una tupla è basata su arità (numero di elementi), tipo, posizione e contesto informativo.
Legati alle tuple, sono molto importanti i concetti di anti-tupla o tupla template - per descrivere/definire set di tuple - e il meccanismo di matching - per associare le tuple alle anti-tuple

Comunicazione Generativa

Le tuple generate dai coordinabili hanno un'esistenza indipendente dal generatore e sono egualmente accessibili da tutti i coordinabili.
La caratteristica principale della comunicazione generativa è il *disaccoppiamento*: nello spazio, nel tempo e di nome. I coordinabili, quindi, non hanno bisogno di conoscersi l'un l'altro, di coesistere nello stesso spazio o allo stesso tempo per comunicare (per scambiare tuple, in particolare).

Accesso Associativo alla Conoscenza

Alle tuple nello spazio di tuple si accede attraverso il loro contenuto e struttura, invece del loro nome, indirizzo o allocazione.
L'accesso associativo è basato sul tuple matching e promuove una sincronizzazione basata sulla struttura e contenuto delle tuple: di conseguenza la coordinazione è data-driven.

Semantica Sospensiva

Come già sottolineato precedentemente, le primitive possono avere una semantica sospensiva, cioè possono essere sospese se nessuna tupla nello spazio di tuple fa match con il template e possono essere risvegliate quando nello spazio arriva una tupla che fa match.

2.1.2 I centri di tuple

Negli spazi di tuple Linda la coordinazione è limitata allo scrivere, leggere, consumare una tupla alla volta: il comportamento del medium di coordinazione è fissato una volta per tutte e quindi solo certi problemi di coordinazione vengono risolti soddisfacentemente.

Per superare questo limite non voglio aggiungere primitive perché non risolvono il problema in generale e non si adattano a scenari aperti (dove invece è importante un numero limitato di primitive ben conosciute) e non voglio che i coordinabili si carichino una parte di coordinazione, devono fare solo quello che compete loro, senza preoccuparsi della coordinazione.
La soluzione giusta è mettere capacità computazionale nello spazio di tuple, cioè incapsulare in esso le soluzioni ai problemi sociali. Deve essere possibile aggiustare il comportamento dello spazio in base al problema di coordinazione che si presenta.
Quello che si deve fare, quindi, è mantenere l'interfaccia standard dello spazio di tuple e rendere possibile l'arricchimento del comportamento dello spazio di tuple in termini di transizioni di stato in risposta all'occorrenza di eventi di comunicazione standard: lo spazio di tuple diventa il *centro di tuple* [18].
Un centro di tuple, secondo la definizione formale, è uno spazio di tuple aumentato con una specifica di comportamento, che definisce il comportamento del centro di tuple in risposta a eventi di interazione. La specifica di comportamento è espressa in termini di *reaction specification language* e associa ad ogni evento del centro di tuple un set di attività computazionali (anche vuoto), chiamate reazioni.

2.2 TuCSoN e ReSpecT

TuCSoN (Tuple Centres Spread over the Network) è un modello per la coordinazione di processi distribuiti, come agenti autonomi, intelligenti e mobili ed è quindi utilizzato come modello e infrastruttura per la coordinazione nei MAS.
TuCSoN usa come medium di coordinazione il centro di tuple, cioè uno spazio di tuple aumentato con la possibilità di programmare il suo comportamento in risposta all'interazione.
I centri di tuple TuCSoN sono programmati attraverso il linguaggio di specifica logic-based ReSpecT (Reaction Specification Tuples).

In questa sezione vengono definiti i centri di tuple TuCSoN e il linguaggio di specifica ReSpecT e come essi sono reconcepiti secondo la prospettiva A&A.

2.2.1 Il centro di tuple TuCSoN

Come definito in precedenza, il centro di tuple ha la stessa interfaccia di uno spazio di tuple standard quindi gli agenti possono operare sul centro di tuple TuCSoN nello stesso modo che su uno spazio di tuple Linda: cioè scambiando tuple attraverso un semplice set di primitive. Un agente può scrivere una

tupla in un centro di tuple con la primitiva *out* oppure può leggerla utilizzando primitive come *in*, *rd*, *inp*, *rdp* specificando un tuple template e la sua richiesta sarà soddisfatta seguendo il meccanismo di matching. Leggere tuple può essere distruttivo o non distruttivo, sospensivo o non sospensivo ma comunque sempre non deterministico: esattamente come in un normale spazio di tuple Linda.
I centri di tuple hanno anche le stesse caratteristiche dello spazio di tuple, cioè la comunicazione generativa, l'accesso associativo e la semantica sospensiva.
Una caratteristica specifica di TuCSoN è che, mentre il modello base dei centri di tuple è indipendente dal tipo di tuple [18], TuCSoN adotta tuple logiche - sia le tuple che i tuple templates sono essenzialmente Prolog facts - e come meccanismo di matching usa l'unification.
L'infrastruttura TuCSoN rende possibile l'utilizzo dei centri di tuple come servizi di coordinazione distribuiti attraverso la rete [19]. In particolare, l'intero spazio di coordinazione TuCSoN è costituito da un set aperto di nodi, che corrispondono agli Internet host o server connessi dalla rete. Ogni nodo può contenere più centri di tuple, ognuno identificato da un nome logico univoco (all'interno del nodo). Un agente può agire sugli spazi di tuple utilizzando il nome intero (nome logico + indirizzo del nodo che ospita il centro di tuple) o utilizzando il nome locale, per centri di tuple allocati sullo stesso host dell'agente. Un agente, quindi, può utilizzare sia lo spazio di coordinazione locale sia quello globale.
Infine, TuCSoN è un centro di tuple cioè uno spazio di tuple programmabile e quindi, come già sottolineato in precedenza, mentre il comportamento di uno spazio di tuple in risposta ad un evento di comunicazione è fissato (quindi gli effetti delle primitive di coordinazione sono fissati), il comportamento di un centro di tuple può essere adattato ai bisogni delle applicazioni, definendo un set di tuple di specifica, o reazioni, che determinano come un centro di tuple deve reagire ad eventi entranti/uscenti.
Mentre il modello base del centro di tuple non è legato a nessun specifico linguaggio per definire le reazioni [18], TuCSoN adotta il linguaggio logic-based ReSpecT per programmare i centri di tuple.

2.2.2 Il linguaggio di specifica ReSpecT

Il ReSpecT originale [20] è un linguaggio logic-based per la specifica del comportamento di un centro di tuple adottato da TuCSoN.
Come linguaggio di specifica del comportamento, ReSpecT:

- abilita la definizione di computazioni, chiamate reazioni, all'interno di un centro di tuple

- rende possibile associare reazioni agli eventi che avvengono in un centro di tuple.

ReSpecT, quindi, ha sia una parte dichiarativa che procedurale. Come *linguaggio di specifica*, permette che gli eventi siano dichiarativamente associati alle reazioni mediante specifiche tuple logiche, chiamate tuple di specifica, la cui forma è *reaction (E,R)*. In breve, dato un evento *Ev*, una tupla di specifica *reaction (E,R)* associa una reazione $R\vartheta$ a *Ev* se $\vartheta = mgu\ (E,\ Ev)$.
Come *linguaggio di reazione*, ReSpecT abilita le reazioni che devono essere definite proceduralmente in termini di sequenze di goal di reazione logici, ognuno sia che abbia successo sia che fallisca. Una reazione nel complesso ha successo se tutti i suoi goal di reazione hanno successo e fallisce altrimenti. Ogni reazione è eseguita sequenzialmente con una semantica transazionale: una reazione fallita, quindi, non ha effetto sullo stato di un centro di tuple logico. Tutte le reazioni attivate da un evento di comunicazione sono eseguite prima di servire ogni altro evento: quindi gli agenti percepiscono i risultati dell'evento e l'esecuzione di tutte le reazioni associate come una singola transizione di stato del centro di tuple.
Adottando l'interpretazione dichiarativa delle tuple ReSpecT, un centro di tuple TuCSoN ha una duplice natura [20]: una teoria di comunicazione (il set di tuple ordinarie) e una teoria di coordinazione (il set di tuple di specifica). Questo principalmente permette ad agenti intelligenti di conoscere lo stato delle attività di collaborazione ed, eventualmente, di influire sulle loro dinamiche.
La duplice interpretazione delle tuple ReSpecT (dichiarativa o procedurale), inoltre, permette che la conoscenza e il controllo siano rappresentate uniformemente e incapsulate nello stesso artefatto di coordinazione.

2.2.3 TuCSoN & ReSpecT nella prospettiva A&A

Nella prospettiva A&A, TuCSoN fornisce agli agenti una molteplicità di artefatti distribuiti (i centri di tuple) contenenti sia conoscenza condivisa sia logica di coordinazione espressa in termini di tuple logiche. I centri di tuple ReSpecT sono artefatti ispezionabili, ma comunque non controllabili. Inoltre, sono malleabili, dato che il loro comportamento può essere influenzato a run-time cambiando le specifiche di comportamento.
La specifica originale di ReSpecT non includeva né la linkabilià né la situatedness [18], ma già si stavano compiendo studi per muoversi in quelle direzioni e ben presto furono introdotte due estensioni.
La prima [21] introduceva la prima primitiva di link per la composizione di centri di tuple e cioè *out_ tc*.

Nella seconda [22] fu definito il Timed ReSpecT che proponeva artefatti e centri di tuple a tempo e permetteva la specifica di politiche di coordinazione dipendenti dal tempo, incapsulate dentro al centro di tuple Timed ReSpecT.

2.3 A&A ReSpecT

In questa sezione si osserva come vengono reconcepiti alcuni concetti secondo la prospettiva A&A, le novità portate in ReSpecT proprio da tale prospettiva e la semantica informale e formale di A&A ReSpecT [23].

2.3.1 Adottare la prospettiva A&A

Adottare la prospettiva A&A promuove una visione più articolata dello spazio di interazione MAS. Prima di tutto, è richiesta una nozione di evento più generale [23]. Dato che gli artefatti sono entità passive, le uniche vere sorgenti di eventi in un MAS sono gli agenti e l'ambiente. Qualunque cosa accade in un MAS, quindi, la sua *causa primaria* è un'azione di un agente o un fenomeno ambientale.
Causa primaria e *causa diretta* possono non coincidere: gli artefatti sono collegati gli uni agli altri quindi la causa diretta di un evento può essere un'invocazione di link da parte di un altro artefatto.
Un descrittore generale di eventi, quindi, dovrebbe includere sia la causa originale di un evento, sia la più diretta – ciò permette di osservare la catena degli eventi e definire in modo adatto il comportamento coordinativo dell'artefatto.
Come meta-modello per la computazione distribuita, A&A promuove anche il disaccoppiamento del controllo [23]: gli artefatti linkati devono essere completamente disaccoppiati, gli agenti devono essere lasciati liberi di scegliere autonomamente primitive sincrone o asincrone, mentre il comportamento dell'artefatto target deve rimanere immutato e non influenzato.
Come risultato, ogni operazione (o link) su un artefatto deve avere una struttura di richiesta/risposta: ogni invocazione (richiesta), una volta servita, implica sempre un messaggio di completion (risposta) – con i risultati, se servono – per essere gestito dall' "operatore" in base alla sua natura.
In caso di link invocato da un altro artefatto, il completion deve essere gestito in un modo completamente asincrono – per assicurare il completo disaccoppiamento del controllo; in caso di un'operazione invocata da un agente, il completion deve essere trattato in modo sincrono o asincrono in base alla scelta autonoma dell'agente.
In generale, le operazioni per usare un artefatto da parte di un agente e i link per la composizione tra artefatti non sono necessariamente legati – se non

dalla struttura e dal comportamento dell'artefatto. Comunque, l'integrità concettuale dell'ingegneria degli artefatti e dei MAS trae beneficio dall'uniformità tra operazioni degli artefatti e link. Quindi, è desiderabile che le primitive per le operazioni dell'artefatto siano disponibili all'utilizzo sia da parte degli agenti che degli artefatti – senza assunzioni a priori sulla natura e il comportamento dell'invocatore della primitiva, ma con l'abilità dell'artefatto di capire la natura dell'invocatore osservando l'invocazione quando è utile o richiesto.

2.3.2 A&A ReSpecT: le novità

Il linguaggio originale ReSpecT è stato rivisitato ed esteso per seguire la prospettiva A&A [23].
La prima estensione riguarda la parte di specifica di ReSpecT A&A: la tupla di specifica *reaction* è stata estesa includendo una *guardia*.
Il comportamento di un centro di tuple ReSpecT A&A, quindi, è definito in termini di tuple di specifica nella forma *reaction (E, G, R)*: tale tupla associa una reazione $R\vartheta$ a *Ev* se $\vartheta = mgu\ (E,\ Ev)$ e la guardia G è vera.
Una guardia è una sequenza di predicati di guardia:

$\langle Guard \rangle ::= \langle GuardPredicate \rangle \mid (\langle GuardPredicate \rangle \{, \langle GuardPredicate \rangle\})$

$\langle GuardPredicate \rangle ::=$ request | response | success | failure | endo | exo | intra | inter | from_agent | to_agent | from_tc | to_tc | before ($\langle Time \rangle$) | after ($\langle Time \rangle$)

Un grande numero di condizioni su un evento sono ora controllate prima che una reazione sia attivata in un centro di tuple: lo stato dell'evento, la sua sorgente, il suo target, il suo tempo.
Lungo la stessa linea, i predicati di osservazione sono stati generalizzati in accordo al nuovo modello degli eventi:

$\langle ObservationPredicate \rangle ::= \langle EventView \rangle\_\langle EventInformation \rangle$

$\langle EventView \rangle ::=$current | event | start

$\langle EventInformation \rangle ::=$predicate | tuple | source | target | time

In particolare i predicati event e start si riferiscono alla causa diretta e a quella primaria, rispettivamente.

Un'altra estensione fondamentale concerne l'uniformità delle operazioni, dei link e delle operazioni interne nei centri di tuple. Le primitive ammissibili in un centro di tuple A&A ReSpecT possono essere invocate da un agente ma possono essere anche usate dentro le reazioni di un centro di tuple per

agire sul suo stato o su quello di un altro centro di tuple attraverso un'invocazione di link. Anche la semantica è essenzialmente la stessa - con l'unica eccezione delle primitive in e rd, la cui semantica sospensiva non è preservata all'interno di una reazione e in un'invocazione di link (esse collassano in inp e rdp). Inoltre, la stessa classe di predicati usata per le tuple ordinarie può essere usata per le tuple di specifica al meglio, semplicemente aggiungendo il postfisso _s - così si aggiunge un'altra dimensione all'uniformità.

$\langle Reaction \rangle ::= \langle ReactionGoal \rangle \mid (\langle ReactionGoal \rangle \{, \langle ReactionGoal \rangle\})$

$\langle ReactionGoal \rangle ::= \langle TCPredicate \rangle (\langle Tuple \rangle) \mid \langle ObservationPredicate \rangle (\langle Tuple \rangle) \mid \langle Computation \rangle \mid (\langle ReactionGoal \rangle ; \langle ReactionGoal \rangle)$

$\langle TCPredicate \rangle ::= \langle SimpleTCPredicate \rangle \mid \langle TCLinkPredicate \rangle$

$\langle TCLinkPredicate \rangle ::= \langle TCIdentifier \rangle \ ? \ \langle SimpleTCPredicate \rangle$

$\langle SimpleTCPredicate \rangle ::= \langle TCStatePredicate \rangle \mid \langle TCForgePredicate \rangle$

$\langle TCForgePredicate \rangle ::= \langle TCStatePredicate \rangle$_s

$\langle TCStatePredicate \rangle ::=$in | inp | rd | rdp | out | no | get | set

Infine, A&A ReSpecT include la prima estensione verso la situatedness degli artefatti di coordinazione: Timed ReSpecT. Esso include eventi di tempo, reazioni a tempo e predicati per gestire il tempo.
Gestire il tempo è una delle prime, essenziali caratteristiche per ogni modello di coordinazione real-world.

2.3.3 A&A ReSpecT: semantica informale

Quando si ha un'invocazione di una primitiva di un centro di tuple da parte di un agente (operazione) o da parte di un artefatto (link), un evento ammissibile A&A ReSpecT è generato e raggiunge il suo centro di tuple target, dov'è automaticamente inserito nella coda *InQ*. Quando il centro di tuple è in idle (cioè nessuna reazione è correntemente in esecuzione), il primo evento ϵ in *InQ* (in accordo con la politica FIFO) è loggato e mosso al multiset *Op* delle richieste che devono essere servite: questa fase è chiamata fase di richiesta di un evento ϵ.
Dopodichè le reazioni della fase di richiesta di ϵ sono attivate, aggiungendole al multiset *Re* delle reazioni attivate, in attesa di essere eseguite. Tutte le reazioni attivate in *Re* sono allora eseguite in un ordine non deterministico. Ogni reazione è eseguita sequenzialmente, con una semantica transazionale e può attivare ulteriori reazioni, che ancora devono essere aggiunte a *Re*, così come i nuovi eventi di output che rappresentano delle invocazioni di link:

tali eventi sono aggiunti al multiset *Out* degli eventi in uscita e poi sono spostati alla coda di uscita del centro di tuple *OutQ* alla fine dell'esecuzione della reazione - se ha successo. Solo quando *Re* è infine vuota, le richieste in attesa di essere servite in *Op* sono eseguite dal centro di tuple e il completion dell'operazione/link e mandato indietro agli invocatori. Ciò può portare ulteriori reazioni, associate alla fase di risposta dell'invocazione originale e sono eseguite ancora con la stessa semantica specificata per la fase di richiesta. E così, il ciclo di un centro di tuple A&A ReSpecT [23] è infine concluso.

2.3.4 A&A ReSpecT: semantica formale

Evento ReSpecT A&A
Un evento ammissibile di un centro di tuple per ReSpecT A&A è così definito:

$\langle GeneralTCEvent \rangle ::= \langle StartCause \rangle , \langle Cause \rangle , \langle TCCycleResult \rangle$

In cui:

$\langle StartCause \rangle , \langle Cause \rangle ::= \langle SimpleTCEvent \rangle , \langle Source \rangle , \langle Target \rangle , \langle Time \rangle$

Tale definzione implicitamente definisce anche come un evento ReSpecT è denotato: se ϵ è un evento A&A ReSpecT, $\epsilon.Cause.Source$ denota l'entità che ha causato direttamente l'evento, $\epsilon.TCCycleResult$ denota i risultati della computazione del centro di tuple attivata dall'evento e così via.
Multiset di A&A ReSpecT Triggered Reaction
Dato un centro di tuple c e la sua specifica di comportamento Σ, se ϵ è un evento A&A ReSpecT, allora il multiset di triggered reaction di ϵ è definito come:

$$Z_\Sigma(\epsilon) ::= \biguplus_{reaction(e,G,R)\epsilon\Sigma} (\epsilon, R\vartheta) \mid \vartheta = Unify(\varepsilon, e) \neq \perp, Guard(\epsilon, G)$$

Dove:

$$Unify(\epsilon,e) ::= mgu(e,\epsilon.Cause.SimpleTCEvent)$$

Multiset di A&A ReSpecT Time-Triggered Reaction
Dato un centro di tuple c e la sua specifica di comportamento Σ, se nc è il tempo locale del centro di tuple, allora il multiset di time-triggered reaction di nc è definito come:

$$Z_\Sigma(\mathrm{nc}) ::= \biguplus_{reaction(time(t)e,G,R)\epsilon timed(nc,\Sigma)} (\epsilon, R) \mid Guard(\epsilon, G)$$

Quindi, dato un centro di tuple c al tempo nc con una specifica di comportamento Σ e un evento ϵ, $Z_\Sigma(\mathfrak{e})$ denota il multiset di triggered reaction causate da ϵ, mentre $Z_\Sigma(nc)$indica il multiset di time-triggered reaction al tempo nc.

Funzione di esecuzione delle reazioni

Siano R e R' sequenze di goal delle reazioni, Tu e Tu' multiset di tuple logiche (ordinarie), Σ e Σ' multiset di tuple di specifica, Re e Re' multiset di triggered reaction , ϵ un evento A&A ReSpecT e Out e Out' sequenze di eventi A&A ReSpecT. Uno stato di esecuzione delle reazioni è quindi definito come una quintupla $\langle R, Tu, \Sigma, Re, Out\rangle_\varepsilon$, dove uno step di esecuzione delle reazioni è una transizione

$$\langle R, Tu, \Sigma, Re, Out\rangle_\epsilon \rightarrow_e \langle R', Tu', \Sigma', Re', Out'\rangle_\epsilon$$

definita seguendo determinate regole.

Lo stato finale della sequenza di esecuzione delle reazioni è $\langle R, Tu, Re, Out\rangle_\epsilon^*$ ed è il primo stato della sequenza per cui non esistono regole da applicare.

Se $\langle R, Tu, \Sigma, Re, Out\rangle_\epsilon^* = \langle R', Tu', \Sigma', Re', Out'\rangle_\epsilon$, allora la funzione di esecuzione delle reazioni E è definita come:

$$\mathsf{E}((\epsilon, R), Tu, \Sigma) ::= \begin{cases} (Tu', \Sigma', Re', Out') & se \quad R' = 0 \\ (Tu, \Sigma, 0, 0) & se \quad R' \neq 0 \end{cases}$$

Ad ogni step di una sequenza di esecuzione di reazioni, R rappresenta i goal di reazione che devono ancora essere eseguiti, Tu lo stato corrente dello spazio delle tuple ordinarie, Σ lo stato corrente dello spazio delle tuple di specifica, Re il set di triggered reaction di reazioni già eseguite, Out la sequenza di eventi che devono essere emessi alla fine dell'esecuzione, mentre ϵ è l'evento che inizialmente ha attivato l'esecuzione delle reazioni.

In modo corrispondente, l'esecuzione di triggered reaction *(ϵ,R)* in un centro di tuple A&A ReSpecT, il cui spazio di tuple è Tu e la cui specifica di comportamento è Σ, è rappresentata da una sequenza il cui stato iniziale è $\langle R, Tu, \Sigma, 0, 0\rangle_\epsilon$.

La definizione di E permette anche di definire una semantica transizionale di successo/fallimento delle reazioni A&A ReSpecT. Se la sequenza di goal delle reazioni R' è vuota, allora la reazione R attivata dall'evento ϵ ha avuto successo e un nuovo multiset Tu' di tuple ordinarie, un nuovo multiset Σ' di tuple di specifica, un nuovo set di triggered reaction Re' e la sequenza di eventi che devono essere emessi Out' sono forniti per aggiornare lo stato del centro di tuple. Nell'altro caso ($R' \neq 0$ e nessun'altro step di esecuzione di reazione è possibile), i vecchi multiset Tu e Σ sono restituiti, nessuna nuova reazione è attivata e nessun evento viene emesso - quindi non ci sono ulteriori cambiamenti nello stato del centro di tuple.

Lo stato finale di una sequenza è raggiunto quando o non ci sono più reazioni che devono essere eseguite o quando non sono disponibili altre regole applicabili. Dato che ogni reazione cancella realmente un goal da una reazione e il numero di goal di reazione è finito per ogni reazione, è garantito che ogni reazione è eseguita in un numero finito di step.

2.3.5 Comportamento dei centri di tuple A&A ReSpecT

Lo stato di un centro di tuple A&A ReSpecT [23] è espresso da una quadrupla ${}^{InQ}\langle Tu, \Sigma, Re, Op\rangle_n^{OutQ}$. Qui, Tu e Σ sono i multiset di tuple ordinarie e di specifica nel centro di tuple rispettivamente, Re è il multiset di triggered reaction in attesa di essere eseguite, Op è il multiset di richieste in attesa di una risposta, InQ e $OutQ$ sono le code di eventi entranti ed uscenti rispettivamente ed infine n è il tempo del centro di tuple. InQ è una coda che viene automaticamente estesa quando si hanno eventi entranti centro di tuple - quindi non sono richieste speciali transizioni per gli eventi entranti. In modo duale, $OutQ$ è automaticamente svuotata emettendo gli eventi uscenti, ancora una volta senza bisogno di transizioni speciali.
Il comportamento operazionale di un centro di tuple A&A ReSpecT [23], il cui stato è ${}^{InQ}\langle Tu, \Sigma, Re, Op\rangle_n^{OutQ}$, può essere modellato in termini di sistema transizionale con quattro diversi tipi di transizioni, in ordine di priorità decrescente:

Reazione
Quando $Re \neq 0$ le triggered reaction in Re sono eseguite attraverso una transizione di reazione ($\rightarrow_r$)
${}^{InQ}\langle Tu, \Sigma, Re \cup re, Op\rangle_n^{OutQ} \rightarrow_r {}^{InQ}\langle Tu', \Sigma', Re \cup Re', Op\rangle_{n'}^{OutQ,Out'}$

Tempo
Quando $Re \neq 0$ e $\mathsf{timed}(n,\Sigma) \neq 0$, le timed reaction possono attivare nuove reazioni attraverso una transizione di tempo ($\rightarrow_t$)
${}^{InQ}\langle Tu, \Sigma, 0, Op\rangle_n^{OutQ} \rightarrow_t {}^{InQ}\langle Tu, \Sigma/timed(n,\Sigma), Z_\Sigma(n), Op\rangle_{n'}^{OutQ}$

Servizio
Quando $Re = \mathsf{timed}(n,\Sigma) = 0$ e $\mathsf{sat}(Op, Tu, \Sigma) \neq 0$ le richieste in attesa di risposta possono essere servite attraverso una transizione di servizio ($\rightarrow_s$)
${}^{InQ}\langle Tu, \Sigma, 0, Op \cup \varepsilon\rangle_n^{OutQ} \rightarrow_s {}^{InQ}\langle Tu', \Sigma', Z_\Sigma(\varepsilon') \cup Z_\Sigma(n), Op\rangle_{n'}^{OutQ,\varepsilon'}$

Log
Quando $Re = \mathsf{timed}(n,\Sigma) = \mathsf{sat}(Op, Tu, \Sigma) = 0$ e $InQ \neq 0$ il centro di tuple accede alle richieste in coda attraverso una transizione di log ($\rightarrow_l$)
${}^{\varepsilon,InQ}\langle Tu, \Sigma, 0, Op\rangle_n^{OutQ} \rightarrow_l {}^{InQ}\langle Tu, \Sigma, Z_\Sigma(\varepsilon) \cup Z_\Sigma(n), Op'\rangle_{n'}^{OutQ}$

Capitolo 3

Esperimenti in ReSpecT

In questo capitolo vengono indicate le scelte architetturali per rappresentare un artefatto in TuCSoN, in particolare osservando come sono rappresentate le proprietà più importanti di un artefatto cioè function description, operating instructions e usage interface. Si definiscono inoltre le reazioni ReSpecT del centro di tuple in risposta ad alcune possibili richieste degli agenti. Infine, viene mostrato un esempio di un semplice artefatto.

3.1 Scelte architetturali

In questa sezione si discutono le possibili scelte architetturali per rappresentare un artefatto in TuCSoN e viene indicata la soluzione adottata.

La nozione di uso è fondamentale per un artefatto e per la relazione tra gli agenti e gli artefatti. È perciò importante rappresentare gli artefatti in modo adeguato e semplice per essere usati efficacemente dagli agenti.
Nel capitolo precedente è stato analizzato come TuCSoN e ReSpecT sono stati reconcepiti ed estesi per adattarsi alla prospettiva A&A quindi pare naturale pensare ad essi quando si decide di rappresentare un artefatto.
Adottando il meta-modello A&A, secondo il quale un MAS è rappresentato da agenti ed artefatti, organizzati in workspaces, gli artefatti verranno rappresentati ciascuno come dei centri di tuple ReSpecT, facenti parte di un workspace rappresentato da un nodo TuCSoN.
Quando si ragiona su di un artefatto, vengono subito in mente le sue proprietà più importanti: la function description, le operating instructions e l'usage interface.
In base a queste proprietà si è pensato a due possibili soluzioni per rappresentare gli artefatti.

Come prima soluzione si può pensare di avere tanti centri di tuple nel nodo TuCSoN (quanti sono gli artefatti) e rappresentare le proprietà come tuple.
Un'altra soluzione interessante è dare a queste proprietà una maggiore rilevanza, concependole come dei centri di tuple. L'idea quindi è di una rete a stella: un centro di tuple sociale che rappresenta l'artefatto e collegati ad esso tre centri di tuple individuali - il centro di tuple function description, il centro di tuple operating instructions e il centro di tuple usage interface. Siccome ciascuna proprietà dell'artefatto non è un singolo elemento ma un insieme di elementi, ogni tupla all'interno di un centro di tuple individuale sarà un elemento della proprietà.
Adottando questa soluzione gli agenti rivolgono le loro richieste al centro di tuple sociale: esso poi si linkerà al centro di tuple adeguato e soddisferà la richiesta dell'agente. È però evidente che questa soluzione non è la più semplice ed appesantisce notevolmente la definizione della specifica di comportamento di ogni centro di tuple, che si troverà a dover gestire molti link.
Viene, quindi, scelta la prima soluzione. Nel centro di tuple che rappresenta un artefatto saranno inserite le tuple e ciascuna sarà un elemento della function description, delle operating instructions o dell'usage interface ed infatti, per essere distinte, avranno un identificativo oltre al loro valore (ad esempio `fd`, `oi` o `ui`)

3.2 Rappresentazione delle proprietà

In questa sezione si osserva come sono rappresentate le proprietà fondamentali degli artefatti. La soluzione scelta per rappresentarle è legata alla scelta precedente di rappresentare un artefatto come un centro di tuple.

3.2.1 Function description

La function description è la descrizione della funzione fornita dall'artefatto, il motivo per cui l'artefatto è stato costruito. Essenzialmente essa contiene informazioni riguardanti i possibili usi che il progettista aveva immaginato per l'artefatto.
Può essere, quindi, concepita come una lista di goal che un agente può realizzare utilizzando l'artefatto oppure come una lista di tasks che quell'artefatto può compiere.
Viene deciso di utilizzare la prima opzione perchè, pensando agli usi da parte dell'agente analizzati successivamente, agevolerà il lavoro: le tuple che rap-

presenteranno la function description, quindi, saranno i goal che si possono raggiungere utilizzando l'artefatto.
Ogni tupla verrà inserita nel centro di tuple con la primitiva `out` e la sua forma sarà `goal(goalX)`.

3.2.2 Operating instructions

Le operating instructions sono come un manuale di istruzioni di un device fisico e descrivono passo-passo il comportamento che deve assumere l'agente per usare correttamente l'artefatto.
Un esempio concreto può servire a capire meglio le operating instructions: si consideri un agente che vuole prendere un caffè utilizzando l'artefatto macchina del caffè.
L'agente, allora, chiederà all'artefatto le operating instructions e esso restituirà la lista di operazioni elementari da compiere per realizzare la "macro" operazione "Prendere il caffè", ad esempio: "Inserire 30 centesimi", "Scegliere la quantità di zucchero", "Spingere il pulsante caffè", "Attendere il bip", "Prendere il bicchierino".
Ciascuna operating instruction, quindi, sarà una lista di operazioni elementari da compiere per realizzare l'operazione voluta.
Il modo più semplice per rappresentarle è una lista con la sequenza di operazioni elementari da eseguire. Un altro esempio interessante è la macchina di Turing, in cui ad ogni stato corrisponde un'operazione elementare e il discriminante per passare da uno stato all'altro è che l'operazione elementare precedente sia stata eseguita.
Anche in questo caso viene scelta la soluzione ritenuta più semplice, cioè la prima: ciascuna tupla sarà relativa ad una singola operazione e sarà rappresentata da una lista.
Ogni tupla sarà inserita nel centro di tuple con la primitiva `out` ed avrà la forma `oi(opA([opX, opY, opZ])`.
Ovviamente all'agente vengono date le istruzioni per eseguire una certa operazione ma lui potrebbe decidere di non seguirle: non vengono posti ulteriori vincoli su questo. L'agente sarà cosciente che, se non seguirà le istruzioni e si comporterà in maniera differente, non riuscirà ad eseguire l'operazione.

3.2.3 Usage interface

L'usage interface è costituita dalle operazioni elementari che può svolgere un artefatto. Ogni operazione elementare è caratterizzata dal suo nome e da un suo eventuale argomento.

Le varie tuple che la rappresenteranno, quindi, saranno ciascuna un'operazione elementare dell'artefatto (una di quelle che poi comparirà in una sequenza di un'operating instruction).
Ogni tupla verrà inserita nel centro di tuple con la primitiva `out` e la sua forma sarà `ui(opX,arg)`.

3.3 Specifica di comportamento

Una volta definito un artefatto come centro di tuple, è necessario definire delle reazioni del centro di tuple alle possibili richieste da parte degli agenti.

Prima di analizzare le reazioni, bisogna fare un'osservazione importante e cioè come l'agente porrà le sue richieste al centro di tuple: normalmente l'agente farebbe `out(richiesta)`, qui invece viene obbligato a porre la sua richiesta con una `in`. In questo modo l'agente è bloccato e percepisce la sua richiesta e la risposta del centro di tuple come un'unica operazione atomica. Se usasse la `out`, sarebbe subito sbloccato e potrebbe anche decidere di fare qualcos'altro, andarsene e non guardare i risultati e in questo modo il centro di tuple avrebbe sfruttato inutilmente tempo e forza computazionale.

Come prima cosa, gli agenti potrebbero voler vedere una delle proprietà del centro di tuple.
Le richieste che devono effettuare, quindi, sono: `get_fd`, `get_oi` o `get_ui`.
Adesso appare chiara l'importanza dell'identificativo di ogni tupla (`goal`, `oi` e `ui`). Infatti, quando l'agente chiederà una delle proprietà, l'artefatto dovrà restituirgli la lista di tutti gli elementi che la compongono e per fare ciò deve utilizzare una `rd_all` passando un template che faccia in modo che tutti gli elementi della proprietà richiesta facciano match e siano quindi restituiti all'agente.

```
reaction( in(get_fd(L)), (operation,invocation), (
rd_all(goal(X),LL),out(get_fd(LL))).

reaction( in(get_oi(L)), (operation,invocation), (
rd_all(oi(X),LL),out(get_oi(LL))).

reaction( in(get_ui(L)), (operation,invocation), (
rd_all(ui(X,ARG),LL),out(get_ui(LL))).
```

In queste reazioni gli eventi sono `in(get_fd(L))`, `in(get_oi(L))` e `in (get_ui(L))`. All'invocazione di questi eventi, il centro di tuple deve raccogliere tutte le tuple che rappresentano la proprietà richiesta quindi si attiva

una reazione che utilizza una primitiva `rd_all` con un template che permetta a tutte le tuple che rappresentano una certa proprietà di fare match. Per come sono state definite precedentemente le tuple, per `in(get_fd)` il template sarà `goal(X)`, per `in(get_oi)` il template è `oi(X)` e per `in(get_ui)` il template è `ui(x)`. L'altra reazione che si attiva è la `out` della richiesta iniziale fatta con `in`, quindi, ad esempio, nel primo caso `out(get_fd(LL))`: in questo modo la primitiva iniziale si sblocca (era sospesa a causa della proprietà di `in` di avere semantica sospensiva), estrae questa tupla inserita con la `out` e così si ritrova in mano anche la lista contenente tutte le tuple che rappresentano la proprietà richiesta. La lista è quella che restituisce la `rd_all`: essa termina sempre con successo infatti, se nessuna tupla fa match, la lista sarà vuota ma verrà comunque restituita. Questo è il motivo per cui l'evento iniziale ha come argomento una lista: in questo modo alla fine farà match con la tupla finale, che sblocca l'agente e gli restituisce la lista. Dato che la `rd_all` non fallisce mai non devo avere altre reazioni nell'eventuale caso di fallimento.

Per l'agente potrebbe essere di maggior interesse non conoscere tutti i possibili goal dell'artefatto ottenendo tutta la function description, ma solo se il suo goal fa match con uno dei goal dell'artefatto e quindi se quell'artefatto potrà servirgli per i suoi scopi futuri.
L'agente, quindi, inserirà una richiesta passando come argomento il suo goal: `in(see_goalmatch(goal(MYGOAL)),X)`.
Con questa richiesta l'agente si aspetta che un argomento della tupla di risposta gli dica se il suo goal fa match con uno di quelli dell'artefatto e quindi se è utile utilizzare quell'artefatto o meno. Oltre a passare il suo goal, quindi, l'agente passerà anche la variabile che conterrà la stringa che gli dirà se il match è avvenuto o meno.

```
reaction( in(see_goal_match(goal(MYGOAL)),X), (operation,
invocation), ( rd(goal(MYGOAL)), out(
see_goal_match(goal(MYGOAL)), match )).

reaction( in(see_goal_match(goal(MYGOAL)),X), (operation,
invocation), ( no(goal(MYGOAL)), out(
see_goal_match(goal(MYGOAL)), no_match )).
```

In queste reazioni l'evento è `in(see_goal_match(goal(MYGOAL)),X)`. Il centro di tuple deve guardare se `goal(MYGOAL)` passato dall'agente fa match con uno dei suoi goal quindi deve prevedere 2 reazioni: una nel caso di match e una nel caso contrario. La prima reazione, all'invocazione dell'evento attiva

una `rd` passando come template `goal(MYGOAL)` e la `out` della richiesta iniziale così l'agente si sblocca ed estrae questa tupla inserita con la `out` che ha come argomento il template che avevo passato all'inizio e la stringa `match` che gli interessava. Se nessuna tupla fa match la `rd` fallisce (perchè all'interno di una reazione collassa in `rdp` quindi non ha più semantica sospensiva) e di conseguenza tutta la reazione fallisce. Il centro di tuple allora passa alla reazione successiva che attiva una `no` (che ha successo se nessuna tupla fa match con il template) e una `out` della richiesta iniziale con argomento il template che avevo passato all'inizio e una stringa `no_match`. In questo caso si gestiscono tutti i casi possibili perchè, se la prima reazione non ha successo, sicuramente avrà successo la seconda - e viceversa.

Una volta che l'agente ha analizzato la function description di un artefatto e decide di utilizzarlo perchè esso può realizzare il suo goal, può voler vedere tutte le sue operazioni (con relative istruzioni) per decidere quale operazione utilizzare per realizzare il suo goal. In altri casi però l'agente sa già quale operazione chiamare per realizzare il suo goal quindi può essere interessato a ricevere le operating instructions della sola operazione che gli ineteressa. Si pensi all'esempio precedente: la macchina del caffè è un artefatto noto quindi se il goal di un agente è "Superare il sonno", sa che l'operazione che dovrà compiere è "Prendere un caffè".
La richiesta dell'agente per richiedere le operating instructions di una sola operazione perciò sarà `get_oi(oi(OPA),X)`. Con questa richiesta l'agente si aspetta di ricevere una lista delle operazioni elementari necessarie per realizzare quell'operazione.

```
reaction( in(get_oi(oi(OPA),X)), (operation, invocation), (
rd(oi(OPA)),out(get_oi(oi(OPA),op_admissible)) )).

reaction( in(get_oi(oi(OPA),X)), (operation, invocation), (
no(oi(OPA)),out(get_oi(oi(OPA),op_not_admissible)) )).
```

L'evento `in(get_oi(oi(OPA),X))` fa attivare una reazione che prima di tutto controlla se quell'operazione è tra quelle ammissibili perchè l'agente potrebbe essersi sbagliato. Il funzionamento è del tutto simile al caso precedente di `in(see_goal_match(goal(MYGOAL),X))`. Il centro di tuple quindi deve prevedere 2 reazioni: una nel caso di match e una nel caso contrario. La prima reazione, all'invocazione dell'evento attiva una `rd` passando come template `oi(OPA)` e la `out` della richiesta iniziale così l'agente si sblocca ed estrae la tupla `get_oi(oi(OPA),op_admissible)` che ha come argomento la lista di operating instructions relativa all'operazione richiesta e una stringa `op_admissible`.

Se nessuna tupla fa match la `rd` fallisce (perchè all'interno di una reazione collassa in `rdp` quindi non ha più semantica sospensiva) e di conseguenza tutta la reazione fallisce. Il centro di tuple allora passa alla reazione successiva che attiva una `no` (che ha successo se nessuna tupla fa match con il template) e una `out` della richiesta iniziale con argomento il template che avevo passato all'inizio e una stringa `op_not_admissible`. Anche in questo caso si gestiscono tutti i casi possibili perchè, se la prima reazione non ha successo, sicuramente avrà successo la seconda - e viceversa.

Una volta che l'agente ha ottenuto le operating instructions, chiama ogni singola operazione elementare dell'usage interface con una richiesta del tipo `in(exec(ui(OPX,ARG),X))`.

```
reaction( in(exec(ui(OPX,ARG),X)), (invocation, from_agent), (
rd(ui(OPX,ARG)),out(exec(ui(OPX,ARG),insert_next_elem_op )) )).

reaction( in(exec(ui(OPX,ARG),X)), (invocation, from_agent), (
no(ui(OPX,ARG)),out(exec(ui(OPX,ARG),elem_op_not_admissible)) )).
```

L'evento `in(exec(ui(OPX,ARG),X))` all'invocazione, fa attivare una reazione che controlla se l'operazione elementare fa match con una delle tuple dell'usage interface perchè l'agente è del tutto autonomo e potrebbe chiamare un'operazione inesistente o non ammissibile (nonostante abbia ricevuto le operating instructions). Il centro di tuple quindi deve prevedere 2 reazioni: una nel caso di match e una nel caso contrario. La prima reazione attiva una `rd` passando come template `ui(OPX,ARG)` e la `out` della richiesta iniziale così l'agente si sblocca ed estrae la tupla inserita con la `out` che ha come argomento il template che avevo passato e la stringa `insert_next_elem_op_`.
Se nessuna tupla fa match la `rd` fallisce (perchè all'interno di una reazione collassa in `rdp` quindi non ha più semantica sospensiva) e di conseguenza tutta la reazione fallisce. Il centro di tuple allora passa alla reazione successiva che attiva una `no` (che ha successo se nessuna tupla fa match con il template) e una `out` della richiesta iniziale che ha come argomento il template che avevo passato all'inizio e una stringa `elem_op_not_admissible`.
Anche in questo caso si gestiscono tutti i casi possibili perchè, se la prima reazione non ha successo, sicuramente avrà successo la seconda - e viceversa.

La guardia `(invocation, from agent)` è inserita pensando a possibili lavori futuri: ad esempio, se si vorranno progettare delle reazioni in risposta alla richiesta `in(exec(oi(OPA,ARG1,ARG2,...),RES))`. In questo caso sarà il centro di tuple che farà ciascuna `in(exec(ui(OPX,ARG),X))` e quindi non ci sarà bisogno del controllo perchè esso non è autonomo come l'agente

e chiamerà sicuramente le operazione elementari indicate nelle operating instructions.
Con questa guardia viene detto di controllare l'ammissibilità dell'operazione elementare solo se la richiesta è effettuata da un agente.

C'è un'ultima reazione importante per il corretto funzionamento dell'esecuzione:

```
reaction( in(exec(ui(last_elem_op,ARG),X)), (operation,
invocation),(out(exec(ui(last_elem_op,ARG),res)) )).
```

Il centro di tuple, quando riceve questa richiesta, capisce che sono state chiamate correttamente tutte le operazioni elementari precedenti e adesso deve restituire il risultato all'agente.
Anche questa operazione elementare `last_elem_op` ha l'argomento ma è solo per mantenerà l'uniformità con le altre tuple dell'usage interface. Quando viene invocata con la `in(exec(...))`, come argomento l'agente passa null.

Le reazioni ReSpecT, quindi, si preoccupano solo di controllare l'ammissibilità delle operazioni elementari richieste e di restituire il risultato: la computazione per ottenere il risultato sarà un'operazione interna del centro di tuple e chi programma le reazioni ReSpecT non se ne preoccupa.

3.4 Esempio

In questa sezione viene dato un esempio di un semplice artefatto. Questo esempio è sviluppato in Java esteso con la libreria TuCSoN quindi si può creare il centro di tuple TuCSoN che sarà l'artefatto, si possono inserire le tuple con le primitive note e le reazioni ReSpecT.
L'esempio implementato è una calcolatrice trivial, che esegue le sole 4 operazioni elementari.

Inizialmente viene creato l'artefatto come centro di tuple:

```
TucsonTupleCentreId ttcid =
   new TucsonTupleCentreId("calcolatrice", ip, port);
```

Dopodichè vengono inserite le tuple che rappresentano i goal che si possono realizzare con l'artefatto:

```
LogicTuple fd1 = LogicTuple.parse("goal(" +
"risolvere_operazioni_semplici)");

acc.out(ttcid, fd1, null);
```

```
LogicTuple fd2 = LogicTuple.parse("goal(" +
"eseguire_una_delle_quattro_operazioni_fondamentali)");
acc.out(ttcid, fd2, null);
```

Successivamente vengono inserite le tuple che rappresentano le operating instructions:

```
LogicTuple op1 = LogicTuple.parse("oi(" +
"somma([inserire_primo_num,inserire_operatore_piu," +
"inserire_secondo_num]))");
acc.out(ttcid, op1, null);

LogicTuple op2 = LogicTuple.parse("oi(" +
"sottrazione([inserire_primo_num," +
"inserire_operatore_meno,inserire_secondo_num]))");

acc.out(ttcid, op2, null);

LogicTuple op3 = LogicTuple.parse("oi(" +
"moltiplicazione([inserire_primo_num," +
"inserire_operatore_per,inserire_secondo_num]))");
acc.out(ttcid, op3, null);

LogicTuple op4 = LogicTuple.parse("oi(" +
"divisione([inserire_primo_num," +
"inserire_operatore_diviso,inserire_secondo_num]))");
acc.out(ttcid, op4, null);
```

Infine vengono inserite le tuple che rappresenteranno l'usage interface:

```
LogicTuple ui1 = LogicTuple.parse("ui(" +
"inserire_primo_num,arg)");
acc.out(ttcid, ui1, null);

LogicTuple ui2 = LogicTuple.parse("ui(" +
"inserire_secondo_num,arg)");
acc.out(ttcid, ui2, null);

LogicTuple ui3 = LogicTuple.parse("ui(" +
"inserire_operatore_piu,arg)");
acc.out(ttcid, ui3, null);
```

```
LogicTuple ui4 = LogicTuple.parse("ui(" +
"inserire_operatore_meno,arg)");
acc.out(ttcid, ui4, null);

LogicTuple ui5 = LogicTuple.parse("ui(" +
"inserire_operatore_per,argop)");
acc.out(ttcid, ui5, null);

LogicTuple ui6 = LogicTuple.parse("ui(" +
"inserire_operatore_diviso,argop)");
acc.out(ttcid, ui6, null);

LogicTuple ui7 = LogicTuple.parse("ui(last_elem_op,ARG)");
acc.out(ttcid, ui7, null);
```

Dopo aver inserito le tuple nell'artefatto vengono inserite le reazioni ReSpecT definite nella sezione precedente.

Conclusione

Lo scopo principale di questa tesi era di costruire un'architettura generale di artefatto secondo il meta-modello A&A basandosi su un'astrazione generale di coordinazione come il centro di tuple ReSpecT.

Prima di tutto era fondamentale, quindi, evidenziare l'importanza degli artefatti nell'ambito dei sistemi multi-agente: si è cercato di raggiungere questo obiettivo nel primo capitolo, in cui si è analizzato il concetto di artefatto e il suo utilizzo nelle varie discipline scientifiche, per trarne insegnamenti interessanti da portare nell'ambito MAS.
Dall'analisi degli artefatti nelle varie discipline, è emerso che l'intelligenza non è solo la capacità di saper utilizzare un linguaggio simbolico ma anche la capacità di saper utilizzare i tools, cioè proprio gli artefatti. Questo vale anche per gli agenti intelligenti e quindi si crea una relazione tra agenti ed artefatti che porta alla creazione del meta-modello A&A: il concetto di agente, di artefatto e di MAS vengono rivisitati secondo questa prospettiva.

I centri di tuple sono dei medium di coordinazione in cui gli agenti vivono ed interagiscono, quindi avendo ridefinito il concetto di agente secondo questa nuova prospettiva, anche i centri di tuple e i linguaggi di specifica sono stati estesi per adattarsi al modello A&A: è stata ridefinita la nozione di evento, si è inserito il concetto di guardia e le reazioni a tempo e si è cercato di uniformare le operazioni, l'invocazioni di link e le operazioni interne al centro di tuple, utilizzando la stessa forma e circa la stessa sintassi. Nel secondo capitolo quindi si è cercato di dare un'idea generale dalle novità portate da questa prospettiva in particolare al centro di tuple TuCSoN e al linguaggio di specifica ReSpecT (che vengono poi usati per realizzare gli artefatti).

Nel terzo capitolo si è cercato di raggiungere l'obiettivo principale della tesi: il MAS è popolato da agenti ed artefatti rappresentati come centri di tuple; le tuple all'interno dei vari artefatti costituiscono le proprietà principali degli artefatti, cioè la function description, le operating instructions e l'usage interface. Dopodichè si è cercato di implementare il linguaggio di specifica di tali

centri di tuple: sono state implementate alcune reazioni ReSpecT riguardanti le richieste principali che potevano essere rivolte ad un artefatto. Inoltre è stato dato un esempio di un artefatto molto semplice, che fornisce una linea guida per poter implementare qualunque artefatto più complesso.

Lo scopo della tesi quindi è stato raggiunto anche se, ovviamente, il lavoro potrà essere esteso in futuro aggiungendo reazioni ad eventuali ulteriori richieste da parte degli agenti (ad esempio l'agente potrebbe voler chiamare un'operazione intera e lasciare tutto a carico del centro di tuple mentre nella tesi è stato gestito solo il caso in cui l'agente guarda le operating instructions e chiama da solo ogni singola operazione elementare) o per gestire situazioni sgradevoli dovute alla totale autonomia degli agenti.
Ad esempio, all'agente vengono fornite le operating instructions ma lui può scegliere di non seguirle e gli viene solo detto se l'operazione richiesta non è ammissibile ma non è gestito altro oppure quando l'artefatto viene a conoscenza del fatto che l'artefatto può realizzare il suo goal (chiedendo l'intera function description o anche solo guardando se il suo goal fa match con uno dei goal dell'artefatto) ma può comunque decidere di non sceglierlo.
Un altro aspetto non gestito è quando l'agente chiama ogni singola operazione elementare con l'operazione di exec e viene solo controllata l'ammissibilità delle operazioni richieste ma non che esse siano nella giusta sequenza: questa è una questione importante che, insieme alle altre già citate, sarà sicuramente oggetto del lavoro futuro.

Bibliografia

[1] S. A. Kauffman (2001) *Investigations*. Oxford University Press.

[2] J. Bardram (1998) *Designing for the dynamics of cooperative work activities*. 1998 ACM conference on computer supported cooperative work (pp.89-98).

[3] Y. Engestrom, K. Brown, L. C. Christopher, J. Gregory (1997) *Coordination, cooperation and communication in the courts: expansive transitions in legal work.*. Mind, culture and activity. Cambridge University Press, chapt. 28.

[4] A. Ricci, A. Omicini, E. Denti (2003) *Activity theory as a framework for MAS coordination.*. Engineering societies in the agent world III, Vol. 2577 of LNCS (pp.96-110).

[5] A. Ricci, M. Viroli, A. Omicini (2006) *Programming MAS with artifacts*. Programming multi-agent systems, Vol. 3862 of LNAI (pp. 206-221).

[6] D. Kirsh (1999) *Distributed cognition, coordination and environment design*. European Conference on Cognitive Science (pp.1-11).

[7] R. Conte, C.Castelfranchi (1995) *Cognitive and social action*. University College London.

[8] K. Schmidt, C.Simone (2000) *Mind the gap! Towards a unified view of CSCW*. Designing cooperative systems: the use of theories and models, Vol.58 of Frontiers in artificial intelligence and applications.

[9] G. W. Hewes (1993) *A history of speculation on the relation between tools and languages*. Gibson and Ingold, 1993 (pp.20-31).

[10] K. R. Gibson, T. Ingold (1993) *Tools, language and cognition in human evolution*. Cambridge University Press.

[11] D. J. Povinelli (2000) *Folk Physics for Apes: The chimpanzee's theory of how the world works*. Oxford University Press.

[12] A. Omicini, A. Ricci, M. Viroli (2008) *Artifacts on the A&A meta-model for multi-agent systems*. Autonomous Agent Multi-Agent Systems 17:432-456.

[13] A. Ricci, M. Viroli, A.Omicini (2008) *The A&A Programming model and technology for devoloping agent environments in MAS*.Programming multi-agent systems, Vol. 4908 of LNAI.

[14] M.Viroli, A. Omicini, A. Ricci (2007) *Engineering MAS environment with artifacts*. Electronic Notes in Theoretical Computer Science 175 (2007) 97-117.

[15] P. Ciancarini (2006) *Coordination models and languages as software integrators*. ACM Computing Surveys, 28(2): 300-302.

[16] D. Gelernter (1985) *Generative communication in Linda*. ACM Transactions on Programming Languages and Systems, 7(1):80-112.

[17] A. Omicini(2009) *Tuple-based Coordination*. Course of Sistemi distributi LA, University of Bologna at Cesena.

[18] A. Omicini, E. Denti (2001) *From tuple space to tuple centres*. Science of Computer Programming, 41(3): 277-294.

[19] A. Omicini, F. Zambonelli (1999) *Coordination for Internet application development*. Autonomous Agent and Multi-Agent Systems, 2(3):251-269.

[20] A. Omicini, E. Denti (2001) *Formal ReSpecT*. Electronic Notes in Theoretical Computer Science, 48:179-196.

[21] A. Ricci, A. Omicini, M. Viroli (2002) *Extending ReSpecT for multiple coordination flows*. Conference on Parallel and Distributed processing Techniques and Applications, volume III, pp.1407-1413.

[22] A. Omicini, A. Ricci, M. Viroli (2005) *Time-aware coordination in ReSpecT*. Coordination Models and Languages, Vol. 3454 of LNCS pp.268-282.

[23] A. Omicini (2007) *Formal ReSpecT in the A&A Perspective*. Electronic Notes in Theoretical Computer Science 175 (2007) 97-117.