AGENTI E MAS PER L'INGEGNERIA DEI SISTEMI DISTRIBUITI

Alessandro Ricci
(aricci@deis.unibo.it)

Seminario per il corso di Sistemi Distribuiti LA - a.a. 2004/2005

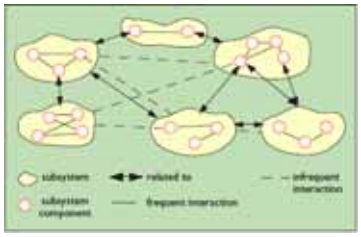
AGENDA

- Ingegneria dei sistemi distribuiti: caratteristiche e problematiche
  - Sistemi distribuiti come sistemi complessi
  - Limiti degli approcci tradizionali
- Paradigma ad Agenti e Sistemi Multi-Agente
  - Visione, Architetture, Modelli
- Ingegneria delle interazioni: modello e infrastruttura di coordinazione TuCSoN
- Conclusioni, appendici e bibliografia

INGEGNERIA SISTEMI DISTRIBUTI ASPETTI e PROBLEMATICHE

Sistemi distribuiti come sistemi complessi

- Sistemi complessi
  - insiemi di parti distinte, separate, *interagenti*



(*) N.R. Jennings. An agent-based approach for building complex software systems. *Communication of ACM*, 44(6):35–41, April 2001

Pervasività Sistemi Complessi

- Esempi familiari...
  - Sistema Operativo
  - Applicazioni di office automation, mail, personal organizers, ..
  - Videogiochi
  - ..
- Esempi più enterprise
  - Business Process Automation (Workflow Management..)
    - Applicazioni on-demand
  - Sistemi di controllo industriali
  - Sistemi per la gestione di treni, aerei, traffico stradale, etc.
  - Ubiquitous Computing
  - ...

Alcune caratteristiche (1/3)

- Distribuzione
  - parti e attività del medesimo sistema risiedono contesti di esecuzione separati
    - rete
  - distribuzione spaziale, distribuzione del controllo
- Concorrenza
  - più attività del medesimo sistema in esecuzione simultaneamente
    - sia nel medesimo contesto di esecuzione, sia distribuite

Alcune caratteristiche (2/3)

- Interazione
  - Comunicazione
  - Interoperabilità
  - Mobilità
  - ...
- Coordinazione
  - Interazione fruttuosa delle parti nel complesso
  - Sistema > somma parti

Alcune caratteristiche (3/3)

- Apertura
  - dinamicità
  - eterogeneità
  - controllo parziale
  - imprevedibilità

“Always on"

- Prospettiva sempre più run-time / *online*
  - 24/7
  - aggiornamento ed evoluzione dinamica, on-the-fly
- Impatto sull’ingegneria del software
  - sui paradigmi
  - sulle metodologie
  - sulle infrastrutture a supporto dei sistemi

Ingegneria dei sistemi distribuiti

- “Quali strumenti per ingegneria di sistemi complessi?”
  - modelli e architetture per la progettazione
  - linguaggi e framework per lo sviluppo
  - infrastrutture per l’esecuzione e la gestione dinamica dei sistemi

Approccio basato su paradigmi tradizionali

- Oggetti, componenti, servizi
  - decomposizione del sistema in entità dotate di interfacce ben definite
  - interazione mediante invocazione di metodi / servizi
  - design by contract
- Architetture client-server e multi-tier
  - pattern MCV
- Architetture service-oriented
  - registrazione, look-up & discovery

Modelli & Tecnologie

- Oggetti distribuiti
  - Infrastrutture: CORBA, RMI
  - Interoperabilità, trasparenza,...
- Architetture multi-tier
  - Servlet, JSP, EJB,...
- Architetture a servizi
  - Infrastrutture: Web Services
    - Standard XML-based: WSDL
    - Interoperabilità, dinamicità...
  - Framework: OSGi
  - Orchestrazione di servizi: BPEL

Alla base del modello: controllo centralizzato

- Applicazioni costituite da un unico flusso di controllo che 'attraversa' i vari componenti
  - interazione come trasferimento di dati *e controllo*
- Entità incapsulano stato e comportamento, ma non il controllo di tale comportamento
- A livello fondazionale, stesso approccio per sistemi centralizzati e distribuiti
  - concettualmente: un unico flusso di controllo

Limiti dell'approccio tradizionale

- Livello di astrazione inadeguato per ingegneria sistemi complessi
  - Concorrenza ?
  - Controllo del sistema ?
    - online management ?
  - Scalabilità ?
  - Adattabilitità e Flessibilità ?
    - self-adaptation ?
  - Tolleranza ai guasti ?
    - self-healing ?

Esempi (1)

- caso di studio: "edificio intelligente"
  - ingegneria di servizi "intelligenti" di supporto alle attività che si svolgono in un certo ambiente
    - es: edificio del DEIS
  - Esempio di servizio: sistema di allarme

Esempi (2)

- caso di studio: "servizio di calcolo parallelo"
  - realizzazione di un servizio per il calcolo di una funzione h(x,y) = f(x)*g(y), sfruttando rete di calcolatori a disposizione
    - f(x) e g(y) funzioni complesse, indipendenti

Verso la distribuzione del controllo

- Dal controllo centralizzato al controllo distribuito
  - incapsulamento del controllo
- Verso la visione ad agenti
  - sistema complesso come *sistema di agenti (MAS)*
  - problem solving come attività collettiva, 'sociale' da parte di un insieme di 'esperti' (agenti), con competenze specifiche, che interagiscono per risolvere il problema nel suo complesso
- Paradigm shift
  - si ribalta totalmente il modo di concepire, progettare e realizzare sistemi

Verso agenti e sistemi multi-agente

- Aspetti first-class
  - distribuzione
  - concorrenza
  - interazione e coordinazione
  - organizzazionbe

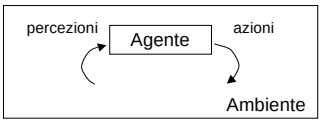
PARADIGMA AGENTI

Nozione di *Agente*

- "Agents everywhere"
  - Nozione interdisciplinare, emersa in vari contesti sia dentro, sia fuori dalla computer science
    - Distributed Artificial Intelligence ('80/'90), Concurrent and Distributed Systems ('90/'00), Software Engineering ('00), Programming Languages ('00), Robotics ('70/'80)
    - Cognitive Science, Economics, Social science,
  - Varie caratterizzazioni, con alcuni aspetti fondazionali in comune
- Paradigma di progettazione e sviluppo del software, in particolare di sistemi complessi
  - Caratterizzazione software-engineering

Una definizione di Agente

- [Jennings & Wooldridge, 1995]
  - "*Any computer system that is **situated** in some environment, and that is capable of **autonomous action** in that environment in order to meet its design **objectives***"
- Qualsiasi entità (computazionale) immersa in un ambiente, con cui ed in cui ***interagisce*** al fine di perseguire un determinato obiettivo, svolgendo una o più attività
  - *sensori e attuatori*



Proprietà fondazionali

- Enitità goal / task oriented / governed
  - progettate per raggiungere uno scopo interno o eseguire un compito
    - comportamento pro-attivo
- Autonomia
  - comportamento autonomo nel raggiungere gli obiettivi
  - Controllo delle proprio stato e comportamento
- Interazione e ambiente (situatedness)
  - Percezioni e azioni come interazione
    - sensori e attuatori
    - Reattività agli eventi dell'environment
  - Osservazione e controllo parziale

Esempi di agenti

- Esempi 'accademici'
  - Termostato (= qualsiasi sistema di controllo)
  - BOT(s) in videogames
  - LEGO Mindstorm Robot
  - ...
- Reali e più complessi
  - Personal User Agent
    - Mail, calendar,...
  - Printer spooler (= qualsiasi demone software)
  - Information searcher in Internet
  - Business Process Agents
  - Unmanned Vehicles/Planes pilots
  - Robots
  - ....

Termostato

- Obiettivo/task: "mantenere la temperatura di un ambiente ad un dato valore costante"
  - Comportamento autonomo
    - quando la temperatura è superiore a X di N1 gradi, aziona il condizionatore a potenza 1; quando è superiore di N2 gradi (N2>>N1) aziona il condizionatore a potenza 2
    - quando la temperatura è inferiore a X di M1 gradi, aziona il termosifone a potenza 1; quando è superiore di M2 gradi (M2<<M1) aziona il termosifone a potenza 2
    - Se la temperatura è compresa fra N1 e M1 mantieni spenti gli apparati

Termostato *intelligente*

- Obiettivi più complessi
  - es: risparmio energetico
- Comportamento più articolato
  - in base allo storico della giornata,...
- ...

Task & Goal

- Esecuzione di *task* per conseguire / mantenere determinato obiettivo
  - L'architettura / modello computazionale dell'agente caratterizza il modo con cui l'agente decide lo svolgimento del task, *quali azioni compiere* in virtù delle *percezioni* e degli *obiettivi*
  - Vari modelli e architetture
    - imperativi, dichiarativi, basati su regole, modelli cognitivi, modelli reattivi / subsimbolici...
- Agente come entità *intenzionale*
  - "Intentional stance" vs. "Design stance"

Autonomia

- Più livelli e caratterizzazioni, da socio-psicologiche a puramente ingegneristiche
  - "il governarsi da sé, sulla base di leggi proprie"
  - "Capacità di sopravvivere in un ambiente con cui si interagisce, senza intervento esterno"
- Incapsulamento di *stato*, *comportamento* e *controllo di entrambi*
  - decisione se e quale azione eseguire, percezione / messaggio percepire
- Incapsulamento flusso di controllo (*thread of control*)
  - esecuzione attività indipendente

Autonomia: Agenti vs. Oggetti

- Primo principio fondazionale che differenzia le astrazioni
  - oggetti incapsulano stato e comportamento, ma non il controllo
  - negli oggetti il controllo fluisce da un oggetto ad un altro, con l'invocazione di metodi
- Oltre il "Design by Contract"
  - Agenti ***non hanno interfacce***
    - interfacce OO (come insieme di operazioni) violerebbero l'incapsulamento del controllo
  - *Pre-condizioni* di una richiesta (client/environment) possono cambiare durante l'attività di soddisfacimento della richiesta stessa (agente), causa apertura
    - Nel'OO le precondizioni non variano

Interazione e Ambiente

- Modelli di interazione
  - Astrazioni e meccanismi per modellare l'interazione con environment
    - risorse + altri agenti
  - Interazione
    - percezioni e azioni
      - includono comunicazione, uso risorse condivise, ...
    - più in generale, effetto dinamico di *dipendenze* presenti fra un agente e il suo ambiente

Modelli, Linguaggi, Architetture...

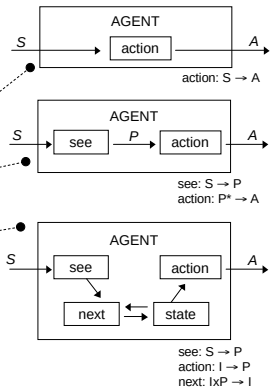
- MAS come paradigma 'stato-dell'arte' nella ricerca
  - Modelli e architetture
    - progettazione
  - Linguaggi
    - sviluppo

Agent abstract loop semplificato

```
while (living) {
  percept();
  compute(); [reason,plan,...]
  act();
}
```


Classi architetturali di base

- Puramente reattivi
- Percezione
- Con stato



S:set of input (environment partial state)
A:set of actions
P:set of perceptions
I: set of agent internal states

Tipi specifici di architetture

- Architetture reattive
  - Automi a stati finiti
  - Architetture sussunzione
  - Situated automata
- Architetture logiche
  - Rules-based
  - Architetture deliberative / cognitive
    - BDI (Belief Desire Intention)
- Architetture ibride
  - A livelli orizzontali e verticali

Architetture logiche

- Rappresentazione esplicita simbolica della conoscenza
  - ambiente, obiettivi, comportamento
    - forma di regole, piani, etc
- Comportamento da intraprendere determinato a partire di forme di ragionamento simbolico sulla conoscenza e sulle interazioni correnti
  - Manipolazione sintattica basata su teorie logiche, deduttive, theorem-proving...
  - Es: Agenti *deliberativi*
    - stato interno come insieme di formule della logica dei predicati del prim'ordine
    - Scelta dell'azione da compiere mediante insieme di regole deduttive

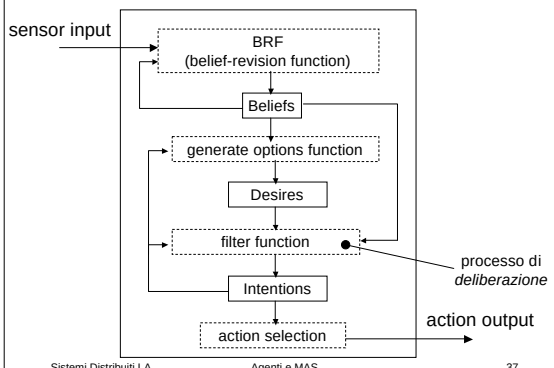
Modelli BDI (Belief-Desire-Intention)

- Radici nella tradizione filosofica del *practical reasoning*
  - processo di decisione dell'azione da intraprendere, istante per istante, per arrivare ad un dato obiettivo
    - (1) quale goal perseguire
    - (2) come perseguire il goal scelto...
- Agente modellato in termini cognitivi / mentalistici
  - ***Beliefs***
    - conoscenza/credenze che l'agente ha del mondo, dell'environment
  - ***Desires***
    - opzioni a disposizione di un agente in virtù degli obiettivi da raggiungere
  - ***Intentions***
    - quelle opzioni che l'agente sceglie di perseguire fra le disponibili (*commit*), divenendo intenzioni
    - è il 'focus' del comportamento dell'agente

Intenzioni

- Proprietà
  - Guidano l'analisi/ragionamento mezzi-fini (*means-ends* reasoning)
    - tecniche dell'ambito AI
  - Vincolano lo spazio delle opzioni future
    - l'agente può evitare la generazione di desideri inconsistenti con le sue intenzioni attuali
  - Sono persistenti
    - fino al momento in cui l'agente crede di aver raggiunto l'obiettivo
- Punto chiave: bilanciare questi aspetti
  - bilanciare comportamento pro-attivo (goal-directed) e reattivo (event-directed)
  - fondamentale saper revisionare/cambiare dinamicamente intenzioni in virtù dei cambiamenti dell'ambiente

Architettura semplificata di un agente BDI

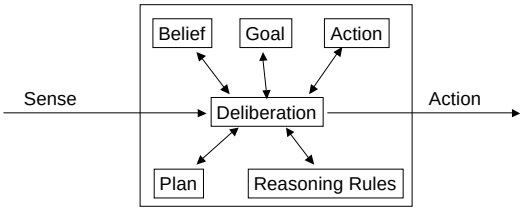


Esempio: 3APL

(IICS/ISG, Utrecht University, Amnsterdam)

- Architettura + Linguaggio per agenti *cognitivi*
  - Beliefs, Goals, Desires, Intentions, Obligations, Actions, Plans, Communication
- Osservazioni e credenze (beliefs) come first order logic
  - fatti e regole Prolog-like
- Azioni come triple
  - action_name
  - pre-conditions: condizioni per eseguire le azioni (beliefs)
  - post-conditions: effetti delle azioni (beliefs)
- Goal procedurali o *to-do goals*
  - espressioni secondo un modello imperativo, con sequenze, test, ricorsione, etc.
- Reasoning rules utilizzate per la generazione di goals, revisione di azioni bloccate, revisione di goal non raggiungibili, etc.

3APL: architettura



Pro e contro architetture logiche

- Pro
  - Rappresentazione e manipolazione esplicita della conoscenza e dei relativi processi che portano all'azione
    - Vasta letteratura in ambito AI
  - Approccio elegante
  - Semantica chiara (basata su teorie logiche)
- Contro
  - Difficoltà di rappresentare a livello simbolico l'ambiente e la sua evoluzione dinamica
    - logiche modali, temporali,...
  - Complessità computazionale
    - non trattabilità

Modelli Reattivi

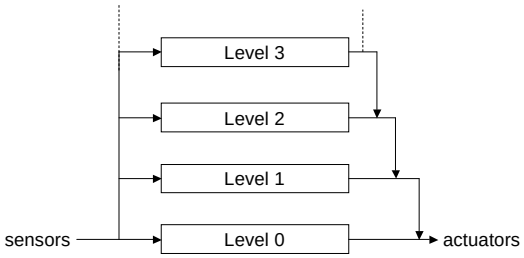
- Visione
  - comportamenti "intelligenti" possono essere ottenuti senza una *rappresentazione* simbolica esplicita
  - comportamenti "intelligenti" possono essere ottenuti senza forme di ragionamento astratto (simbolico)
  - Intelligenza come proprietà emergente
- Due idee chiave (Brooks)
  - intelligenza richiede situatedness
    - interazione con environment
  - intelligenza deriva dall'interazione con l'ambiente
    - "I*ntelligence is in the eye of the beholder*"

Architettura a sussunzione (*)
(*subsumption architectures)*

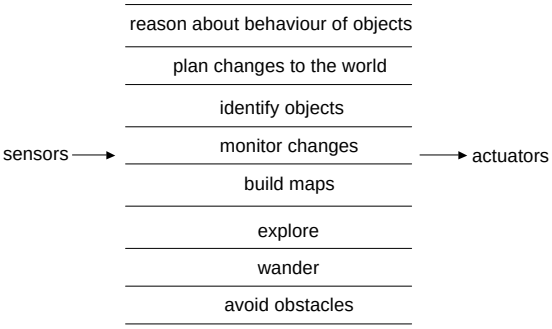
- Gerarchia di comportamenti task-oriented (*behaviours*) semplici
- Controllo (comportamento) dell'agente dato dalla *competizione* fra i comportamenti
  - Livelli inferiori → comportamenti primitivi (es: scansare ostacoli)
  - Livello superiori → comportamenti alto livello
  - Livelli inferiori hanno priorità rispetto a quelli inferiori
- "Intelligenza" come proprietà emergente
  - interazione ambiente
  - competizione di comportamenti semplici

(*) "A Robust Layered Control System for Mobile Robots"- Rodney Brooks, 1985

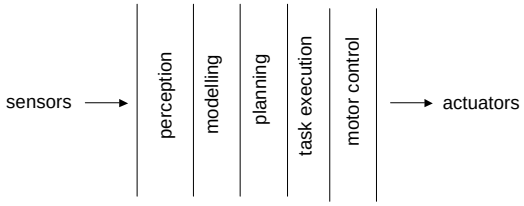
Layered control in una architettura a sussunzione



Esempio: decomposizione comportamento robot in task achieving behaviours



Confronto con decomposizione tradizionale su moduli funzionali



Mars explorer system (Steels)

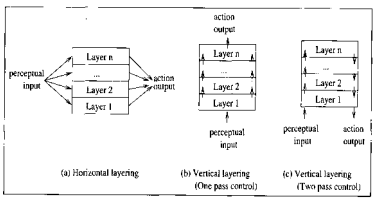
- Architettura a sussunzione per il comportamento di robot per la raccolta di rocce su Marte
  - comportamento quasi-ottimo
- 5 livelli
  - [lev 0] if detect an obstacle then change direction
  - [lev 1] if carrying samples and at the base then drop samples
  - [lev 2] if carrying samples and not at the base then travel up gradient
  - [lev 3] if detect a sample then pick up sample
  - [lev 4] if true then move randomly

Pro e contro architetture reattive

- PRO
  - semplicità
  - computazionalmente trattabili
  - robusti alle failure
  - eleganti
- CONTRO
  - difficoltà di gestire agenti con comportamenti complessi
  - difficoltà di gestire agenti con ampio numero di comportamenti
  - difficoltà di ingegnerizzazione agenti
    - non esistono metodologie robuste
  - local environment vs. non local information
    - incapacità di affrontare problemi che richiedono decisioni a partire da informazioni non recuperate da interazioni con environment locale

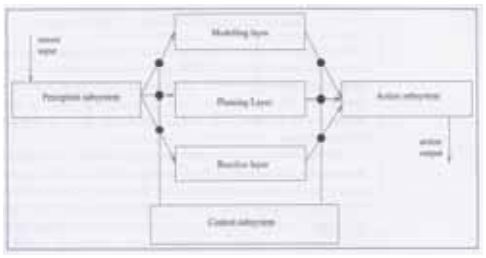
Architetture ibride

- Agenti con due o più sottosistemi eterogenei
  - *parte deliberativa*, contenente modello simbolico del mondo, che sviluppa piani, e attua forme di reasoning
  - *parte reattiva*, in grado di reagire ad eventi senza la necessità di ragionamenti complessi
- Architetture a livelli



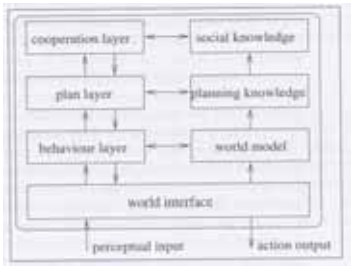
Livelli orizzontali: TOURINGMACHINES

- 2 sottosistemi per interazione con ambiente (percezione e azione) + 3 livelli di controllo



Livelli verticali: InteRRaP

- Architettura a 2-passate



Flessibilità

- Dal design stance all'intentional stance
  - comportamento dell'agente non stabilito a priori dal designer, ma dinamicamente, in base alle interazioni con l'ambiente e all'evoluzione degli obiettivi
- Capacità di far fronte all'apertura dell'ambiente
  - adattandosi ai suoi cambiamenti per raggiungenere in ogni caso i propri obiettivi
- Far fronte alla conoscenza parziale del designer dell'agente
  - capacità di determinare dinamicamente il proprio comportamento al fine di raggiungere gli obiettivi prefissati
- Differenza sostanziale rispetto agli oggetti
  - oggetti hanno un comportamento specificato completamente dal designer

Altri esempi interessanti

- Situated Automata
  - Evoluzione automi a stati finiti (teoria dei sistemi) verso architetture reattive
  - ("Situated automata" - Rosenschein and Kaelbling)
- Model-based systems(*)
  - Teoria dei sistemi di controllo + tecniche e modelli AI (programmazione a vincoli)
  - Sistemi di controllo avanzati
    - prossime spedizioni su Marte

(*)("Model-based programming of Intelligent Embedded Systems and Robotic Space Explorers" - Williams, Ingham, Chung, Elliott, 2003)

Linguaggi Agent-Oriented

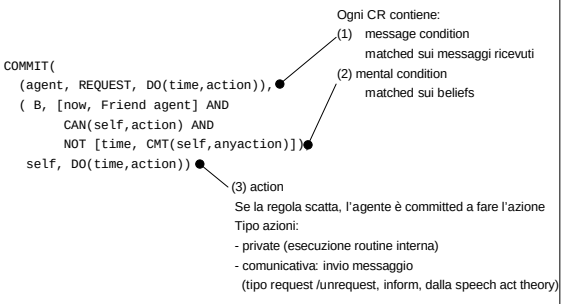
- "Ma abbiamo bisogno di linguaggi agent-oriented?"
  - questione ancora dibattuta nella ricerca
- Tipi di risposte (nella "comunità agenti")
  - *NO!*
    - C, Java, Prolog + patterns, architetture, Infrastrutture
  - *SI!*
    - 3APL, GO! (ed Agent-0, AgentSpeak, PLACA,...)
  - *SI, estendendo il mainstream (JAVA)*
    - JACK, JASON, JADEX,...
- ... fuori dalla comunità agenti
  - *SI*
    - es: Model-Based Programming: RMPL

Agent-0

- Primo tentativo di linguaggio agent-oriented, per *agenti cognitivi*
  - "new programming language, based on societal view of computation" (*)
- agente espresso in termini *mentalistici,* dalle teorie ad agenti esistenti
  - capabilities
    - cosa l'agente sa fare
  - initial beliefs
    - cosa l'agente conosce
  - initial committments
    - obiettivi dell'agente
  - insieme di committment rules
    - come arrivare agli obiettivi

(*) Shoham. Agent-oriented programming. Artificial Intelligence, 60(1):51-92, 1993

Esempio di committment rule



Esempio stato dell'arte: 3APL

- Programmazione dell'agente in termini mentalistici
  - Belief (credenze)
    - Base di conoscenza come insieme di formule sottoinsieme della logica del prim'ordine
  - Goals and actions
    - "Goals-to-do": supporto per versione procedurale dei goals
    - Espressi come composizione esplicita di goal più semplici, mediante costrutti imperativi
    - Azioni come forma più semplice di goal
  - Regole per il practical reasoning
    - supporto diretto per la manipolazione, composizione dei goal dell'agente

3APL: un esempio

```
PROGRAM "robot"
CAPABILITIES:
    { pos(X,Y) } West() { NOT pos(X,Y) , pos(X - 1,Y) }
BELIEFBASE:
    pos(100,10).
GOALBASE:
    goWest()
RULEBASE:
    goWest() <- pos(0,Y) | SKIP .
    goWest() <- pos(X,Y) AND X>0 | BEGIN West();goWest() END
```

In this example, it is assumed that the agent, called robot, is situated in a grid having a position specified by the pair (X,Y). This agent can perform only one action, i.e. West(), after which the horizontal position of the agent (X-value) is decreased by one. Initially, the agent believes that it is situated at the position (10,10) and wants to achieve the goal goWest(). How the goal can be reached is specified by two rules. The first rule indicates that the goal has been achieved if the agent is at the position (0,Y) and the second rule specifies that the goal can be achieved by performing the action West() after which the goal should be achieved again.

Esempi di agenti Java-based: JADE

- Piattaforma per lo sviluppo e deployment di agenti Java-based FIPA-compliant
  - FIPA ACL per la comunicazione (vedere in seguito)
- Modello agente non necessariamente cognitivo, ma con attività strutturata in *comportamenti*
- Agenti realizzati come pattern sul linguaggio Java
  - Pattern state-behaviour
  - Classe Agent
  - Classe Behaviour

JADE: un esempio

```
import jade.core.Agent;
import jade.core.behaviours.*;

public class MyAgent extends Agent     {
  protected void setup()          {
    addBehaviour( new MyBehaviour( this ) );
  }
  class MyBehaviour extends SimpleBehaviour{
    public myBehaviour(Agent a) {
      super(a);
    }
    public void action() {
      //...this is where the real programming goes !!
    }
    private boolean finished = false;
    public boolean done() {
      return finished;
    }
  } // ---------- End myBehaviour
}//end class myAgent
```


Esempio di estensioni: JACK

- JACK è una piattaforma per lo sviluppo di agenti basati su una *estensione* del linguaggio Java
  - aggiunti costrutti per specificare aspetti di tipo mentalistico
  - Modello di riferimento: BDI
    - desires → obiettivi
    - intentions → piani per raggiungere obiettivi
    - beliefs → conoscenza del mondo

Costrutti nuovi JACK

- Agent
  - comportamento dell'agente
  - include capabilities, eventi/messaggi, piani
- Capability
  - incapsula piani, beliefset, eventi per il riuso
- BeliefSet
  - conoscenza dell'agente in un modello relazionale
- View
  - query sulla conoscenza
- Event
  - eventi significativi per l'agente, a cui deve reagire con opportune azioni
- Plan
  - istruzioni per raggiungere determinati obiettivi

Aspetto di un agente JACK

```
event MyEvent extends
        BDIGoalEvent {
  String text; // For example.
  #posted as
       myPostingMethod(String s){
    text = s;
  }
}
```

```
plan MyPlan extends Plan {
  #handles event MyEvent me;
  body(){
    System.out.println(me.text);
  }
}
```

```
agent MyAgent extends Agent {

  #handles event MyEvent;
  #uses plan MyPlan;
  #posts event MyEvent myEventRef;

  MyAgent(String name){
    super(name);
  }
  public void method1(String exampleMessage){
    postEvent(myEventRef.myPostingMethod(exampleMessage));
  }
}
```

```
public class Program
{
  public static void main(String args[]){
    MyAgent agent1 = new
                        MyAgent("agent1");
    agent1.method1("data to be posted");
  }
}
```


Esempio di Model-based Programming: RMPL

- RMPL = Reactive-Model-based-Programming-Language
  - Synchronous/reactive language + constraint programming
- Model-based programming
  - Specifica di strategie di controllo ad alto livello, lasciando all'esecutore model-based il reasoning necessario 'on-the-fly'
    - tracking system state, diagnosing faults, perform reconfigurations
- Esecutore di RMPL: Titan
  - esegue i programmi RMPL utilizzando modelli dichiarativi component-based del sistema da controllare, al fine di fare il tracking dello stato, analizzare situazioni anomale, generare sequenze di controllo
  - Motore inferenziare per dedurre lo stato attuale del sistema, quindi lo stato futuro desiderato
  - *Model-based reactive planning* per far transitare il sistema dallo stato corrente allo stato desiderato

RMPL: un esempio

- Programma di controllo nell'inserimento in orbita di una navicella spaziale (*)

```
OrbitInsert()::{
  do {
    EngineA = Standby,
    EngineB = Standby,
    Camera = Off,
    do {
      when EngineA = Stanby AND Camera = Off
        doNext EngineA = Firing
    } watching EngineA = Failed,
    when EngineA = Failed AND
         EngineB = Standby AND Camera = Off
      donext EngineB = Firing
  } watching EngineA = Firing OR
             EngineB = Firing
}
```

(*) "Model-based programming of Intelligent Embedded Systems and Robotic Space Explorers" - Williams, Ingham, Chung, Elliott, 2003

Agenti come pattern

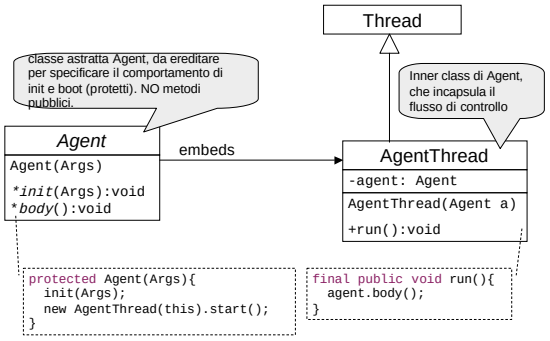
- Obiettivo: realizzare l'astrazione di agente in termini di pattern su linguaggi mainstream
  - Object-oriented (es: Java)
  - Liguaggi dichiarativi (es: Prolog)
  - ...
- Letteratura abbondante
  - in particolare: lavoro della Kendall (*)

(*) "A Java Application Framework for Agent Based Systems" - Kendall, Krishna, Pathak, Suresh (RMIT)

Esempio semplice su linguaggi OO

- Agente non ha interfaccia à-la OO
  - Nessun metodo pubblico
    - "public methods considered harmful"
- Ogni interazione con l'environment deve avvenire mediante le astrazioni/meccanismi fornite dal modello di interazione
  - Può sfruttare e interagire con oggetti / servizi direttamente (API Object-Oriented), se queste sono viste come sue risorse private
- Incapsulamento flusso di controllo

Agente di base: bozza di pattern



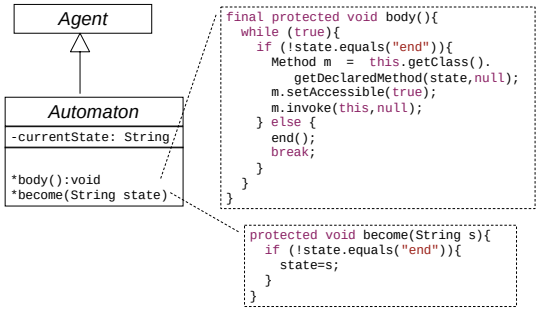
Esempio più articolato: automa a stati

- Agente come *automa a stati*
  - il comportamento viene definito come insieme di stati, fornendo la possibilità di specificare la transizione di stato
  - alla nascita l'agente è nello stato boot
  - transitare nello stato end = morte
- In generale 2 possibili design
  - Reflection pattern
  - State-Behaviour pattern

Reflection pattern

- Stati/Comportamenti definiti da metodi protetti della classe agent
  - alcuni di default: `idle, boot, end`
  - nome del metodo come identificatore dello stato
- Meccanismo per specificare transizione di stato
  - metodo `become(String stateName)`
- Si può ottenere estendendo il pattern di base, specializzando il comportamento di `body`

Automi con reflection: bozza di pattern



State-Behaviour pattern

- Stati / comportamenti rappresentati direttamente come classi
  - riferimento al pattern State (*)

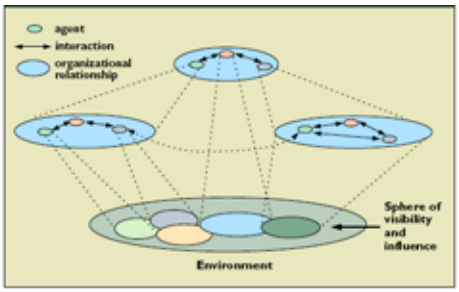
(*) "Design Pattern" -Gamma et al. - Addison Wesley

DAGLI AGENTI AI SISTEMI MULTI-AGENTE (MAS)

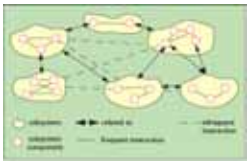
Applicazioni come sistemi multi-agente (MAS)

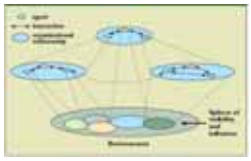
- Sistema multi-agente (MAS)
  - insieme di agenti con ruoli / compiti diversi che interagiscono in modo fruttuoso in un certo ambiente al fine realizzare scopi / attività del sistema *nel suo complesso*
    - attività individuali e *sociali / collettive*
- Applicazioni progettate e sviluppate come sistemi multi-agente

MAS: una rappresentazione



Decomposizione per obiettivi / compiti (task-oriented - goal-oriented)





- Ad ogni livello, i sottosistemi lavorano cooperativamente per raggiungere le funzionalità globali del loro sistema 'padre'
- All'interno di un sottosistema, i componenti interagiscono strettamente per fornire nel complesso le funzionalità del sotttosistema
- loci di controllo multipli → naturale associare incapsulamento del controllo per ogni attività / obiettivo

Aspetti centrali

- Interazione
  - come comunicano gli agenti? quali linguaggi di comunicazione?
  - come interagiscono con l'ambiente? cos'è l'ambiente?
- Coordinazione
  - come interagiscono in modo fruttoso gli agenti? In che modo si raggiungono obiettivi collettivi, aldilà degli obiettivi individuali?
  - Nello specifico: come sincronizzare attività di agenti distinti? come condividere e accedere concorrentemente alle risorse senza conflitti? Come suddividersi i compiti in un problema complesso?
- Organizzazione
  - Come strutturare l'insieme degli agenti?

Modelli di Interazione: knowledge level

- Interazioni condotte al *knowledge level*
  - Obiettivi da raggiungere, quando, da chi
    - aspetto semantico (cosa significa) e pragmatico (come usare) della comunicazione
  - Differenza sostanziale rispetto all'interazione mediante esecuzione di operazioni / invocazione di metodi
    - no trasferimento di controllo

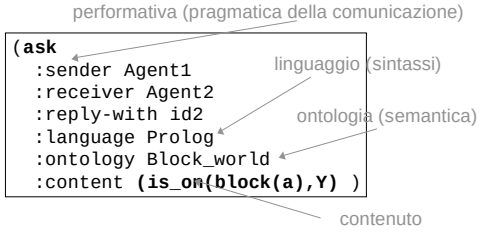
Interazione diretta e mediata

- Modelli di interazione / coordinazione classificabili in due grandi famiglie:
  - Interazione/comunicazione diretta
    - si astrae dal mezzo che realizza la comunicazione
    - da segnali a scambio di messaggi, fino a ACL (Agent Communication Language) di alto livello
  - Interazione *mediata*
    - rappresentazione esplicita delle astrazioni che gli agenti usano per comunicare
      - Message box, lavagne, ...

Agent Communication Languages basati su Speech Acts

- Comunicazione basata su scambio diretto di messaggi che specificano *azioni*
  - l'emittente esegue un atto di comunicazione al fine di cambiare il mondo, in particolare lo stato conoscitivo di colui che ascolta
    - "Saying it makes it so"
  - Ogni messaggio ha 3 parti fondamentali
    - Performativa (pragmatica)
    - Contenuto (sintassi)
    - Ontologia (semantica)
- Esempi: KQML (Knowledge Query Manipulation Language) e FIPA ACL
  - Protocolli standard per scambiare informazioni e conoscenza

KQML: esempio (1)

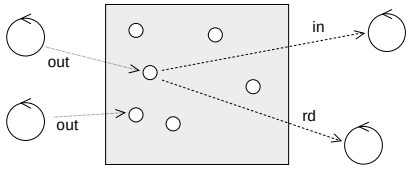


KQML: esempio (2)

```
(tell
  :sender Agent2
  :receiver Agent1
  :in-reply-to id2
  :language Prolog
  :ontology Block_world
  :content (is_on(block(a),block(b)) )
```


Un modello ad interazione mediata: spazi delle tuple (*)

- S*pazi* di informazione *condivisi* dove gli agenti inseriscono e prelevano informazioni (tuple) *in modo associativo*



(*)"Generative Communication in Linda" - Gelernter, TOPLAS, 1985

Esempio: Modello Linda / TuCSoN

- Tuple logiche
  - tuple come termini logica prim'ordine
- Inserimento tuple: **out**(T)
- Rimozione tupla: **in**(TT)
  - TT tuple template (specifica un insieme di tuple)
  - Non appena una tupla che fa match con TT è disponibile, viene rimossa e data all'agente
- Lettura tupla: **rd**(TT)
  - TT tuple template (specifica un insieme di tuple)
  - Non appena una tupla che fa match con TT è disponibile, viene letta (senza rimozione) e data all'agente
- Rimozione tupla con eventuale fallimento: **inp**(TT)
- Lettura tupla con eventuale fallimento: **rdp**(TT)

Organizzazione e Coordinazione

- Dimensioni sistemiche fondamentali, legate alla dimensione dell'interazioni
- Modelli Organizzazionali
  - Struttura e relazioni strutturali dei sistemi
  - Nozione di ruolo, di gruppi / società / organizzazioni, di relazioni fra ruoli, fra ruoli e risorse
- Modelli di Coordinazione
  - Processi e comportamenti dinamici dei sistemi
  - Regole/Leggi di coordinazione che costruiscono e veicolano le interazioni verso gli obiettivi del sistema
  - Gestione delle dipendenze: dalla comunicazione al workflow, della condivisione di risorse ad aspetti di sincronizzazione...
- Modelli correlati: sicurezza....

Organizzazione e Coordinazione: Agenti vs. Oggetti

- Organizzazione
  - Oggetti → + statica
    - relazioni fra classi, specifica/verifica al compile time
  - Agenti → + dinamica
    - ruoli e relazioni inter-ruolo, specifica/verifica al runtime
- Coordinazione
  - Oggetti → pattern
    - Mediator, MVC, Event-Listener, Publish-Subscriber...
  - Agenti → modelli di interazione e coordinazione
    - negoziazione
    - cooperazione

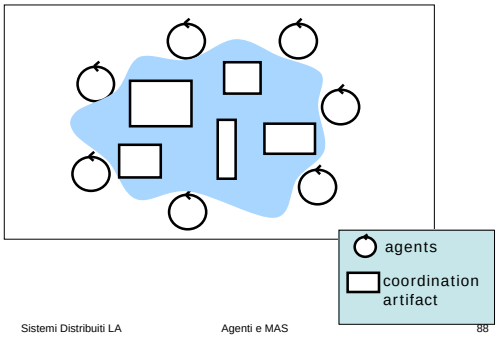
Coordinazione nei MAS

- Moltitudine di approcci
  - modelli tratti da scienze cognitive, sociali, economiche, biologiche, etologiche
- Due grandi famiglie
  - Coordinazione *soggettiva*
    - coordinazione totalmente a carico degli agenti e delle loro capacità di percepire, ragionare, agire
    - approccio principale nell'ambito DAI (Distributed artificial Intelligence)
  - Coordinazione *oggettiva*
    - approccio ingegneristico alla coordinazione
    - progettazione e sviluppo di opportune astrazioni per agevolare le attività di coordinazione degli agenti
  - A mezza via...
    - Coordinazione mediante *protocolli* di comunicazione

Coordinazione come ingegneria delle interazioni

- Definizione, progettazione, sviluppo di astrazioni di *artefatti di coordinazione*
  - astrazioni di prima classe per supportare l'e attività di comunicazione e coordinazione fra agenti
  - tipi diversi di artefatti, a vari livelli di complessità
    - Esempi: message box, lavagne, semafori, sincronizzatori, mappe, workflow engine, ...
- Prospettiva dell'interazione mediata
  - ciò che abilita l'interazione fra gli agenti diventa oggetto e luogo di ingegneria per realizzare coordinazione

Agenti e artefatti di coordinazione



Ruolo delle Infrastrutture

- Infrastruttura come middleware che fornisce un insieme di servizi a disposizione degli agenti per supportare le loro attività (strutture e processi)
  - sussistenza
  - organizzazione
  - comunicazione / Interoperabilità semantica
  - coordinazione
  - sicurezza, mobilità,...
  - ..
- Prospettiva *runtime* dei MAS
  - sistemi always on
  - tools di supporto per validazione, testing, debugging *runtime*
- Impatto sulle metodologie
  - infrastrutture come sorgenti di astrazioni per la progettazione

Infrastrutture di coordinazione

- Infrastrutture specializzate nel fornire servizi a supporto delle attività di coordinazione
  - dalla comunicazione e interoperabilità, a sincronizzazione, gestione del workflow, etc
- Portano a runtime le astrazioni di coordinazione definite a livello di modello (a design/development time)
  - Es: infrastrutture di supporto per artefatti di coordinazione
    - Gestione della creazione, utilizzo, ispezione, adattamento artefatti di coordinazione

“keep the abstraction alive”

- Principio fondamentale per l’ingegneria dei sistemi complessi
  - agenti e artefatti come astrazioni di prima classe anche a runtime, supportate da apposite infrastrutture
  - runtime come fase soggetta processo ingegneristico
    - “online engineering”
  - osservabilità, controllabilità, capacità di diagnosticare problemi, validare sistemi, etc

Esempi di Infrastrutture MAS

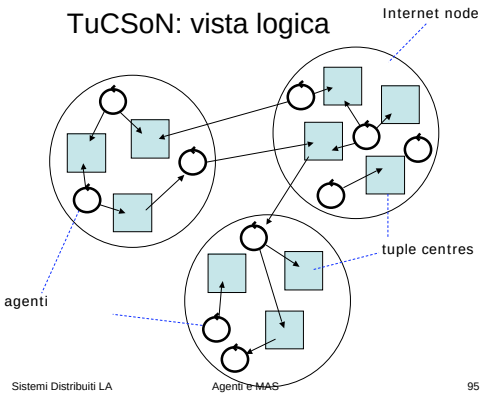
- JADE
  - Sussistenza e servizi di base per gli agenti, compliant con lo standard FIPA
    - Comunicazione via FIPA ACL
    - Servizi: Pagine Gialle,...
  - Agenti scritti in Java
- TuCSoN
  - Infrastruttura di coordinazione
  - Tuple centre come artefatti di coordinazione programmabili, general purpose
  - Tecnologia TuCSoN (1.4.0): supporto per agenti scritti in Java, Prolog

INFRASTRUTTURA di COORDINAZIONE TuCSoN

TuCSoN

- **TuCSoN** = *Tu*ple *C*entre *S*pread *o*ver the *N*etwork (Omicini, Zambonelli)
  - Infrastruttura che fornisce servizi per la comunicazione e coordinazione in sistemi multi-agente
- **Tuple centres (centri di tuple)** come artefatti di coordinazione general purpose
  - spazi di tuple *programmabili*
  - comportamento di coordinazione programmabile dinamicamente
- Tuple centres vivono nei nodi dell’infrastruttura, distribuiti per la rete
  - un applicazione/sistema su TuCSoN = insieme di nodi TuCSoN + agenti che interagiscono e si coordinano mediante tuple centres dei nodi

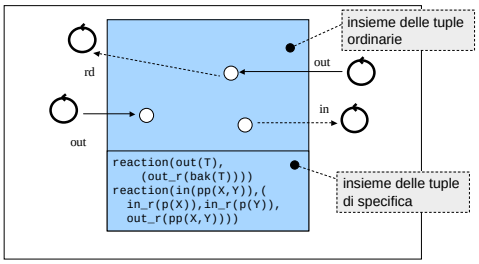
TuCSoN: vista logica



Centri di tuple come artefatti di coordinazione

- Spazi di tuple *programmabili* (*)
  - primitive di coordinazione out, in, rd, inp, rdp come operazioni per inserire, leggere, rimuovere associativamente tuple
    - tuple *logiche* come linguaggio di comunicazione
  - linguaggio **ReSpecT** per programmare dinamicamente il comportamento reattivo del tuple centre, definendo vere e proprie leggi di coordinazione
    - servizio di coordinazione fornito dal tuple centre come artefatto è definito dalle specifiche in ReSpecT

Tuple centre: una rappresentazione



ReSpecT

- Programma ReSpect = insieme di *reazioni*
  `reaction(`*`Event, Body`*`).`
- Una reazione specifica un insieme di *azioni* con cui si manipola/trasforma l'insieme delle tuple
  - primitive ReSpecT: out_r, in_r, no_r, current_agent...
- Le reazioni vengono scatenate in seguito as *eventi*
  - di comunicazione
    - esecuzione di primitive di coordinazione come la out, in, rd,...
  - di coordinazione
    - esecuzione di primitive interne come out_r, in_r, no_r
- Una reazione può aver successo o fallire causa il fallimento di una delle azioni delle primitive
  - fallimento atomico

ReSpecT: esempi di reazioni

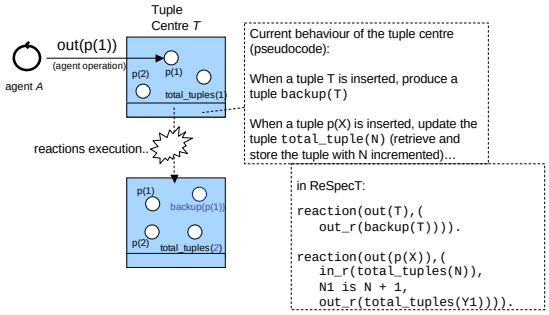
per ogni tupla immessa viene creata una copia di backup

```
reaction(out(T),(
      out_r(backup(T)) )).
```

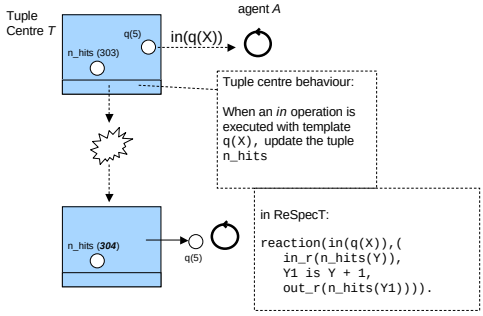
viene mantenuto il conteggio dei valori di temperatura inseriti

```
reaction( out(temperature(City,Value)),(
      in_r(nvalues(N)),
      N1 is N + 1,
      out_t(nvalues(N1)))).
reaction( in(temperature(City,Value)),(
post,  in_r(nvalues(N)),
      N1 is N - 1,
      out_t(nvalues(N1)))).
```


Esempio di reazione (1)

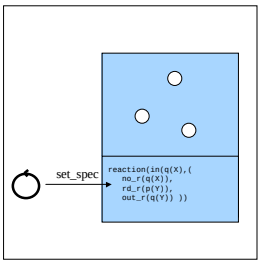


Esempio di reazione (2)



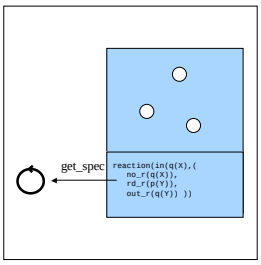
Cambiamento dinamico delle specifiche di coordinazione

- Il comportamento di coordinazione del tuple centre può essere cambiato dinamicamente
- Operazione **set_spec** con cui si setta una nuova specifica ReSpecT



Ispezionamento dinamico delle specifiche di coordinazione

- Il comportamento di coordinazione del tuple centre può essere ispezionato dinamicamente
- Operazione **get_spec** con cui si legge la specifica ReSpecT con cui è attualmente programmato il tuple centre

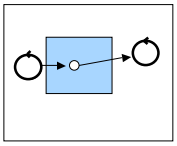


Pattern di coordinazione in TuCSoN

- Tuple centre come artefatti di coordinazione general purpose 'plasmabili' dinamicamente
  - comportamento di coordinazione specifico programmabile in ReSpecT
- Verso un catalogo di artefatti (pattern)
  - Artefatti per comunicazione, sincronizzazione, mediazione semantica, workflow...
  - Riuso

Comunicazione (1)

- Message Passing



SENDER:

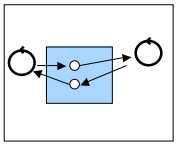
```
out(msg(agentB,content('test',13)))
```

RECEIVER (called agentB):

```
in(msg(agentB,Info))
```


Comunicazione (2)

- RPC style



SERVICE USER:

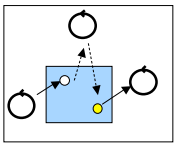
```
…
out(compute_sum(5,8,me))
in(compute_sum_result(me,Value)
…
```

SERVICE PROVIDER

```
in(compute_sum(X,Y,Who))
Sum ← X + Y
out(compute_sum_result(Who,Sum))
```


Interoperabilità (1)

- Mediazione sintattica / semantica fra ontologie di comunicazione diverse



SERVICE USER:

```
…
out(compute_sum(5,8,me))
in(compute_sum_result(me,Value)
…
```

SERVICE PROVIDER

```
in(make_sum(term(X,Y)))
Sum ← X + Y
out(sum_result(X,Y,Sum))
```

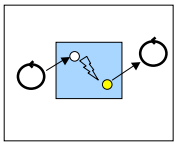
SERVICE MEDIATOR

```
in(compute_sum(X,Y,Who))
out(service_requested(sum(X,Y),Who))
```

```
in(sum_result(X,Y,Sum))
in(service_requested(sum(X,Y),Who))
out(compute_sum_result(Who,Sum))
```


Interoperabilità (2)

- Sfruttando la programmabilità del centro di tuple



TuCSoN Style

SERVICE USER:

```
…
out(compute_sum(5,8,me))
in(compute_sum_result(me,Value)
…
```

SERVICE PROVIDER

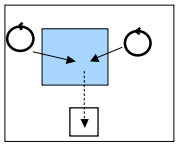
```
in(make_sum(term(X,Y)))
Sum ← X + Y
out(sum_result(X,Y,Sum))
```

MEDIATION POLICY in ReSpecT

```
reaction(out(compute_sum(X,Y,Who)),(
    in_r(compute_sum(X,Y,Who)),
    out_r(service_requested(sum(X,Y),Who)),
    out_r(make_sum(term(X,Y)) )).
reaction(out(sum_result(X,Y,Sum)),(
    in_r(sum_result(X,Y,Sum)),
    in_r(service_requested(sum(X,Y),Who)),
    out_r(compute_sum_result(Who,Sum)) )).
```

Sincronizzazione (1)

- Semplice semaforo per regioni critiche



Synchronised agent:

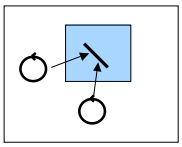
```
<outside critical region>
…
in(token)
<inside critical region>
out(token)
…
<outside critical region>
…
```

HYPOTHESIS:
Initial space content with the tuple `token`

To have synchronised region allowing N users inside
→ N tuples `token`

Sincronizzazione (2)

- Barriera di sincronizzazione a 2



Agent A:

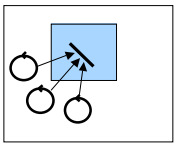
```
…
<before barrier>
…
out(reay(agentA))
rd(ready(agentB))
<agents A and B
are now synchronised>
```

Agent B:

```
…
<before barrier>
…
out(ready(agentB))
rd(ready(agentA))
<agents B and A
are now synchronised>
```


Sincronizzazione (3)

- Barriera di sincronizzazione a 3+ agenti



Agent A:

```
…
out(ready(agentA))
rd(ready(agentB))
rd(ready(agentC))
…
```

Agent B:

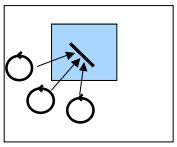
```
…
out(ready(agentB))
rd(ready(agentA))
rd(ready(agentC))
…
```

Agent C:

```
…
out(ready(agentC))
rd(ready(agentA))
rd(ready(agentB))
…
```


Sincronizzazione (4)

- Barriera a N agenti, sfruttando programmazione del tuple centre...



ANY agent:

```
…
out(ready)
rd(ready_all)
…
```

BASIC BARRIER SYNCHRONISATION in ReSpecT

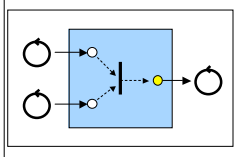
```
reaction(out(ready),( in_r(ready),
    in_r(ready_entities(N)), N1 is N+1,
    out_r(ready_entities(N1)) )).

reaction(out_r(ready_entities(N)),(
    rd_r(barrier_size(N)), out_r(ready_all))).
```

HYPOTHESIS:
- Initial space content with `ready_entities(0)`
- `barrier_size(N)` tuple to specify number of coordinables to be synchronised

Workflow

- Semplice esempio di coordinazione di tipo join
  - taskC parte solo quando ultimato il task A e il task B



Agent A:

```
…
out(task_done(taskA,info(...)))
…
```

Agent B:

```
…
out(task_done(taskB,info(...)))
…
```

Agent C:

```
…
in(task_todo(taskC,Info))
…
```

Coordinating behaviour in ReSpecT

```
reaction(out(task_done(taskA,A),(
    rd_r(task_done(taskB,B)),
    out_r(task_todo(taskC,args(A,B))) )).
reaction(out(task_done(taskB,B),(
    rd_r(task_done(taskA,A)),
    out_r(task_todo(taskC,args(A,B))) )).
```


Tecnologia TuCSoN

- TuCSoN disponibile come progetto open-source
  - tecnologia fully Java-based
  - scaricabile da http://lia.deis.unibo.it/research/tucson
- Tecnologia
  - TuCSoN Service
    - per installare TuCSoN su un nodo
  - TuCSoN API
    - per sviluppare agenti Java / Prolog che comunicano e si coordinano via TuCSoN
  - TuCSoN tools
    - per gestire tuple centre

TuCSoN Service

- Il servizio si installa lanciando l'applicazione Node (versione 1.4.0)

```
java -cp tucson.jar alice.tucson.runtime.Node
```

- Installato il servizio, l'host diventa un nodo TuCSoN, quindi in grado di ospitare tuple centres

TuCSoN Tools

- Inspector
  - permette di monitorare e controllare lo stato di un tuple centre e di cambiarne le specifiche di coordinazione
  - si lancia con (versione 1.4.0)

```
java -cp tucson.jar alice.tucson.ide.Inspector
```

- TuCSoN shell
  - shell comandi che permette ad umani di interagire con centri di tuple
    - Shell interface for human agents
  - si lancia con (versione 1.4.0)

```
java -cp tucson.jar alice.tucson.ide.CLIAgent
```


TuCSoN API

- Java
  - package alice.tucson.api
    - contiene classi / interfacce per interagire con TuCSoN
      - Tucson, TupleCentreId, TucsonContext
    - classi di ausilio come semplici template per agenti
      - Agent, Automaton
  - package alice.logictuple
    - contiene classi per il linguaggio di comunicazione basato su tuple logiche
- tuProlog
  - libreria per sviluppare programmi Prolog che usano TuCSoN

TuCSoN in Java (1)

```
import alice.tucson.api.*;
import alice.logictuple.*;

public class Test {
  public static void main(String[] args) throws Exception {
    TucsonContext cn = Tucson.enterDefaultContext();
    TupleCentreId tid = new TupleCentreId("test_tc");
    cn.out(tid, new LogicTuple("p",new Value("hello world")));
    LogicTuple t=cn.in(tid, new LogicTuple("p",new Var("X"));
    System.out.println(t);
  }
}
```


TuCSoN in Java (2)

```
import logictuple.*;
import tucson.api.*;

public class Test2 {
    public static void main (String args[]) throws Exception{

        AgentId aid=new AgentId("agent-0");
        TucsonContext cn = Tucson.enterContext(new DefaultContextDescription(aid));

        // put the tuple value(1,38.6) on the temperature tuple centre
        TupleCentreId tid=new TupleCentreId("temperature");
        LogicTuple outTuple=LogicTuple.parse("value(1,38.6)");
        cn.out(tid,outTuple);

        // retrieve the tuple using value(1,X) as a template
        LogicTuple tupleTemplate=new LogicTuple("value",
                                                new Value(1),
                                                new Var("X"));
        LogicTuple inTuple=cn.in(tid,tupleTemplate);

        cn.exit();
        System.out.println(inTuple);
    }
}
```


TuCSoN in Java: A simple agent

```
import alice.tucson.api.*;
import alice.logictuple.*;

class MyAgent extends Agent {
  protected void body(){
    try {
      TupleCentreId tid = new TupleCentreId("test_tc");
      out(tid, new LogicTuple("p",new Value("hello world")));
      LogicTuple t=in(tid, new LogicTuple("p",new Var("X"));
      System.out.println(t);
    } catch (Exception ex){}
  }
}

public class Test {
  public static void main(String[] args) throws Exception {
    AgentId aid=new AgentId("alice");
    new MyAgent(aid).spawn();
  }
}
```

TuCSoN in Prolog (tuProlog)

```
:- load_library('alice.tuprologx.lib.TucsonLibrary').
:- solve(go).

go:-
  test_tc ? out(p('hello world')),
  test_tc ? in(p(X)),
  write(X), nl.
```


CONCLUSIONI

Agenti e MAS per Ingegneria dei sistemi distribuiti

- Agenti e MAS come paradigma di progettazione e sviluppo di sistemi complessi
  - distribuiti, concorrenti,...
- Nella ricerca: modelli, linguaggi, metodologie, infrastrutture agent-based
- Modelli e infrastrutture di coordinazione: TuCSoN
- Agenti e MAS su paradigmi mainstream
  - Agenti Java/Prolog + TuCSoN

Tesi disponibili

- Gruppo di ricerca DEIS Cesena/Bologna
  - Andrea Omicini, Mirko Viroli, Enrico Denti, Alessandro Ricci, Luca Gardelli, ...
- Ricerca
  - Metodologie
    - Estensione di SODA
  - Progettazione e realizzazione linguaggi ad agenti
    - compilatori / interpreti
  - Infrastrutture di coordinazione / organizzazione
    - artefatti di coordinazione, ACC,
- Ricerca applicata
  - Design e sviluppo infrastruttura/tools per TuCSoN ver. 2.0.0
- Applicazioni
  - Sviluppo sistemi ad agenti basati su agenti Prolog e Java + TuCSoN

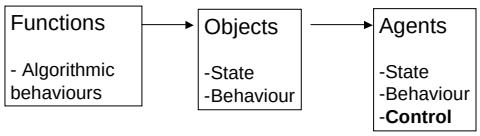
APPENDICI

CONFRONTO AGENTI vs. OGGETTI: QUADRO SINTETICO

Incapsulamento

- Cosa
  - "The result of hiding a representation and implementation" (Gamma et al)
  - "Wrapping data (structures) and related operations (behaviours) in the same abstraction"
  - Astraendo: principio generale di *località* e *modularità*
- Perché importante
  - Riuso, estendibilità, controllabilità

Incapsulamento: confronto

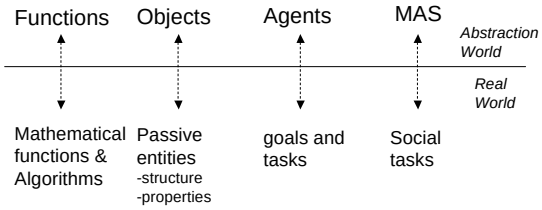


"*An object has control on its state, but not on its behaviour. An agent has control over both*"

Livello di astrazione

- Cosa
  - Modellazione efficace ed naturale del mondo
    - Superare il gap fra idee e software
  - Espressività
- Perché importante
  - Progettazione e sviluppo sistemi
    - Costo (risorse temporali, umane), "eleganza / bellezza",...
  - Fondamentale per il mantenimento ed evoluzione sistemi
    - incubo dell'ingegnere del software

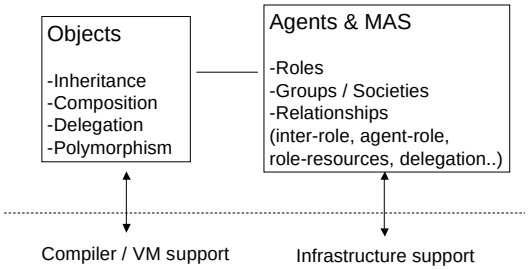
Livello di astrazione: confronto



Organizzazione: Modelli & Meccanismi

- Cosa
  - "Come scalare con la complessità strutturale del mondo"
  - Meccanismi e astrazioni first-class per dominare complessità strutturale
- Perché importante
  - Riuso, estendibilità, evoluzione

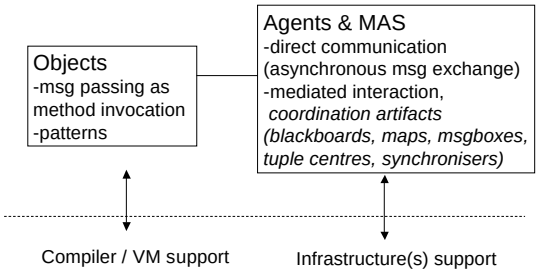
Organizzazione: confronto



Interazione e Coordinazione: Modelli & Meccanismi

- Cosa
  - "Come scalare con la complessità delle interazioni e dei processi di coordinazione del mondo"
  - Astrazioni e meccanismi first-class per rappresentare e governare le interazioni
- Perché importante
  - Comportamento corretto del sistema nel suo complesso, controllabilità, riuso

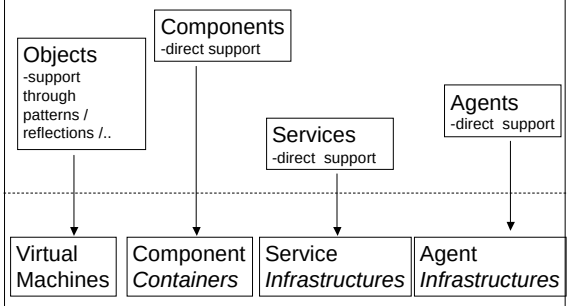
Interaction & Coordination Models & Mechanisms



Supporto per Apertura

- Cosa
  - "Come scalare con l'apertura dei sistemi e dei loro ambienti"
    - eterogeneità
    - dinamismo & non-prevedibilità (al design)
      - "*changing the engine while the car is running*"
      - "unknown at design, discovered at runtime"
  - Focus su runtime / deployment time
- Perché importante
  - Robustezza dei sistemi, riuso, estendibiulità
  - Scenario delle applicazioni future

Supporto per Apertura: confronto



BIBLIOGRAFIA

Alcuni articoli di riferimento: agenti e MAS

- Paradigma
  - N.R. Jennings. An agent-based approach for building complex software systems. *Communication of ACM*, 44(6):35–41, April 2001
  - Michael J. Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice.*The Knowledge Engineering Review*, 10(2):115–152, 1995.
- Agent oriented software engineering
  - Mike Wooldridge. Agent-based software engineering. *IEE Proceedings of SoftwareEngineering*, 144(1):26–37, 1997.
  - Franco Zambonelli, Nicholas R. Jennings, Andrea Omicini, and Michael Wooldridge. Agent-oriented software engineering for internet applications. chapter 13, pages 369–398, of *Coordination of Internet Agents: Models, Technologies, and Applications*. (Omicini et al eds) Springer-Verlag, March 2001.
  - Andrea Omicini. SODA: Societies and infrastructures in the analysis and design ofagent-based systems. In Paolo Ciancarini and Michael J. Wooldridge, editors, *Agent-Oriented Software Engineering*, volume 1957 of *LNCS*, pages 185–193. Springer-Verlag, 2001. 1st International Workshop (AOSE 2000), Limerick, Ireland, 10 June2000. Revised Papers
  - Van Dyke Parunak. 'Go To The Ant': Engineering principles from natural agent systems.*Annals of Operations Research*, 75:69–101, 1997.

Alcuni articoli di riferimento: agenti e MAS (cont.)

- Storici
  - Shoham, "Agent Oriented Programming", *Artificial Intelligence*, 60(1), p 51-92, 1993
  - Newel, "The knowledge level", *Artificial Intelligence,* 18(1), p. 87-127, 1982

Articoli di riferimento: coordinazione

- Interaction
  - Why Interaction is more powerful that algorithms (Wegner) – Communication of ACM, Vol. 40, No. 5, May 1997
  - Interactive Foundation of Computing (Wegner) – Theoretical Computer Science, Vol. 192, No. 2, February 1998
- Coordination Overview & Surveys
  - Coordination Languages and their Significance (Gelernter, Carriero) – Communication of ACM, Vol. 33, No. 2, February 1992
  - Coordination Models and Languages as Software Integrators (Ciancarini) – ACM Computing Surveys, Vol. 28, No. 2, June 1996
  - Programmable Coordination Media (Denti, Natali, Omicini) – 2nd International Conference (COORDINATION '97), Proceedings, LNCS 1282, Springer-Verlag, 1997
  - Coordination Models and Languages (Arbab, Papadopoulos) – Advances in Computers, Vol. 46, Academic Press, August 1998
  - The Interdisciplinary Study of Coordination (Malone, Crostow) – ACM Computing Surveys, Vol. 26, No. 1, March 1994

Articoli: coordinazione (cont.)

- Coordination Models/Languages/Infrastructures and TuCSoN
  - Tuple Spaces & Linda
    - Generative Communication in Linda (Gelernter) – ACM Transactions on Programming Languages and Systems, Vol. 7, No. 1, January 1985
  - Tuple Centre & ReSpecT
    - On the Expressive Power of a Language for Programming Coordination Media (Denti, Natali, Omicini), 1998 ACM Symposium on Applied Computing (SAC), 1998
    - From Tuple Spaces to Tuple Centres (Omicini, Denti) – Science of Computer Programming, No. 41, 2001
    - Formal ReSpecT (Omicini, Denti) – Electronic Notes in Theoretical Computer Science, No. 48, 2001
  - TuCSoN
    - <http://www.lia.deis.unibo.it/Staff/AndreaOmicini publication section>
    - Several other works can be found at the articles page of the web sites:
      - http://www.lia.deis.unibo.it/Staff/AndreaOmicini
      - http://www.lia.deis.unibo.it/Staff/AlessandroRicci

Libri

- Recenti
  - "Introduction to Multi-Agent Systems" (Michael Wooldridge)
    - utilizzato come testo di riferimento in varie università al mondo, inclusa la II Facoltà di Cesena (corso sistemi distribuiti intelligenti)
  - "Agent-Based Software Development" (Ashri et al.)
    - raccolta recente, organica, delle principali metodologie per l'ingegneria ad agenti (tra cui SODA)
  - "Multi-Agent Systems" (Gerard Weiss)
    - Raccolta organica (non troppo) di contributi che introducono gli aspetti fondamentali dei MAS
  - "Coordination of Internet Agents" (Omicini et al. eds) - Springer Verlag
    - Raccolta organica di contributi sui vari aspetti della coordinazione per sistemi ad agenti

Libri

- Storici
  - "Multi-Agent Systems - An Introduction to Distributed Artificial Intelligence" (Ferber) - Addison Wesley
    - libro "storico", molto completo, un po' datato, scritto da uno dei ricercatori di riferimento
  - "Software Agents" (Bradshaw ed.) - MIT Press
    - insieme di primi contributi sulla 'visione' ad agenti
  - "Readings in Distributed Artificial Intelligence" (Bond, Gasser eds) - Morgan Kaufman
    - riferimento primo per articoli scritti prima del 1998

Conferenze

- AAMAS (International Joint Conference on Autonomous Agents and Multi-Agent Systems)
  - Dal 2001 (prima edizione, a Bologna, organizzata dal DEIS) ha unito insieme precedenti conferenze su agenti (ATAL, Agents,ICMAS)
- ESAW (Engineering Societies in an Agent World)
  - Workshop internazionale, con fucus su ingegneria sistemi / società ad agenti
- AT2AI (From Agent Theory to Agent Implementation)
  - Workshop internazionale, con fucus su ingegneria sistemi in termini di agenti