ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Seconda Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

METAMODELLO A&A E UTILIZZO ARTEFATTI PER MONITORAGGIO DI MAS

Elaborata nel corso di: Sistemi Multiagente LS

Relazione di: ALBERTO CASTORI

ANNO ACCADEMICO 2009–2010

PAROLE CHIAVE

Metamodello A&A

Artefatto

Monitoraggio

Indice

Introduzione

In questa relazione si parte illustrando l'evoluzione dei linguaggi di programmazione, dalle origini fino alla programmazione ad agenti.
Si prosegue nel capitolo 2 dove si fornisce una panoramica sul meta-modello A&A, in cui è presente l'astrazione di artefatto.
Nel capitolo 3 si illustra come gli artefatti oltre a modellare l'ambiente e la coordinazione in un workspace, possano essere usati in maniera naturale anche per il monitoraggio di un sistema MAS.
Concludiamo nel capitolo 4 fornendo un elenco di tecnologie disponibili e proponendo un piccolo esempio pratico.

Capitolo 1

Evoluzione dei paradigmi: dalle origini agli agenti

In questo capitolo si passano in rassegna le varie fasi dell'evoluzione dei linguaggi di programmazione:

- programmazione monolitica;
- programmazione modulare;
- programmazione object-oriented;
- programmazione agent-oriented.

Poi ci si sofferma sul paradigma agent-oriented dando una definizione di agente.

1.1 Programmazione monolitica

All' inizio l'unità base di un sistema software era il programma stesso di cui il programmatore aveva il pieno controllo.
Lo stato era definito totalmente a design-time dal programmatore, l'invocazione avveniva solo dall'esterno per opera dell'utente.
Il fatto di avere un unico blocco monolitico andava a discapito della riusabilità, una volta sviluppato il sistema serviva solo per un specifico ambito applicativo.

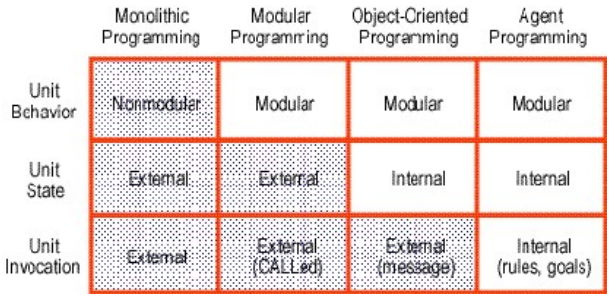

Figura 1.1: Evoluzione dei linguaggi di programmazione secondo Odell

1.2 Programmazione modulare

Col passare del tempo i programmi diventarono sempre più complessi, ma allo stesso tempo aumentò anche la disponibilità di memoria, ciò portò ad adottare delle forme di modularizzazione del codice: le funzioni o procedure. Queste sono delle porzioni di programma che possono essere riutilizzate in ambiti diversi. Così facendo si ebbe modularità e riuso, tuttavia lo stato rimaneva definito dall'esterno (tramite i parametri di input) e pure l'invocazione era esterna (chiamata a procedura).

1.3 Programmazione object-oriented

La fase successiva è quella dell'Object-Oriented che inizia a prendere piede attorno agli anni '80. L'unità base di un sistema software diventa l'oggetto istanza di una classe. Si ha modularità e riuso, inoltre qui lo stato è interno in quanto incapsulato nei campi dell'oggetto, ed accessibile attraverso i metodi dell'interfaccia. L'invocazione rimane però esterna.

1.4 Programmazione agent-orientd

L'ultima fase di questa evoluzione è l'agent-oriented. L'unità base diviene l'agente, esso rispetto ad un oggetto ha la proprietà che incap-

sula al suo interno il flusso di controllo ed il criterio governarlo; così facendo l'invocazione non è più esterna.

1.5 Ancora sugli agenti

Secondo il paradigma agent-oriented un sistema è composto da più entità, gli agenti appunto ed è per questo che si parla di sistemi multi-agente o più brevemente MAS (Multi-Agent System).
Sul concetto di agente hanno dato il loro contributo diverse discipline: Intelligenza Artificiale classica e distribuita, ricerca sui linguaggi di programmazione, Robotica e Software Engineering. Una definizione semplice e universalmente riconosciuta è la seguente: **un agente è un'entità computazionale autonoma**. Essere autonomo significa incapsulare il controllo ed il criterio per governarlo. Da questa caratteristica ne derivano altre:

- **pro-attività**, un agente non può essere invocato, per cui non presenta un'interfaccia e decide da solo quando agire sul mondo esterno;
- **situadness**, termine non facilmente traducibile in Italiano, le azioni di un agente dipendono fortemente dall'ambiente in cui è inserito;
- **essere sociale**, si è autonomi solo in relazione ad altre entità, quindi quando si è immersi in una società.

Un agente può avere altre caratteristiche reattività, intelligenza, mobilità che non sono mandatorie.
Perchè c'è bisogno del paradigma agent-oriented? Il perchè sta nelle caratteristiche dei sistemi complessi attuali. Essi sono:

- situati;
- aperti;
- formati da componenti dotati di una propria autonomia;
- sempre connessi.

Emerge che anche i sistemi artificiali presentano caratteristiche simili a quelle dei sistemi biologici, che con il tradizionale paradigma ad oggetti non si riescono a modellare nel migliore dei modi; da qui la necessità di nuove astrazioni.

Capitolo 2

Meta-modello A&A

In questo capitolo si dà una panoramica del meta-modello A&A per sistemi multiagente.

2.1 Struttura

Le entità presenti:

- agenti;
- artefatti;
- workspace.

Per quanto riguarda gli agenti valgono le considerazioni fatte nel capitolo precedente.
Si definiscono gli artefatti come: **entità computazionali passive che vengono usate dagli agenti**.
Il workspace è un contenitore logico di agenti ed artefatti.
Ogni artefatto gode delle seguenti proprietà:

- **ispezionabilità**, le caratteristiche di un artefatto devono essere visibili ad un agente;
- **controllabilità**, deve essere possibile controllare un artefatto, ovvero fermarlo o farlo ripartire quando necessario;

- **malleabilità**, possibilità di modificare a run-time il comportamento di un artefatto;
- **predicibilità**, il comportamento di un artefatto deve essere predicibile;
- **formalizzabilità**, verificare automaticamente le proprietà di un artefatto;
- **linkabilità**, possibilità di comporre più artefatti tra loro;
- **distribuibilità**, avere artefatti su più nodi della rete.

Emergono differenze sostanziali tra agenti ed artefatti:

- gli agenti sono autonomi, gli artefatti no;
- gli agenti sono pro-attivi, gli artefatti no;
- gli agenti sono opachi, mentre gli artefatti trasparenti;
- gli agenti sono impredicibili, gli artefatti predicibili.

2.2 Interazione

Le interazioni ammissibili all'interno di un MAS sono:

- **comunicazione**, un agente comunica con un altro agente;
- **operazione**, un agente usa un artefatto;
- **composizione**, più artefatti si linkano tra loro;
- **presentazione**, un artefatto si manifesta ad un agente.

Soffermiamoci sulla operazione, ovvero sull'uso di un artefatto da parte di un agente. In letteratura si individuano tre livelli di uso:

- **co-construction**, gli ingegneri possono mettere all'interno di un artefatto dell'intelligenza in grado di condizionare l'evolvere di un MAS;

- **co-operation**, un agente può modificare un artefatto in base alle proprie esigenze;
- **co-ordination**, un agente usa un artefatto così come è.

2.3 Metafora e basi storiche

Per comprendere in maniera più intuitiva il metamodello si può pensare alla seguente metafora: un ambiente di lavoro umano. L'ambiente stesso è il workspace, gli agenti sono i lavoratori umani e gli artefatti i loro strumenti.
L'idea di introdurre anche l'astrazione di artefatto viene dai contributi di altri discipline non correlate alla Computer Science. Tra le più importanti citiamo: Activity Teory,Distributed Cognition e CSCW.
La prima è una scienza sovietica il cui oggetto di studio è l'evolvere della società in seguito alle attività umane. Essa afferma che ciascuna attività umana è mediata da artefatti di natura sia fisica che cognitiva. Questi hanno una funzione sia abilitativa che vincolante, possono permettere delle azioni e vincolare delle altre.
La Distributed Cognition è un ramo delle Cognitive Sciences, scienze che studiano i processi cognitivi umani; affermazione di interesse per noi è che la conoscenza non è confinata nella mente umana, ma è distribuita nell'ambiente esterno all'interno degli oggetti. Effettuando una trasposizione al mondo ad agenti, emerge che la conoscenza che un agente ha del mondo non è confinata al suo interno, ma sarà anche negli artefatti. Essendo questi fonti di conoscenza, possono influenzare il comportamento e l'evoluzione del MAS.
CSCW (Computer Supported Cooperative Work) è una disciplina il cui scopo è l'automatizzazione dell'attività umana, per raggiungerla sono necessari due elementi:

- context awareness, le entità da coordinare devono essere consapevoli delle presenza delle altre;
- artefatti coordinativi, devono consentire solo le azioni utili ai fini della coordinazione impedendo le altre.

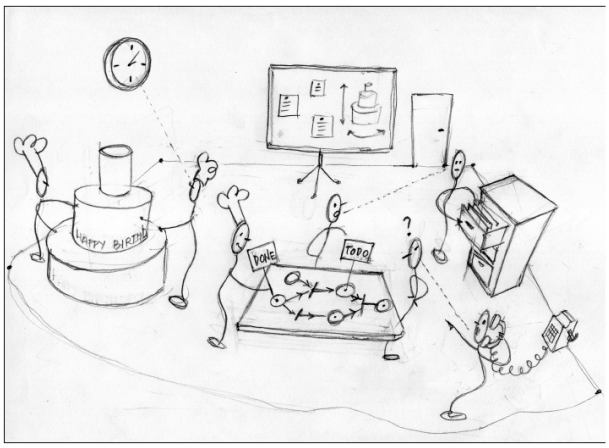

Figura 2.1: Metafora del metamodello A&A col mondo umano

Capitolo 3

Artefatti nel monitoraggio di un MAS

In questo capitolo, dopo aver definito cosa si intende per monitoraggio di un MAS, ci si sofferma sul ruolo degli artefatti in ciò.

3.1 Monitoraggio di un MAS e problematiche

Il monitorare un sistema in generale comprende sia la capacità di osservare la sua evoluzione nel tempo sia quella di intervenire sul comportamento qualora necessario, per esempio poter fermare o far ripartire tutto il sistema o una sua parte. Si è detto che un sistema MAS che segua il metamodello A&A è formato da agenti ed artefatti. Nel monitoraggio intervenire sugli agenti è difficoltoso in quanto:

- **sono opachi**, non rendono visibile il loro stato interno all'esterno, pertanto un'osservazione dell'evoluzione temporale risulta difficoltosa;

- **sono autonomi**, in quanto tali non possono essere invocati (come invece è possibile fare con gli oggetti), per cui problematico è intervenire sul loro comportamento: per esempio se diciamo ad un agente di fermarsi, questo è libero di farlo o meno.

Ad un primo sguardo sembrerebbe arduo monitorare un MAS, tuttavia ciò è possibile utilizzando gli artefatti. Riprendendo quanto detto al capitolo 2, ogni artefatto gode delle proprietà di ispezionabilità e malleabilità, che è ciò che ci serve. Quindi gli artefatti possono essere monitorati direttamente, inoltre se opportunatamente concepiti possono tener traccia dell'evoluzione di un MAS ed essere utilizzati per condizionarne il comportamento.

3.2 Gli artefatti nel monitoraggio

Per spiegare in maniera più esplicita come gli artefatti possano influenzare l'evoluzione di un MAS, riprendiamo la metafora dell'ambiente di lavoro umano. La presenza degli strumenti modella l'ambiente in cui le persone lavorano e condiziona anche la loro attività: tool più evoluti suggeriranno un certo modo di lavorare, altri più rudimentali un modo diverso.
Possiamo trasporre questa analogia all'interno di un workspace dove troviamo agenti ed artefatti.
Condizionare il comportamento di un MAS, significa anche poter governare lo spazio dell'interazione in modo da avere:

- coordinazione, gestione dell'interazione a tempo di esecuzione;
- organizzazione, gestione dell'interazione a design time;
- sicurezza, duale alle precedenti, si definisce cosa non deve verificarsi.

In base quindi al ruolo che un artefatto ha nell'interazione, possiamo distinguere in:

- **artefatto individuale**, ha interazione con un solo agente, gestisce aspetti di sicurezza e organizzazione;
- **artefatto sociale**, gestisce l'interazione con più agenti, si occupa della parte di coordinazione;
- **artefatto ambientale**, gestisce interazione tra MAS e ambiente esterno.

Come debba essere un artefatto e cosa deve racchiudere al suo interno viene deciso dal progettista al livello di uso co-construction.
Quando vogliamo osservare lo stato del nostro MAS in un dato istante, basterà andare a vedere le proprietà osservabili dell'opportuno artefatto in modo da avere una fotografia del nostro sistema. Quando invece si vuole andare a condizionare il comportamento di un agente o più agenti, si modificherà, grazie alla proprietà di malleabilità, un artefatto individuale o sociale.

3.3 Un esempio di artefatto per il monitoraggio: lo spazio di tuple

Il concetto di spazio di tuple si rifà al metamodello di coordinazione tuple space, proposto per la prima volta da David Gelernter nel 1985. Elementi base di questo meta-modello, oltre alle entità da coordinare:

- coordination media, lo spazio di tuple, ovvero un multiset di strutture dati, ossia le tuple;
- communication language, tuple, collezioni ordinate di informazione;
- coordination language, primitive per inserire, leggere o rimuovere tuple dallo spazio (LINDA).

Possiamo utilizzare tutto questo all'interno del A&A per gestire la coordinazione ed effettuare il monitoraggio. Le entità da coordinare sono gli agenti, mentre lo spazio di tuple lo modelliamo come un artefatto. E' possibile osservare lo stato del sistema osservando quello dell'artefatto tuple space. Tale stato è determinato dalle tuple in esso presenti.
Per poter condizionare il comportamento degli agenti, bisogna poter cambiare lo stato dello spazio di tuple. Ciò è possibile se si passa dallo spazio di tuple al centro di tuple. Un centro di tuple ha in più che ad ogni evento è possibile associare una specifica di comportamento (attività computazionale). Attraverso questa è possibile accedere o modificare lo stato del tuple centre e di conseguenza condizionare il comportamento di un agente.

Capitolo 4

Tecnologie ed esempio pratico

In questo capitolo si fornisce un elenco di alcune tecnologie disponibili, infine un piccolo esempio pratico su quanto detto.

4.1 Tecnologie

Delle tante disponibili ne citiamo alcune.
Per programmare ad agenti:

- **Jason**;
- **simpA**.

Jason è un'interprete per una versione estesa di AgentSpeak, linguaggio di alto livello per programmare ad agenti.
simpA è un framework che estende Java fornendo un layer agent-oriented.
Per implementare istanze di A&A:

- **Cartago**, framework che estende Java fornendo il supporto completo per poter implementare A&A;

Per tuple centre:

- **Tucson**;

- **Respect**.

Tucson è l'implementazione di un centro di tuple, mentre Respect è un linguaggio per la configurazione di centri di tuple.
Jason è disponibile al link *http://jason.sourceforge.net/JasonWebSite/Jason Home.php*. Tutti gli altri al sito *http://apice.unibo.it/xwiki/bin/view/Main/*.

4.2 Esempio pratico

Per illustrare cosa significa mettere in pratica quanto fin qui detto, concludiamo con un piccolo esempio pratico. In esso abbiamo un workspace ('MySimpleMAS') con all'interno un agente da monitorare ed un artefatto da usare per il monitoraggio.
Riportiamo il codice che definisce l'agente:

```
package mymas;

import alice.cartago.*;
import alice.simpa.*;

//Agent that we have to control
public class SimpleAgent extends Agent{

  @ACTIVITY void main() throws SIMPAException {
    try{
      SensorId sid=linkDefaultSensor();
      ArtifactId id=lookupArtifact("MonitorArtifact1"');
      focus(id,sid);
      String state="'running";
      while(true){
        try{
          Perception p=sense(sid,"update",100);
          String event=(String)p.getContent(0);
          if(event.equals("running")){
            state="running";
          }else{
```

```
            state="sleeping";
          }
        }catch(NoPerceptionException ex){

        }catch(InterruptedException ex){
          ex.printStackTrace();
        }finally{
          System.out.println(state);
        }
      }
    }catch(CartagoException ex){
      ex.printStackTrace();
    }
  }

}
```

Il comportamento dell'agente è molto semplice, tramite l'utilizzo del sensore di default (linkDefaultSensor), ascolta gli eventi di tipo *update* provenienti dall'artefatto MonitorArtifact1; se l'argomento è la stringa *running* stamperà questa a video durante la sua esistenza, altrimenti *sleeping*.
Il codice che definisce l'artefatto è il seguente:

```
package mymas;

import alice.cartago.*;

//Artifact for monitoring our MAS
public class MonitorArtifact extends Artifact{

  @OPERATION void init(){
    defineObsProperty("state","running");
    new MonitorArtifactGUI(this);
  }
```

```
  public void changeState(){
    String state = getObsProperty("state").stringValue();
    if(state.equals("running")){
      updateObsProperty("state","sleeping");
      signal("update","sleeping");//Emit an event
    }else{
      updateObsProperty("state","running");
      signal("update","running");
    }
  }

  public String getState(){
    String state=getObsProperty("state").stringValue();
    return state;
  }

}
```

Tramite GUI (MonitorArtifactGUI) è possibile visualizzare lo stato del MAS (agente che stampa in output sleeping o running) e tramite il bottone *Change Agent State* cambiarlo. Dello stato si tiene traccia all'interno della proprietà osservabile state, l'allineamento effettivo tra questa e lo stato dell'agente viene garantito all'avvio del sistema. Il cambiamento del comportamento dell'agente sotto monitoraggio avviene così: in seguito alla pressione del tasto, l'artefatto lancia un evento *update*, l'agente (che lo sta osservando) riceve l'evento, ne legge l'argomento ed in base al valore di questo si metterà a stampare a video *sleeping* o *running*.
Benchè banale, l'esempio illustra chiaramente il ruolo che gli artefatti hanno nel monitorare un sistema MAS, sia in termini di tener traccia (visualizzare sleeping o running), sia di condizionare l'evoluzione temporale.

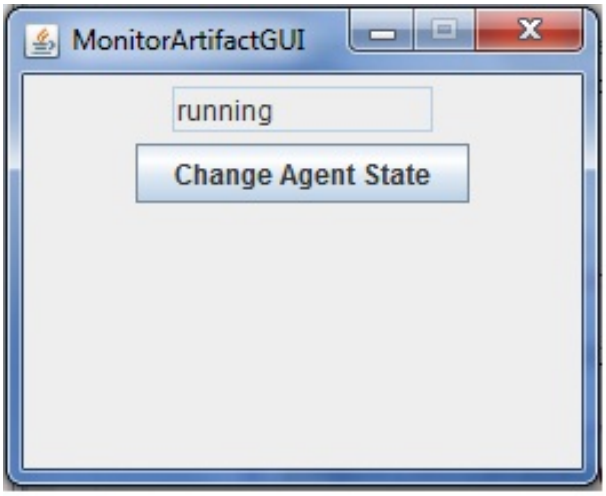

Figura 4.1: GUI

Bibliografia

- James Odell, *Objects and Agents Compared*, Journal of Object Technology, Vol 1, Number 1, May, 2002.
- Omicini A., Ricci A., and Viroli M., *Artifacts in the A&A meta-model for multi-agent systems*,2008.
- Ricci A., Viroli M., Piancastelli G., *Agents and Artifacts for Prototyping Concurrent Applications On Top of Java*,DEIS, Alma Mater Studiorum, Università di Bologna.
- Gelernter D., *Generative communication in Linda.*(1985).
- Ricci A., *simpA Annotated Manual*, aliCE team, DEIS, Università di Bologna, Italy.
- Ricci A., *CARTAGO Annotated Manual*, aliCE team, DEIS, Università di Bologna, Italy.