Relazione di "Sistemi complessi: modelli e simulazione"

Simon Vocella
Matricola: 718289

21 luglio 2012

1 Il problema

Al giorno d'oggi esistono molti sistemi di catalogazione dei paper scientifici (ex. DBLP), il nostro programma si propone di fare scraping in due o più di questi sistemi, con ovviamente dello scraping ad-hoc dipendente da come è strutturato il sito, e si propone di risolvere due problemi:

- Calcolare un indice di similarità tra i vari siti
- Filtrare le citazioni in Google Scholar tramite DBLP

2 Sorgenti informative utilizzate

In questo progetto di è deciso di utilizzare i seguenti sistemi di catalogazione:

- DBLP (http://www.informatik.uni-trier.de/∼ley/db/)
- Google scholar (http://scholar.google.it/)

3 Approccio e metriche adottate

3.1 Indice di similarità

Esiste un sistema centrale che comunica ai vari agenti cosa bisogna cercare. Ogni agente è specializzato in un sistema di catalogazione diverso. Nel nostro caso avremo due agenti. Ogni agente farà scraping nel proprio sito e ritornerà i risultati al sistema centrale. Man mano che ogni agente consegna i propri risultati, viene creata una lista $< Key, Value >$ in cui la Key sarà il titolo del paper j-esimo e $Value$ sarà uguale a $\sum_{i=0}^{n} i(j)/(n * m)$ dove $i(j)$ sarà uguale a 1 nel caso l'i-esimo agente ha il j-esimo paper, altrimenti 0 e m e n sono rispettivamente il numero dei paper trovati in totale e il numero degli agenti. L'indice di similarità sarà la sommatoria dei vari $Value$, $\sum_{j=0}^{m} Value(j) = \sum_{j=0}^{m} \sum_{i=0}^{n} i(j)/(n * m)$ e in questo caso ci indica quanto le n liste raccolte dagli n agenti siano simili.

Nel caso in cui $\sum_{i=0}^{n} i(j)/(n * m) = 1/m$ allora qualsiasi agente ha trovato il paper j-esimo.

Nel caso in cui l'indice di similarità sia $\sum_{j=0}^{m} Value(j) = \sum_{j=0}^{m} \sum_{i=0}^{n} i(j)/(n*m) = 1$, ci indica che le n liste di paper raccolte dagli n agenti sono identiche.

3.2 Validazione dei paper da Google Scholar

È notorio che Google Scholar indicizza moltissime voci anche non riguardanti il mondo scientifico, quindi si è pensato di filtare le citazioni in Google Scholar tramite un sistema più rigoroso, DBLP, avendo così da avere una sorta di validazione. Per ogni paper scaricato nel calcolare l'indice di similarità, scarichiamo k citazioni, dove $k >= 0$, e ogni citazione la cerchiamo in DBLP e nel caso viene trovata la validiamo.

4 Architettura del sistema

Il sistema è basato su quattro livelli diversi. Inizialmente abbiamo un piano di controllo basato su Jason (Java-based AgentSpeak interpreter used with SACI for multi-agent distribution over the net) che chiameremo Librarian. Librarian si prende l'onere di chiedere all'utente cosa vuole cercare, tramite una GUI in Swing e poi comunica il termine da ricercare ai due agenti Dblp e Scholar. Il nome è abbastanza esplicativo su quale sito di catalogazione cercano.

Listing 1: Il file MAS: Multiple Agent System

```
// creates an MAS called bibliographyJason

MAS bibliographyJason {

    infrastructure: Centralised

    agents: scholar;
            dblp;
            librarian agentArchClass LibrarianGUI;
}
```

Il file mas2j ci permette di specificare l'architettura del nostro sistema, il tipo di infrastruttura e gli agenti che interagiscono e di che tipo sono.

4.1 Infracstructure

Un'infrastruttura fornisce i seguenti servizi per i MAS:

- comunicazione (ad esempio, le infrastrutture centralizzate implementano la comunicazione basata su KQML, mente JADE implementa la comunicazione utilizzando FIPA-ACL)
- il controllo del ciclo di vita dell'agente (creazione, l'esecuzione, distruzione)

Sono disponibili le seguenti infrastrutture:

- centralizzata:
  Questa infrastruttura esegue tutti gli agenti nello stesso host. Esso fornisce prestazioni di avvio veloce e alta per i sistemi che possono essere eseguite in un singolo computer. È anche utile per testare e sviluppare (prototipi) sistemi. È l'infrastruttura di default.

- Jade:
  Fornisce la distribuzione e la comunicazione con Jade, che si basa su FIPA-ACL. Con questa infrastruttura, tutti gli strumenti disponibili con JADE sono disponibili anche per monitorare e controllare gli agenti Jason.

4.2 Librarian

Listing 2: L'agente Librarian

```
// this agent starts the scraping!
// ST <- Search Term
// R <- Result
// S <- Source
// C <- Count
// T <- Total
// CI <- Citations

/* plans */

+!start_search(ST) : true
    <- .print("start_search");
       -+count(0, 2);
       .broadcast(tell, search_term(ST)).

+send_information_dblp(ST, R)[source(S)] : true
   <-    .print("send information from DBLP");
                get_index(S, ST, R, _);
                +finish_first_search(S).

+send_information_scholar(ST, R, CI)[source(S)]: true
   <-    .print("send information from SCHOLAR");
                get_index(S, ST, R, CI);
                +finish_first_search(S);
                .print("search_citation in DBLP ", CI);
                .send(dblp, tell, search_citation(CI));
                .abolish(send_information(_,_,_)).

+finish_first_search(S): count(C, T) & C < T-1
   <-    .print("finish_first_search ", S);
                -count(C, T);
                +count(C+1, T).

+finish_first_search(S): count(C, T) & C = T-1
   <-    .print("finish_first_search ", S);
                show_index(S);
                .abolish(send_information(_,_,_)).

+send_filtered_citations(R)[source(S)]: true
   <-    .print("send filtered information from DBLP: ", R).
```

Listing 3: Architettura LibrarianGUI

```
import jason.architecture.AgArch;
import jason.asSemantics.ActionExec;
import jason.asSyntax.ASSyntax;
import jason.asSyntax.Literal;
import jason.asSyntax.ObjectTerm;
import jason.asSyntax.VarTerm;

import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map.Entry;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;

/** example of agent architecture's functions overriding */
public class LibrarianGUI extends AgArch {

 JTextArea jt;
 JTextField jf;
 JFrame f;
 JButton search;
 HashMap<String, Object> results;
 HashMap<String, Double> papers;
 int numberAgents;

 public LibrarianGUI() {
  jt = new JTextArea(10, 30);
  jf = new JTextField(10);

  results = new HashMap<String, Object>();
  papers = new HashMap<String, Double>();
  numberAgents = 0;

  jf.setText("Giuseppe Vizzari");

  search = new JButton("Start a new search");
  search.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
    String keyword = jf.getText();
    jt.append("Search this keyword: "+keyword+"\n");
    Literal goal = ASSyntax.createLiteral("start_search",
      ASSyntax.createString(keyword));
    getTS().getC().addAchvGoal(goal, null);
    search.setEnabled(false);
```

```
    }
   });

   f = new JFrame("Librarian agent");
   f.getContentPane().setLayout(new BorderLayout());
   f.getContentPane().add(BorderLayout.NORTH, new JScrollPane(jt));
   f.getContentPane().add(BorderLayout.CENTER, new JScrollPane(jf));
   f.getContentPane().add(BorderLayout.SOUTH, search);
   f.pack();
   f.setVisible(true);
  }

  @Override
  public void act(ActionExec action, List<ActionExec> feedback) {
   if (action.getActionTerm().getFunctor().startsWith("show_index")) {
    jt.append("show index\n");

    for(Entry<String, Object> entry : results.entrySet()) {
     jt.append("-----------------------------------\n");
     jt.append("agent: "+entry.getKey()+"\n");
     jt.append("result: "+entry.getValue()+"\n");
     jt.append("-----------------------------------\n");
    }

    int m = papers.size();
    int n = numberAgents;
    double k;
    double sim_index = 0.0;

    for(Entry<String, Double> entry : papers.entrySet()) {
     k = entry.getValue().doubleValue();
     jt.append("paper: "+entry.getKey()+", value: "+(k/(m*n))+"\n");
     sim_index += (k/(m*n));
    }

    jt.append("Similarity Index: "+sim_index);

    action.setResult(true);
    feedback.add(action);

    search.setEnabled(true); // enable GUI button
   } else if (action.getActionTerm().getFunctor().startsWith("get_index")) {
    String agent = ((VarTerm) action.getActionTerm().getTerm(0)).toStringAsTerm();
    Object result = ((ObjectTerm) action.getActionTerm().getTerm(2)).getObject();

    if(result instanceof ArrayList<?>) {
     ArrayList<String> papersFromAgent = (ArrayList<String>) result;

     jt.append("get index for "+agent+"\n");

     numberAgents += 1;
```

```
    results.put(agent, result);

    for(String paper : papersFromAgent) {
     paper = paper.trim();
     Double value = papers.get(paper);

     if(value == null) {
   value = 0.0;
     }

     papers.put(paper, value+1.0);
    }

    action.setResult(true);
   } else {
    action.setFailureReason(Literal.LFalse, "result is not ArrayList");
    action.setResult(true);
   }

   feedback.add(action);
  } else {
   // send the action to the environment to be performed.
   super.act(action,feedback);
  }
 }

 @Override
 public void stop() {
  f.dispose();
  super.stop();
 }
}
```

L'agente Librarian è definito nel file asl e grazie al fatto che nel file mas2j sia stata definita la proprietà *agentArchclassLibrarianGUI* definiamo che l'architettura dell'agente è definita nel file LibrarianGUI.java.
Tramite GUI, l'utente può ricercare un qualsiasi autore dentro i siti di catalogazione, questo lancerà all'interno dell'agente Librarian una chiamata *search_term* ad ogni agente.

4.3 DBLP e Scholar

All'azione di *search_term* ogni agente comunica un azione interna di $< Agent >$.*scraping_search_term* che comunica il termine ad un azione interna.

Listing 4: L'agente Scholar

```
// this agent search on Google Scholar

+search_term(ST)[source(S)] : true
   <-    .print("search_term SCHOLAR");
                scholar.scraping_search_term(ST, R, CI);
                .send(S, tell, send_information_scholar(ST, R, CI)).
```

Listing 5: Azione interna di scraping di Scholar

```
package scholar;

import jason.asSemantics.DefaultInternalAction;
import jason.asSemantics.TransitionSystem;
import jason.asSemantics.Unifier;
import jason.asSyntax.ObjectTermImpl;
import jason.asSyntax.StringTerm;
import jason.asSyntax.Term;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map.Entry;

import scraping.Node;
import scraping.Session;


public class scraping_search_term extends DefaultInternalAction {

 private static final long serialVersionUID = 1L;
 private static final boolean DEBUG = false;

 @Override
 public Object execute(TransitionSystem ts, Unifier un, Term[] args)
                 throws Exception {
  System.out.println("Search information about "+args[1]+" with "+args[0]);

  String search_term = ((StringTerm) args[0]).getString();

  ArrayList<String> papers = new ArrayList<String>();
  HashMap<String, ArrayList<String>> citations =
                  new HashMap<String, ArrayList<String>>();

  if(DEBUG) {
   papers.add("prova1");
   papers.add("prova2");

   ArrayList<String> citationsLocal = new ArrayList<String>();
   citationsLocal.add("citazione1");
   citationsLocal.add("citazione2");
   citationsLocal.add("citazione3");

   citations.put("prova1", citationsLocal);
  } else {
   String base_url = "http://scholar.google.it";
   Session s = new Session("50001");

   try {
    s.setErrorTolerance(true);
    s.visit(base_url);
    ArrayList<Node> nodes = s.xpath("//*[@name='q']");
```

```
Node n = nodes.get(0);
n.set(search_term);
n.getForm().submit();
HashMap<String, String> citiedBies = new HashMap<String, String>();

int npage = 1;
boolean new_page = true;

while(new_page) {
 new_page = false;
 ArrayList<Node> divs = s.xpath("//*[@class=\"gs_r\"]");

    for(int i=1; i<divs.size(); i++) {
     Node div = divs.get(i);
     ArrayList<Node> page_papers = div.xpath("./*[@class=\"gs_rt\"]/a");

     if(page_papers != null) {
         for(Node page_paper : page_papers) {
          String paperTitle =
            java.net.URLDecoder.decode(page_paper.text().trim(), "UTF-8");
          papers.add(paperTitle);

          ArrayList<Node> citedbiesPaper =
           page_paper.xpath("ancestor::div/*[@class=\"gs_fl\"]/a");

          if(citedbiesPaper != null) {
           String citedbiesPaperUrl = citedbiesPaper.get(0).get("href");

           if(citedbiesPaperUrl.contains("cites=")) {
            citiedBies.put(paperTitle, citedbiesPaperUrl);
           }
          }
         }
     }
    }

    ArrayList<Node> pages =  s.xpath("//*[@id=\"gs_n\"]/center/table/" +
                    "tbody/tr/td/a[text()=\""+(npage+1)+"\"]");

    if(pages != null && pages.size() > 0) {
        String next_page = pages.get(0).get("href");
        npage += 1;
        new_page = true;
        System.out.println("next_page: "+next_page);
        s.visit(base_url+next_page);
    }
}

for(Entry<String, String> citiedBy : citiedBies.entrySet()) {
 s.visit(base_url+citiedBy.getValue());
 ArrayList<String> papersThatCite = new ArrayList<String>();
```

```
      npage = 1;
      new_page = true;

      while(new_page) {
       new_page = false;
       ArrayList<Node> divs = s.xpath("//*[@class=\"gs_r\"]");

          for(int i=1; i<divs.size(); i++) {
           Node div = divs.get(i);
           ArrayList<Node> page_papers = div.xpath("./*[@class=\"gs_rt\"]/a");

           if(page_papers != null) {
               for(Node page_paper : page_papers) {
                String paperTitle =
                 java.net.URLDecoder.decode(page_paper.text().trim(), "UTF-8");
                papersThatCite.add(paperTitle);
               }
           }
          }

          ArrayList<Node> pages = s.xpath("//*[@id=\"gs_n\"]/center/table/" +
                        "tbody/tr/td/a[text()=\""+(npage+1)+"\"]");

          if(pages != null && pages.size() > 0) {
              String next_page = pages.get(0).get("href");
              npage += 1;
              new_page = true;
              System.out.println("next_page: "+next_page);
              s.visit(base_url+next_page);
          }
       }

       citations.put(citiedBy.getKey(), papersThatCite);
      }

      System.out.println("papers: "+papers);
     } catch (Exception e) {
      e.printStackTrace();
     }
    }

          return un.unifies(args[1], new ObjectTermImpl(papers))
                        && un.unifies(args[2], new ObjectTermImpl(citations));
      }
}
```

Listing 6: L'agente Dblp

```
// this agent search on DBLP

+search_term(ST)[source(S)] : true
   <-    .print("search_term DBLP");
                dblp.scraping_search_term(ST, R);
```

```
                .send(S, tell, send_information_dblp(ST, R)).

+search_citation(CI)[source(S)] : true
    <-  .print("search citation DBLP");
        dblp.scraping_search_citation(CI, R);
        .print("result: ", R);
        .send(S, tell, send_filtered_citations(R)).
```

Listing 7: Azione interna di scraping Dblp

```
package dblp;

import jason.asSemantics.DefaultInternalAction;
import jason.asSemantics.TransitionSystem;
import jason.asSemantics.Unifier;
import jason.asSyntax.ObjectTermImpl;
import jason.asSyntax.StringTerm;
import jason.asSyntax.Term;

import java.util.ArrayList;

import scraping.Node;
import scraping.Session;

public class scraping_search_term extends DefaultInternalAction {

 private static final long serialVersionUID = 1L;
 private static final boolean DEBUG = true;

 @Override
 public Object execute(TransitionSystem ts, Unifier un, Term[] args)
     throws Exception {
  System.out.println("Search information about "+args[1]+" with "+args[0]);

  String search_term = ((StringTerm) args[0]).getString();

  ArrayList<String> papers = new ArrayList<String>();

  if(DEBUG) {
   papers.add("prova1");
   papers.add("prova3");
  } else {
   Session s = new Session("50002");
   String base_url = "http://www.informatik.uni-trier.de/" +
                "~ley/db/indices/a-tree/index.html";

   try {
    s.setErrorTolerance(true);
    s.visit(base_url);
    ArrayList<Node> nodes = s.xpath("//*[@name=\"author\"]");
    Node n = nodes.get(0);
    n.set(search_term);
    n.getForm().submit();
```

```
    ArrayList<Node> trs = s.xpath("//p[1]/table/tbody/tr");

    for(Node tr : trs) {
     ArrayList<Node> tds = tr.xpath("./td");

     if(tds != null) {
      String all_title = tds.get(2).text();
      String title = all_title.substring(all_title.indexOf(":")+1);
      title = title.substring(0, title.indexOf("."));
      papers.add(java.net.URLDecoder.decode(title.trim(), "UTF-8"));
     }
    }

    System.out.println("papers: "+papers);
   } catch (Exception e) {
    e.printStackTrace();
   }
  }

  return un.unifies(args[1], new ObjectTermImpl(papers));
 }
}
```

Come vediamo ogni azione interna comunica ad un interfaccia ad hoc di scraping chiamata Session. Session si preoccupa, una volta istanziata, di far partire un webkit modificato che rimane in ascolto su una porta specificata o di default che chiameremo per semplicità Webkit. Tramite Session comunichiamo le nostre azioni di visita al Webkit e tramite xpath manipoliamo il DOM della pagina e tramite funzioni di QT o Javascript manipoliamo funzioni come il submit, click o history, etc.
Ogni agente esegue il proprio scraping e manipola le pagine che visita per raccogliere informazioni (nel nostro caso i titoli dei paper), lo fa su una sola pagina nel caso di DBLP o su più pagine nel caso dii Google Scholar.
Una volta ricevuti i risultati, l'azione interna restituisce o meglio unifica (visto che parliamo di linguaggio funzionale) il risultato come una stringa e poi viene restituito all'oggetto Librarian.
Librarian si occuperà di raccogliere i paper in un HashMap e di calcolare incrementalmente l'indice di similarità.
Una volta che tutti gli agenti hanno restituito i loro risultati, Librarian calcola completamente l'indice di similarità.

5 Descrizione dei risultati

DOMANDA: devo fare qualche screenshot?

Risultati dell'agente Dblp:

- An analysis of different types and effects of asynchronicity in cellular automata update schemes
- Towards an agent-based proxemic model for pedestrian and group dynamics: motivations and first experiments

- An Agent Model of Pedestrian and Group Dynamics: Experiments on Group Cohesion
- An Agent-Based Proxemic Model for Pedestrian and Group Dynamics: Motivations and First Experiments
- A Cellular Automata Based Model for Pedestrian and Group Dynamics: Motivations and First Experiments
- ... etc.

Risultati dell'agente Scholar:

- Modeling dynamic environments in multi-agent simulation
- Situated cellular agents: A model to simulate crowding dynamics
- Situated cellular agents approach to crowd modeling and simulation
- Awareness in collaborative ubiquitous environments: The multilayered multi-agent situated system approach
- Toward a platform for multi-layered multi-agent situated system (MMASS)-based simulations: focusing on field diffusion
- ... etc.

Hashmap risultante:

- key: Web Intelligence and Intelligent Agent Technology. Proceedings of the 2009 IEEE/WIC/ACM International Conference on Web Intelligence, Worshops, value: 0.0017857142857142857
- key: Visualization of Discrete Crowd Dynamics in a 3D Environment, value: 0.0017857142857142857
- key: PASSIONE STORICA E STORIA CIVICA NELLA CALABRIA NORDOCCIDENTALE. RASSEGNA BIBLIOGRAFICA E RIFLESSIONI STORIOGRAFICHE, value: 0.0017857142857142857
- key: Le memorie del vecchio maresciallo, value: 0.0017857142857142857
- key: Bio-ICT Convergence: Filling the Gap Between Computer Science and Biology, value: 0.0017857142857142857
- ... etc.

Indice di similarità risultante: 0.5464285714285698.

6 Conclusioni

Come abbiamo visto le due liste di risultati differiscono per un buon numero di paper, in questo caso è colpa di Scholar che mette molti altri risultati oltre agli articoli scientifici (un es. è PASSIONE STORICA E STORIA CIVICA NELLA CALABRIA NORDOCCIDENTALE. RASSEGNA BIBLIOGRAFICA E RIFLESSIONI STORIOGRAFICHE, value: 0.0017857142857142857, che non mi sembra molto ad avere a che fare con l'informatica). Si potrebbe migliorare il risultato, aumentando il numero dei siti di catalogazione da visitare, così riducendo il peso di Scholar da 1/2 a 1/n dove n è il numero dei siti scelti.