Find me an internship already

(Documentation)

Student: Vasile Suciu

Group: 30434

Other team members: Novacean Ligia Nicoleta, Samarghitan Flaviu, Supuran Marius

Problem statement

Design and build an application to aid students in finding an internship based on their preferences (technology, duration, experience, location).

The application has been developed in Python and deployed on Google Cloud. It uses pymysql framework to connect to a data base and rpyc to help modules running on different machines to communicate with one another. It has a client-server architecture and it is based on 3 different prism case studies:

Crowds Protocol <http://www.prismmodelchecker.org/casestudies/crowds.php>

IPv4 Zeroconf Protocol <http://www.prismmodelchecker.org/casestudies/zeroconf.php>

Stable Matchings <http://www.prismmodelchecker.org/casestudies/stable_matching.php>

Out of the three I am responsible for implementing the stable matching. Each student looking for an internship has access to a client application. The client will then connect to a server (one for each client, hence stable matching) and request an internship based on several criteria. A server can accept connections from only one client (the first who performs a request). If another client tries to connect, it is disconnected and has to look for a different server.

Specification

The application presented is meant to help students in finding the right internship according to their preferences. It uses stable matching to pair each client (student) to a server. The server performs the actual search for the internship and returns the result. To ensure anonymity the clients can also communicate between them using crowds protocol. Thus they may issue requests for themselves or other clients in the network. In this manner the servers are unaware who is the original author of the request, thus preserving the privacy of the students. The IPv4 zeroconf protocol is normally used to assign IP’s in a network such that no conflicts appear.

Stable Matching Case Study (Gale and Shapley approach)

**dtmc** *// model is a DTMC/Markov chain*

*//------------------------------------------------------*

*// PREFERENCE LISTS*

*// man i prefers woman j over woman k if mij>mik*

*// woman i prefers man j over man k if wij>wik*

*// preference list for men*

**const** **int** m11=3;

**const** **int** m12=2;

**const** **int** m13=1;

**const** **int** m22=3;

**const** **int** m23=2;

**const** **int** m21=1;

**const** **int** m33=3;

**const** **int** m31=2;

**const** **int** m32=1;

*// preference list for women*

**const** **int** w12=3;

**const** **int** w13=2;

**const** **int** w11=1;

**const** **int** w23=3;

**const** **int** w21=2;

**const** **int** w22=1;

**const** **int** w31=3;

**const** **int** w32=2;

**const** **int** w33=1;

*//------------------------------------------------------*

*// constants used in renaming*

**const** **int** N1=1;

**const** **int** N2=2;

**const** **int** N3=3;

*//------------------------------------------------------*

*// module for first man*

**module** man1

*// current matching (0 means no matching)*

m1 : [0..3];

*// wants to change matching*

[e11] m1=0 | (m1=1 & m11>m11)|(m1=2 & m11>m12)|(m1=3 & m11>m13) -> (m1'=1);

[e12] m1=0 | (m1=1 & m12>m11)|(m1=2 & m12>m12)|(m1=3 & m12>m13) -> (m1'=2);

[e13] m1=0 | (m1=1 & m13>m11)|(m1=2 & m13>m12)|(m1=3 & m13>m13) -> (m1'=3);

*// one of the other pairs change so may need to reset matching*

[e21] **true** -> (m1'=(m1=1)?0:m1);

[e31] **true** -> (m1'=(m1=1)?0:m1);

[e22] **true** -> (m1'=(m1=2)?0:m1);

[e32] **true** -> (m1'=(m1=2)?0:m1);

[e23] **true** -> (m1'=(m1=3)?0:m1);

[e33] **true** -> (m1'=(m1=3)?0:m1);

**endmodule**

*// construct further men through renaming*

**module** man2=man1[m1=m2, m11=m21, e11=e21, e12=e22, e13=e23, m12=m22, e21=e31, e22=e32, e23=e33, m13=m23, e31=e11, e32=e12, e33=e13 ] **endmodule**

**module** man3=man1[m1=m3, m11=m31, e11=e31, e12=e32, e13=e33, m12=m32, e21=e11, e22=e12, e23=e13, m13=m33, e31=e21, e32=e22, e33=e23 ] **endmodule**

*//------------------------------------------------------*

*// module for first woman*

**module** woman1

*// do not need to store the matching (can work it out from the men's variables)*

*// man 1 wants to change*

[e11] ( m1!=N1 & m2!=N1 & m3!=N1 ) | (m1=N1 & w11>w11)|(m2=N1 & w11>w12)|(m3=N1 & w11>w13) -> **true**;

[e21] ( m1!=N1 & m2!=N1 & m3!=N1 ) | (m1=N1 & w12>w11)|(m2=N1 & w12>w12)|(m3=N1 & w12>w13) -> **true**;

[e31] ( m1!=N1 & m2!=N1 & m3!=N1 ) | (m1=N1 & w13>w11)|(m2=N1 & w13>w12)|(m3=N1 & w13>w13) -> **true**;

**endmodule**

*// construct further women through renaming*

**module** woman2=woman1[ N1=N2, w11=w21, e11=e12, e21=e22, e31=e32, w12=w22, e12=e13, e22=e23, e32=e33, w13=w23, e13=e11, e23=e21, e33=e31 ] **endmodule**

**module** woman3=woman1[ N1=N3, w11=w31, e11=e13, e21=e23, e31=e33, w12=w32, e12=e11, e22=e21, e32=e31, w13=w33, e13=e12, e23=e22, e33=e32 ] **endmodule**

*//------------------------------------------------------*

*// reward structure: expected rounds*

**rewards** "rounds"

**true** : 1;

**endrewards**

Experiments

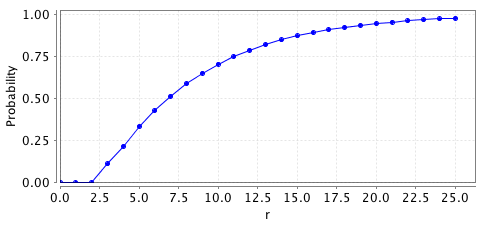
*// probability reached a stable matching by round R*

**const** **int** r; *// bound of number of rounds*

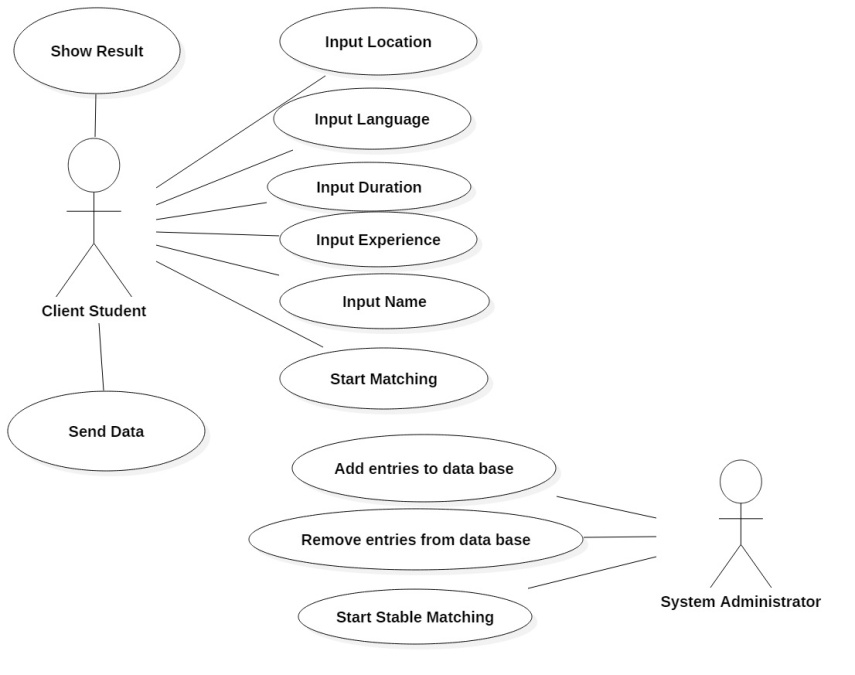
**P**=?[ **F**<=r "deadlock" ]

*// expected number of rounds to reach a stable matching*

**R**{"rounds"}=? [ **F** "deadlock" ]



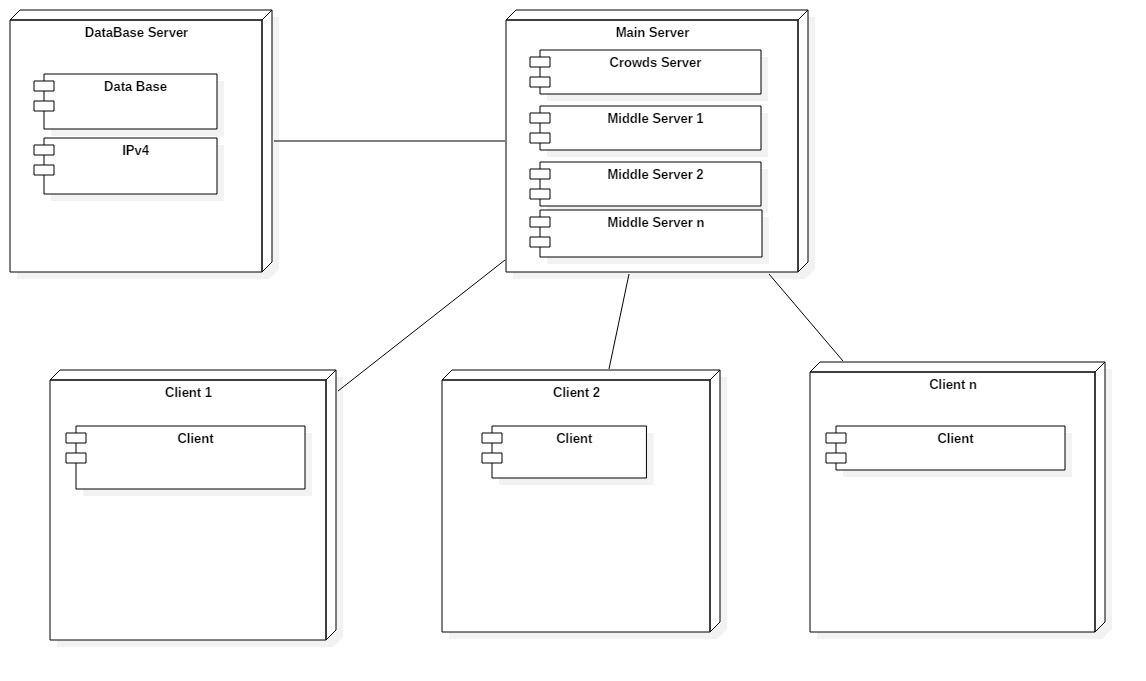
Use Case Diagram



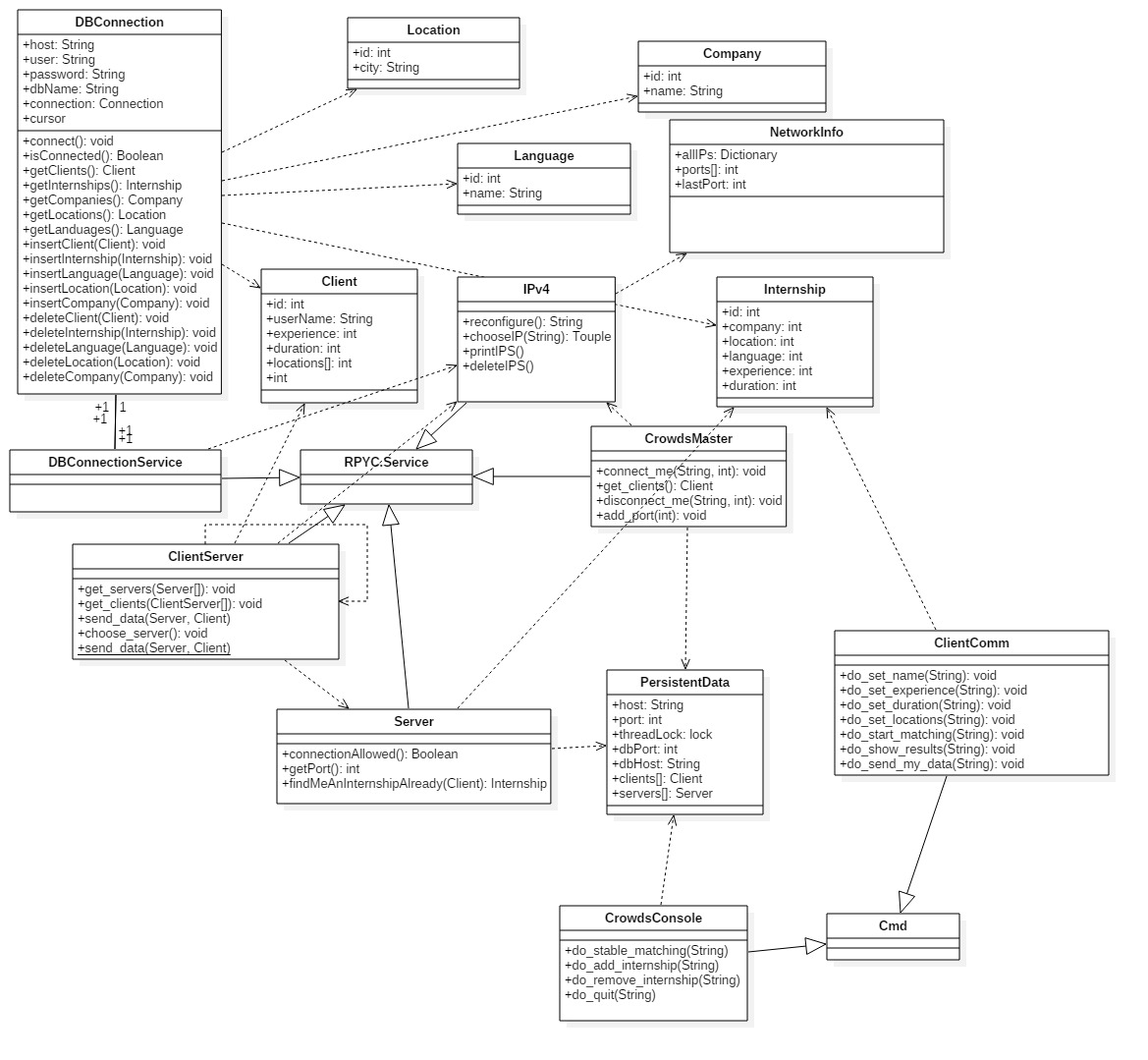
Architecture and design

The application has a client-server architecture. No particular design patterns have been required.

Deployment Diagram



Class Diagram



Source Code

**rpDBMethods.py**

|  |
| --- |
| import pymysql |
|  | import thread |
|  |  |
|  | import rpyc |
|  |  |
|  |  |
|  | class Client: |
|  | def \_\_init\_\_(self): |
|  | self.id = 0 |
|  | self.userName = "" |
|  | self.experience = 0 |
|  | self.duration = 0 |
|  | self.password\_encr = "" |
|  | self.password\_salt = "" |
|  | self.locations = [] |
|  | self.languages = [] |
|  |  |
|  |  |
|  | class Internship: |
|  | def \_\_init\_\_(self): |
|  | self.id = 0 |
|  | self.company = 0 |
|  | self.location = 0 |
|  | self.language = 0 |
|  | self.experience = 0 |
|  | self.duration = 0 |
|  |  |
|  | def \_\_str\_\_(self): |
|  | stri = str(self.id)+" "+str(self.company)+" +"+str(self.location)+" "+str(self.language)+" "+str(self.experience)+" "+str(self.duration) |
|  | return stri |
|  |  |
|  | def clone(self): |
|  | internship2 = Internship() |
|  | internship2.id = self.id |
|  | internship2.company = self.company |
|  | internship2.location = self.location |
|  | internship2.language = self.language |
|  | internship2.experience = self.experience |
|  | internship2.duration = self. duration |
|  | return internship2 |
|  |  |
|  |  |
|  | class Company: |
|  | def \_\_init\_\_(self): |
|  | self.id = 0 |
|  | self.name = "" |
|  |  |
|  |  |
|  | class Location: |
|  | def \_\_init\_\_(self): |
|  | self.id = 0 |
|  | self.city = "" |
|  |  |
|  |  |
|  | class Language: |
|  | def \_\_init\_\_(self): |
|  | self.id = 0 |
|  | self.name = "" |
|  |  |
|  |  |
|  | class DBConnectionService(rpyc.Service): |
|  | class exposed\_DBConnection(object): |
|  | def \_\_init\_\_(self): |
|  | self.host = "localhost" |
|  | self.user = "root" |
|  | self.password = "10lase" |
|  | self.dbName = "internship" |
|  | self.connection = None |
|  | self.cursor = None |
|  |  |
|  | def connect(self): |
|  | self.connection = pymysql.connect(self.host, self.user, self.password, self.dbName) |
|  |  |
|  | def isConnected(self): |
|  | if self.connection is None: |
|  | return False |
|  | return True |
|  |  |
|  | def exposed\_getClients(self): |
|  | self.connect() |
|  | self.cursor = self.connection.cursor() |
|  | clients = [] |
|  | self.cursor.execute("SELECT \* FROM Client") |
|  | resultSet = self.cursor.fetchall() |
|  | for row in resultSet: |
|  | client = Client() |
|  | client.id = row[0] |
|  | client.userName = row[1] |
|  | client.experience = row[4] |
|  | client.duration = row[5] |
|  | self.cursor.execute("SELECT \* FROM Client\_Language WHERE client\_id=" + str(client.id)) |
|  | resultSet2 = self.cursor.fetchall() |
|  | for e in resultSet2: |
|  | client.languages.append(e[0]) |
|  | self.cursor.execute("SELECT \* FROM Client\_Location WHERE client\_id=" + str(client.id)) |
|  | resultSet2 = self.cursor.fetchall() |
|  | for e in resultSet2: |
|  | client.languages.append(e[1]) |
|  | clients.append(client) |
|  | self.cursor.close() |
|  | self.connection.close() |
|  | return clients |
|  |  |
|  | def exposed\_getInternships(self): |
|  | self.connect() |
|  | self.cursor = self.connection.cursor() |
|  | internships = [] |
|  | self.cursor.execute("SELECT \* FROM Internship") |
|  | resultSet = self.cursor.fetchall() |
|  | for row in resultSet: |
|  | internship = Internship() |
|  | internship.id = row[0] |
|  | internship.company = row[2] |
|  | internship.location = row[3] |
|  | internship.language = row[4] |
|  | internship.experience = row[5] |
|  | internship.duration = row[6] |
|  | internships.append(internship) |
|  | self.cursor.close() |
|  | self.connection.close() |
|  | return internships |
|  |  |
|  | def exposed\_getCompanies(self): |
|  | self.connect() |
|  | self.cursor = self.connection.cursor() |
|  | companies = [] |
|  | sql = "SELECT \* FROM Company" |
|  | self.cursor.execute(sql) |
|  | resultSet = self.cursor.fetchall() |
|  | for row in resultSet: |
|  | company = Company() |
|  | company.id = row[0] |
|  | company.name = row[1] |
|  | companies.append(company) |
|  | self.cursor.close() |
|  | self.connection.close() |
|  | return companies |
|  |  |
|  | def exposed\_getLocations(self): |
|  | self.connect() |
|  | self.cursor = self.connection.cursor() |
|  | locations = [] |
|  | sql = "SELECT \* FROM Location" |
|  | self.cursor.execute(sql) |
|  | resultSet = self.cursor.fetchall() |
|  | for row in resultSet: |
|  | location = Location() |
|  | location.id = row[0] |
|  | location.city = row[1] |
|  | locations.append(location) |
|  | self.cursor.close() |
|  | self.connection.close() |
|  | return locations |
|  |  |
|  | def exposed\_getLanguages(self): |
|  | self.connect() |
|  | self.cursor = self.connection.cursor() |
|  | languages = [] |
|  | sql = "SELECT \* FROM Language" |
|  | self.cursor.execute(sql) |
|  | resultSet = self.cursor.fetchall() |
|  | for row in resultSet: |
|  | language = Language() |
|  | language.id = row[0] |
|  | language.name = row[1] |
|  | languages.append(language) |
|  | self.cursor.close() |
|  | self.connection.close() |
|  | return languages |
|  |  |
|  | def exposed\_insertClient(self, client): |
|  | self.connect() |
|  | self.cursor = self.connection.cursor() |
|  | sql = """INSERT INTO Clieant(client\_id, username, password\_encr, password\_salt, |
|  | experience, duration) VALUES ('%d','%s','%s','%s','%d','%d')""" % \ |
|  | (client.id, client.userName, client.password\_encr, client.password\_salt, \ |
|  | client.experience, client.duration) |
|  | self.cursor.execute(sql) |
|  | for l in client.languages: |
|  | sql = """INSERT INTO Client\_Language(language\_id, client\_id) |
|  | VALUES ('%d', '%d')""" % (l, client.id) |
|  | self.cursor.execute(sql) |
|  | for l in client.locations: |
|  | sql = """INSERT INTO Client\_Location(client\_id, location\_id) |
|  | VALUES ('%d', '%d')""" % (client.id, l) |
|  | self.cursor.execute(sql) |
|  | self.connection.commit() |
|  | self.cursor.close() |
|  | self.connection.close() |
|  |  |
|  | def exposed\_insertInternship(self, internship): |
|  | self.connect() |
|  | self.cursor = self.connection.cursor() |
|  | sql = """INSERT INTO Internship (internship\_id, company\_id, |
|  | location\_id, language\_id, experience, duration) values ('%d', '%d', |
|  | '%d', '%d', '%d', '%d')""" % (internship.id, internship.company, internship.location, \ |
|  | internship.language, internship.experience, internship.duration) |
|  | self.cursor.execute(sql) |
|  | self.connection.commit() |
|  | self.cursor.close() |
|  | self.connection.close() |
|  |  |
|  | def exposed\_insertLanguage(self, language): |
|  | self.connect() |
|  | self.cursor = self.connection.cursor() |
|  | sql = """INSERT INTO Language (language\_id, name) |
|  | values ('%d', '%s')""" % (language.id, language.name) |
|  | self.cursor.execute(sql) |
|  | self.connection.commit() |
|  | self.cursor.close() |
|  | self.connection.close() |
|  |  |
|  | def exposed\_insertLocation(self, location): |
|  | self.connect() |
|  | self.cursor = self.connection.cursor() |
|  | sql = """INSERT INTO Location (location\_id, city) |
|  | values ('%d', '%s')""" % (location.id, location.city) |
|  | self.cursor.execute(sql) |
|  | self.connection.commit() |
|  | self.cursor.close() |
|  | self.connection.close() |
|  |  |
|  | def exposed\_insertCompany(self, company): |
|  | self.connect() |
|  | self.cursor = self.connection.cursor() |
|  | sql = """INSERT INTO Company (company\_id, name) |
|  | values ('%d', '%s')""" % (company.id, company.name) |
|  | self.cursor.execute(sql) |
|  | self.connection.commit() |
|  | self.cursor.close() |
|  | self.connection.close() |
|  |  |
|  | def exposed\_deleteClient(self, client): |
|  | self.connect() |
|  | self.cursor = self.connection.cursor() |
|  | sql = "DELETE FROM Client\_Location WHERE client\_id='%d'" % (client.id) |
|  | self.cursor.execute(sql) |
|  | sql = "DELETE FROM Client\_Language WHERE client\_id='%d'" % (client.id) |
|  | self.cursor.execute(sql) |
|  | sql = "DELETE FROM Client WHERE client\_id='%d'" % (client.id) |
|  | self.cursor.execute(sql) |
|  | self.connection.commit() |
|  | self.cursor.close() |
|  | self.connection.close() |
|  |  |
|  | def exposed\_deleteInternship(self, internship): |
|  | self.connect() |
|  | self.cursor = self.connection.cursor() |
|  | sql = "Delete FROM Internship where internship\_id = '%d'" % internship.id |
|  | self.cursor.execute(sql) |
|  | self.connection.commit() |
|  | self.cursor.close() |
|  | self.connection.close() |
|  |  |
|  | def exposed\_deleteLanguage(self, language): |
|  | self.connect() |
|  | self.cursor = self.connection.cursor() |
|  | sql = "DELETE FROM Internship WHERE language\_id='%d'" % (language.id) |
|  | self.cursor.execute(sql) |
|  | sql = "DELETE FROM Client\_Language WHERE language\_id='%d'" % (language.id) |
|  | self.cursor.execute(sql) |
|  | sql = "DELETE FROM Language WHERE language\_id='%d'" % (language.id) |
|  | self.cursor.execute(sql) |
|  | self.connection.commit() |
|  | self.cursor.close() |
|  | self.connection.close() |
|  |  |
|  | def exposed\_deleteLocation(self, location): |
|  | self.connect() |
|  | self.cursor = self.connection.cursor() |
|  | sql = "DELETE FROM Client\_Location WHERE location\_id='%d'" % (location.id) |
|  | self.cursor.execute(sql) |
|  | sql = "DELETE FROM Internship WHERE location\_id='%d'" % (location.id) |
|  | self.cursor.execute(sql) |
|  | sql = "DELETE FROM Location WHERE location\_id='%d'" % (location.id) |
|  | self.cursor.execute(sql) |
|  | self.connection.commit() |
|  | self.cursor.close() |
|  | self.connection.close() |
|  |  |
|  | def exposed\_deleteCompany(self, company): |
|  | self.connect() |
|  | self.cursor = self.connection.cursor() |
|  | sql = "DELETE FROM Internship WHERE company\_id='%d'" % (company.id) |
|  | self.cursor.execute(sql) |
|  | sql = "DELETE FROM Company WHERE company\_id='%d'" % (company.id) |
|  | self.cursor.execute(sql) |
|  | self.connection.commit() |
|  | self.cursor.close() |
|  | self.connection.close() |
|  |  |
|  | def exposed\_Sal(self): |
|  | print("PaBafta") |
|  |  |
|  | def server\_start(): |
|  | ThreadedServer(DBConnectionService, port=1234,protocol\_config={"allow\_public\_attrs": True, "allow\_all\_attrs": True}).start() |
|  |  |
|  | from rpyc.utils.server import ThreadedServer |
|  |  |
|  | if \_\_name\_\_ == "\_\_main\_\_": |
|  | print "ok" |
|  | thread.start\_new\_thread(server\_start,()) |
|  | print "also ok" |
|  | while 1==1: |
|  | pass |

**crowdsboss.py**

|  |
| --- |
| import rpyc |
|  | import subprocess |
|  | import socket |
|  | import thread |
|  | import os |
|  | import time |
|  | from cmd import Cmd |
|  | from rpyc.utils.server import ThreadedServer |
|  | from rpDBMethods import Client |
|  | from rpDBMethods import Internship |
|  | from rpDBMethods import Location |
|  | from rpDBMethods import Language |
|  | from rpDBMethods import Company |
|  |  |
|  | global data |
|  |  |
|  | class PersistentData(): |
|  | def \_\_init\_\_(self): |
|  | self.clients = [] |
|  | #self.clientsRPYC = [] |
|  | self.servers = [] |
|  | # to be changed accordingly |
|  | self.host = "10.142.0.3" |
|  | self.port = 1024 |
|  | self.dbHost = "10.142.0.2" |
|  | self.dbPort = 1234 |
|  |  |
|  | class CrowdsConsole(Cmd): |
|  | def do\_stable\_matching(self, args): |
|  | global data |
|  | ports = 1024 |
|  | for i in data.clients: |
|  | ports += 1 |
|  | thread.start\_new\_thread( |
|  | os.system, |
|  | ("python "+ "Server.py " + str(data.host) + " " + str(ports) + " " + str(data.dbHost) + " " + str(data.dbPort) + " " + str(data.host) + " " + str(data.port),)) |
|  | # get port of each server in a list |
|  | data.servers.append((data.host, ports)) |
|  | time.sleep(1) |
|  | for c in data.clients: |
|  | # name should be adjusted accordingly |
|  | conn = rpyc.connect(c[0], c[1], config={"allow\_all\_attrs": True}) |
|  | s = conn.root |
|  | s.get\_servers(data.servers) |
|  | s.choose\_server() |
|  | clients = list(data.clients) |
|  | clients.remove(c) |
|  | s.get\_clients(clients) |
|  | print "ok" |
|  | conn.close() |
|  |  |
|  | def do\_show\_conn(self, args): |
|  | global data |
|  | print str(data.clients) |
|  |  |
|  | def do\_add\_internship(self, args): |
|  | connection = rpyc.connect(data.dbHost, data.dbPort, config={"allow\_all\_attrs": True}) |
|  | db = connection.root.DBConnection() |
|  | internships = db.getInternships() |
|  | languages = db.getLanguages() |
|  | locations = db.getLocations() |
|  | companies = db.getCompanies() |
|  | internship\_id = 0 |
|  | for internship in internships: |
|  | if internship.id > internship\_id: |
|  | internship\_id = internship.id |
|  | internship\_id = internship\_id + 1 |
|  | print("Select one of the companies' IDs") |
|  | ids = [] |
|  | for c in companies: |
|  | ids.append(c.id) |
|  | print(str(c.id) + " " + c.name) |
|  | company = -1 |
|  | while (company == -1): |
|  | company\_string = input(); |
|  | try: |
|  | company = int(company\_string) |
|  | if company not in ids: |
|  | company = -1 |
|  | except ValueError: |
|  | company = -1 |
|  | if (company == -1): |
|  | print ("Invalid input, try again") |
|  | print("Select one of the languages' IDs") |
|  | ids = [] |
|  | for c in languages: |
|  | ids.append(c.id) |
|  | print(str(c.id) + " " + c.name) |
|  | language = -1 |
|  | while (language == -1): |
|  | language\_string = input(); |
|  | try: |
|  | language = int(language\_string) |
|  | if language not in ids: |
|  | language = -1 |
|  | except ValueError: |
|  | language = -1 |
|  | if (language == -1): |
|  | print ("Invalid input, try again") |
|  | print("Select one of the locations' IDs") |
|  | ids = [] |
|  | for c in locations: |
|  | ids.append(c.id) |
|  | print(str(c.id) + " " + c.city) |
|  | location = -1 |
|  | while (location == -1): |
|  | location\_string = input(); |
|  | try: |
|  | location = int(location\_string) |
|  | if location not in ids: |
|  | location = -1 |
|  | except ValueError: |
|  | location = -1 |
|  | if (location == -1): |
|  | print ("Invalid input, try again") |
|  | duration\_string =input("Please insert the duration of the internship") |
|  | duration = -1 |
|  | while (duration == -1): |
|  | try: |
|  | duration = int(duration\_string) |
|  | except ValueError: |
|  | duration = -1 |
|  | print ("Invalid input, try again") |
|  | experience\_string = input("Please insert your experience") |
|  | experience = -1 |
|  | while (experience == -1): |
|  | try: |
|  | experience = int(experience\_string) |
|  | except ValueError: |
|  | experience = -1 |
|  | print ("Invalid input, try again") |
|  | myInternship = Internship(); |
|  | myInternship.id = internship\_id; |
|  | myInternship.experience = experience; |
|  | myInternship.duration = duration; |
|  | myInternship.company = company; |
|  | myInternship.language = language; |
|  | myInternship.location =location |
|  | db.insertInternship(myInternship) |
|  | connection.close() |
|  |  |
|  | def do\_remove\_internship(self, args): |
|  | connection = rpyc.connect(data.dbHost, data.dbPort, config={"allow\_all\_attrs": True}) |
|  | db = connection.root.DBConnection() |
|  | internships = db.getInternships() |
|  | print ("Select the id of the internship to delete") |
|  | ids = [] |
|  | for i in internships: |
|  | ids.append(i.id) |
|  | print(str(i.id)+ " " +str(i.company)) |
|  | id\_string = input() |
|  | id = -1 |
|  | while (id == -1): |
|  | try: |
|  | id = int(id\_string) |
|  | if id not in ids: |
|  | id = -1 |
|  | except ValueError: |
|  | id = -1 |
|  | print("Invalid input, please try again") |
|  | myInternship = Internship(); |
|  | myInternship.id = id |
|  | db.deleteInternship(myInternship) |
|  |  |
|  | def do\_quit(self, args): |
|  | print "Program terminated." |
|  | raise SystemExit |
|  |  |
|  |  |
|  | class CrowdsMaster(rpyc.Service): |
|  | #called by client |
|  | def exposed\_connect\_me(self, ip, port): |
|  | global data |
|  | #client = rpyc.connect(ip, port, config={"allow\_all\_attrs": True}) |
|  | #data.clientsRPYC.append(client) |
|  | data.clients.append((ip, port)) |
|  |  |
|  | def exposed\_get\_clients(self): |
|  | global data |
|  | return data.clients |
|  |  |
|  | def exposed\_disconnect\_me(self, ip, port): |
|  | global data |
|  | data.clients.remove((ip, port)) |
|  |  |
|  | #not using this anymore |
|  | def exposed\_add\_port(self, port): |
|  | global data |
|  | data.servers.append(port) |
|  |  |
|  |  |
|  | def server\_start(): |
|  | ThreadedServer(CrowdsMaster, port=data.port, |
|  | protocol\_config={"allow\_public\_attrs": True, "allow\_all\_attrs": True}).start() |
|  |  |
|  |  |
|  | def main(): |
|  | global data |
|  | data = PersistentData() |
|  | thread.start\_new\_thread(server\_start, ()) |
|  | myConsole = CrowdsConsole() |
|  | myConsole.prompt = ">" |
|  | myConsole.cmdloop("Server has started\nAvailable commands\nstable\_mathcing\nadd\_internship\nremove\_internship\n") |
|  |  |
|  | if \_\_name\_\_ == "\_\_main\_\_": |
|  | main() |

**Server.py**

|  |
| --- |
| import sys |
|  | import rpyc |
|  | import socket |
|  | import thread |
|  | from rpDBMethods import Internship |
|  | from rpDBMethods import Client |
|  | from rpyc.utils.server import ThreadedServer |
|  |  |
|  | data = None |
|  |  |
|  | class PersistentData: |
|  | def \_\_init\_\_(self, myHost, myPort, dbHost, dbPort, crowdsHost, crowdsPort): |
|  | self.host = myHost |
|  | self.port = myPort |
|  | self.threadLock = thread.allocate\_lock() |
|  | self.dbPort = dbPort |
|  | self.dbHost = dbHost |
|  | #self.db = rpyc.connect(dbHost, dbPort, config={"allow\_all\_attrs": True}) |
|  | #self.crowds = rpyc.connect(crowdsHost, crowdsPort, config={"allow\_all\_attrs": True}) |
|  | #self.crowds.add\_port(self.port) |
|  | #self.crowds.close() |
|  |  |
|  | class MiddleServer(rpyc.Service): |
|  | connections = 0 |
|  |  |
|  | #ensure stable matching |
|  | def on\_connect(self): |
|  | MiddleServer.connections = MiddleServer.connections + 1 |
|  |  |
|  | def on\_disconnect(self): |
|  | MiddleServer.connections = MiddleServer.connections - 1 |
|  |  |
|  | #IPv4 when working with sockes, not suitable for rpyc |
|  | def generatePort(self): |
|  | global data |
|  | s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) |
|  | host = data.host |
|  | port = 1024 |
|  | for attempt in range(1024,4000,1): |
|  | try: |
|  | result = s.connect((host, attempt)) |
|  | except Exception: |
|  | port = attempt |
|  | break |
|  | return port |
|  |  |
|  | def exposed\_connectionAllowed(self): |
|  | global data |
|  | with data.threadLock: |
|  | if MiddleServer.connections > 1: |
|  | return False |
|  | return True |
|  |  |
|  | def exposed\_getPort(self): |
|  | global data |
|  | return data.port |
|  |  |
|  | def exposed\_findMeAnInternshipAlready(self, client): |
|  | global data |
|  | with data.threadLock: |
|  | internship = None |
|  | connection = rpyc.connect(data.dbHost, data.dbPort, config={"allow\_all\_attrs":True}) |
|  | db = connection.root.DBConnection() |
|  | internships = db.getInternships(); |
|  | bestMatch = 0 |
|  | for i in internships: |
|  | match = 0; |
|  | if client.experience >= i.experience: |
|  | match = match + 1 |
|  | if client.duration == i.duration: |
|  | match = match + 1 |
|  | if len(client.locations) == 0: |
|  | match = match + 2 |
|  | else : |
|  | if (i.location is not None) and (i.location in client.locations): |
|  | match = match + 2 |
|  | if len(client.languages) == 0: |
|  | match = match + 3 |
|  | if (i.language is not None) and (i.language in client.languages): |
|  | match = match + 3 |
|  | if match > bestMatch: |
|  | bestMatch = match |
|  | internship = str(i) |
|  | connection.close() |
|  | return internship |
|  |  |
|  | #only for testing reasons |
|  | def sayHello(self): |
|  | print ("hello, i've connected on port " + str(self.exposed\_getPort())) |
|  |  |
|  | #only for testing reasons |
|  | def wait(self): |
|  | while(True): |
|  | pass |
|  |  |
|  | def main(): |
|  | global data |
|  | data = PersistentData(sys.argv[1], sys.argv[2], sys.argv[3], sys.argv[4],sys.argv[5], sys.argv[6]) |
|  | ThreadedServer(MiddleServer, port = int(sys.argv[2]), |
|  | protocol\_config={"allow\_public\_attrs": True, "allow\_all\_attrs": True}).start() |
|  |  |
|  | if \_\_name\_\_ == "\_\_main\_\_": |
|  | main() |

**client.py**

|  |
| --- |
| import cmd |
|  | import rpyc |
|  | from rpyc.utils.server import ThreadedServer |
|  | import sys |
|  | import random |
|  | from rpDBMethods import Client |
|  | import thread |
|  | from Server import MiddleServer |
|  |  |
|  |  |
|  | class ClientComm(cmd.Cmd): |
|  | def do\_set\_name(self, args): |
|  | global pref |
|  | global name |
|  | pref += 1 |
|  | name = args |
|  | print "Name: " + name |
|  |  |
|  | def do\_set\_experience(self, args): |
|  | global pref |
|  | global exp |
|  | pref += 1 |
|  | exp = int(args) |
|  | print "Experience: " + str(exp) |
|  |  |
|  | def do\_set\_duration(self, args): |
|  | global pref |
|  | global duration |
|  | pref += 1 |
|  | duration = int(args) |
|  | print "Duration: " + str(duration) |
|  |  |
|  | def do\_set\_locations(self, args): |
|  | global pref |
|  | global locations |
|  | pref += 1 |
|  | locations = args.split() # map(str, args.split()) |
|  | print "Locations: " + str(locations) |
|  |  |
|  | def do\_set\_languages(self, args): |
|  | global pref |
|  | global languages |
|  | pref += 1 |
|  | languages = args.split() |
|  | print "Languages: " + str(languages) |
|  |  |
|  | def do\_start\_matching(self, args): |
|  | global pref, port, ip |
|  | if pref > 4: |
|  | conn = rpyc.connect("10.142.0.3", 1024, config={"allow\_all\_attrs": True}) |
|  | conn.root.connect\_me(ip, port) |
|  | conn.close() |
|  | # ThreadedServer(ClientServer, port=port, |
|  | # protocol\_config={"allow\_public\_attrs": True, "allow\_all\_attrs": True}).start() |
|  | thread.start\_new\_thread(start\_server, ()) |
|  |  |
|  | def do\_show\_results(self, args): |
|  | global result |
|  | print str(result) |
|  |  |
|  | def do\_see\_client(self, args): |
|  | print next\_client[0] |
|  | print str(next\_client[1]) |
|  |  |
|  | def do\_send\_my\_data(self, args): |
|  | # check if data can be sent |
|  | me = Client() |
|  | me.userName = name |
|  | me.experience = exp |
|  | me.duration = duration |
|  | me.locations = locations |
|  | me.languages = languages |
|  | global result |
|  | result = ClientServer.send\_data(my\_server, me) |
|  |  |
|  |  |
|  | class ClientServer(rpyc.Service): |
|  | def exposed\_get\_servers(self, server\_list): |
|  | global servers |
|  | servers = server\_list |
|  |  |
|  | def exposed\_get\_clients(self, client\_list): |
|  | global next\_client |
|  | next\_client = client\_list[random.randint(0, len(client\_list)-1)] |
|  |  |
|  | def exposed\_send\_data(self, server, data): |
|  | rand = random.randint(1, 100) |
|  | if rand < 20: |
|  | conn = rpyc.connect(server[0], server[1], config={"allow\_all\_attrs": True}) |
|  | internship = conn.root.findMeAnInternshipAlready(data) |
|  | return internship |
|  | else: |
|  | conn = rpyc.connect(next\_client[0], next\_client[1], config={"allow\_all\_attrs": True}) |
|  | internship = conn.root.send\_data(server, data) |
|  | return internship |
|  |  |
|  | @staticmethod |
|  | def send\_data(server, data): |
|  | rand = random.randint(1, 100) |
|  | if rand < 20: |
|  | conn = rpyc.connect(server[0], server[1], config={"allow\_all\_attrs": True}) |
|  | internship = conn.root.findMeAnInternshipAlready(data) |
|  | return internship |
|  | else: |
|  | conn = rpyc.connect(next\_client[0], next\_client[1], config={"allow\_all\_attrs": True}) |
|  | internship = conn.root.send\_data(server, data) |
|  | return internship |
|  |  |
|  | def exposed\_choose\_server(self): |
|  | global my\_server |
|  | # my\_server = servers[random.randint(0, len(servers)-1)] |
|  | for i in servers: |
|  | # print "Connecting to "+i[0]+" "+str(i[1]) |
|  | conn = rpyc.connect(i[0], i[1], config={"allow\_all\_attrs": True}) |
|  | if conn.root.connectionAllowed(): |
|  | my\_server = i |
|  | conn.close() |
|  |  |
|  |  |
|  | def start\_server(): |
|  | global port, ip |
|  | ThreadedServer(ClientServer, port=port, |
|  | protocol\_config={"allow\_public\_attrs": True, "allow\_all\_attrs": True}).start() |
|  |  |
|  |  |
|  | my\_server = None |
|  | random.seed() |
|  | pref = 0 |
|  | next\_client = None |
|  | name = "" |
|  | exp = 0 |
|  | duration = 0 |
|  | locations = [] |
|  | languages = [] |
|  | servers = [] |
|  | ip = sys.argv[1] # start with self ip and port |
|  | port = int(sys.argv[2]) |
|  | result = None |
|  | cC = ClientComm() |
|  | cC.prompt = 'Client>>' |
|  | cC.cmdloop('Client') |