Mục lục

Danh sách hình vẽ

I. Đặt vấn đề

Công nghệ càng ngày càng hiện đại, các sản phẩm điện tử ngày càng đa dạng và phong phú, do đó nhu cầu truy cập mạng internet trở nên rất phổ biến với con người. Vấn cần đặt ra ở đây đó là với nhiều sản phẩm như thế khi người dùng truy cập vào mạng internet thì kích cỡ của màn hình cũng là hoàn toàn không giống nhau. Vậy làm thế nào để với từng kích cỡ màn hình hiển thị sẽ có thể hiển thị khái quát nội dung chính cho người dùng nhìn thấy khi đọc báo hay tìm kiếm thông tin trên mạng.

Ngoài ra, trong mạng máy tính thì mô hình client-server là mô hình được sử dụng phổ biến nhất và là mô hình nổi tiếng trong mạng máy tính. Ý tưởng của mô hình client-server đó là mô hình con-client gửi các yêu cầu tới cho mô hình cha-server. Tại đây server sẽ sử lý yêu cầu và hồi đáp lại cho client dưới dạng thông điệp (messages).

Tuy nhiên, mô hình còn client-server còn nhiều hạn chế như sau [1] :

Thứ nhất, khi nhiều Client gửi yêu cầu đến Server, Server không có khả năng đáp ứng, xử lý hết các yêu cầu. Như vậy, các yêu cầu không được xử lý sẽ gây ra hiện tượng "thắt nút cổ chai". Khi đó, tài nguyên mạng bị chiếm dụng và tắc nghẽn sẽ ảnh hưởng đến tốc độ trong mạng.

Thứ hai, Client có thể gửi các yêu cầu giống nhau. Khi đó, Server sẽ phải thực hiện công việc xử lý trùng lặp, gây ra việc tốn thời gian chờ của Client và hiệu suất của Server sẽ giảm xuống.

Thứ ba, hệ thống thiếu tính mềm dẻo trong khả năng mở rộng khi có sự thay đổi về cơ sở dữ liệu, phương thức truy nhập.

Như đã nói ở trên, Server sẽ phải làm việc nhiều và sẽ phải lặp lại một số công việc làm lãng phí thời gian hồi đáp với client hơn. Ngoài ra còn có các công việc tầng thấp như làm việc với cơ sở dữ liệu, tính toán kích thước màn hình..v..v

Áp dụng Agent vào hệ thống nhằm mục đích đơn giản hóa công việc của server, giảm thiểu hạn chế nêu trên và nó sẽ có trách nhiệm giao tiếp ở tầng thấp khi được yêu cầu.

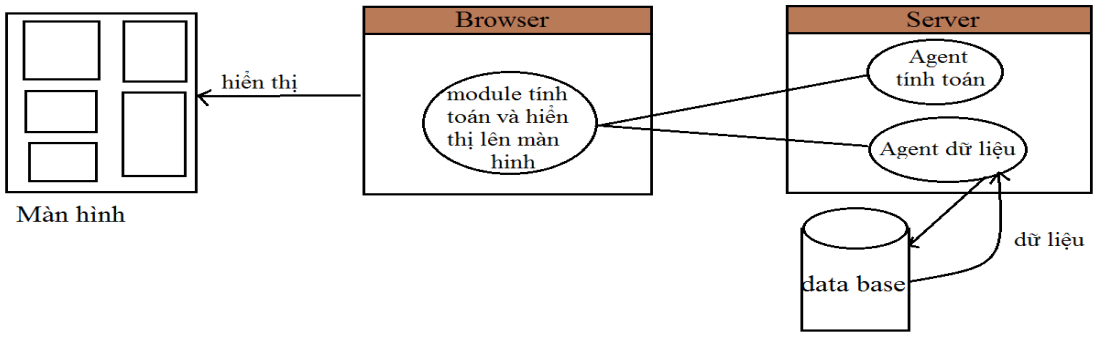
II. Giới thiệu

Các vấn đề được nêu ở mục đặt vấn đề cũng được quan tâm từ rất nhiều các chuyên gia phát triển phần mềm trên nền web và đưa ra một số giải pháp nhằm lấp đầy sự hạn chế trên. Nhưng các hệ thống chỉ dừng lại ở bài toán cụ thể mà họ áp dụng vào, chưa được ứng dụng cao.

Qua đó, sử dụng agent vào mô hình client-server để nhằm xử lý thiếu sót trong mô hình client-server. Đã có nhiều người đi trước áp dụng agent vào mô hình client-server để hoàn thiện mô hình này và đạt được mục đích của họ.

Sử dụng Agent vào mô hình client-server là nhằm mục đích đơn giản công việc của server hơn. Ngoài ra, Agent còn có trách nhiệm tính toán kích cỡ màn hình để hiển thị sao cho phù hợp với màn hình hơn. Trong khóa luận sắp tới tôi sẽ hướng Agent theo việc truyền tải dữ liệu.

Để dễ hình dùng hơn tôi sẽ mô tả bằng hình vẽ dưới đây :



Hình 1 : Mô hình hệ thống tính toán hiển thị layout

Khi người dùng sử dụng, hệ thống sẽ gửi một yêu cầu tới server. Tại đây server kích hoạt các agent. Agent tính toán kích thước màn hình sẽ làm việc và gửi cho browser biết. Đồng thời agent làm việc với cơ sở dữ liệu sẽ đẩy dữ liệu lên cho browser, nó sẽ xử lý lại và hiển thị lên màn hình cho người dùng.

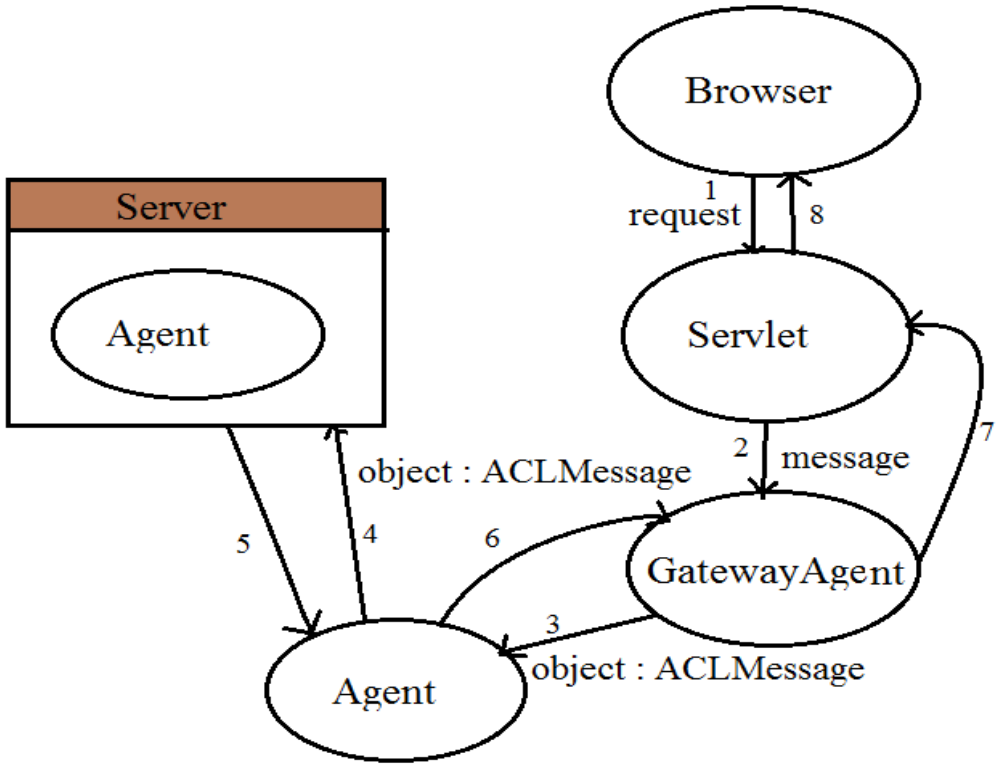
- Sẽ có 2 phần việc chính dành cho Agent:
  - Tính toán : xác định kích thước của màn hình, tính toán để sắp xếp layout sao cho phù hợp với màn hình.

- Dữ liệu : lấy dữ liệu từ database gửi lên browser . Browser lúc này đã được tính toán và sắp xếp layout sao cho hợp lý với người sử dung.

III. Mô hình hệ thống sử dụng Agent

1. Sơ đồ mô hình
   Hiện tại, tôi đang lập trình một mô hình đơn giản gồm có :

   - Một browser : người dùng ghi nội dung rồi gửi tới server.
   - Servlet : nhận các yêu cầu từ browser và đẩy xuống cho GatewayAgent.
   - GatewayAgent : lấy yêu cầu tới từ browser gửi tới agent tương ứng và nó sẽ gửi tới server.



Hình 2: Mô hình hệ thống sử dụng Agent

Ví dụ mô tả hoạt động của hệ thống.

Giá trị đầu vào: tên quyển sách cần mua.

Giá trị mong đợi trả về: giá tiền của quyển sách (nếu có).

- Mô tả các bước :
    - Bước 1 : người dùng gõ nôi dụng (tên sách) sau đó ấn gửi, browser gửi yêu cầu tới Servlet
    - Bước 2: Servlet nhận được yêu cầu từ browser và đẩy nội dung xuống cho GatewayAgent.
    - Bước 3: Tại GatewayAgent, sau khi nhận được yêu cầu từ Servlet gửi tới. Nó xác định AID của Agent con và sử dụng ACLMessage gửi đi thông điệp.
    - Bước 4: Agent con nhận được yêu cầu của GatewayAgent. Nó gửi tới Agent tại Server và chờ hồi đáp.
    - Bước 5: Agent ở Server nhận được yêu cầu bắt đầu tìm trong cơ sở dữ liệu sau đó xác nhận đối tượng cần gửi và hồi đáp.
    - Bước 6: Agent con nhận được hồi đáp từ Server, gửi lại cho GatewayAgent.
    - Bước 7: GatewayAgent đẩy nội dung từ Agent con gửi tới lên Servlet
    - Bước 8: Servlet đẩy lên cho Browser và hiển thị lên màn hình cho người dùng nhìn thấy.

2. Giải thích khái niệm

    a. Agent là gì?

    Định nghĩa về Agent được nói như sau trong quyển "Developing Intelligent Agent Systems- Micheal WiniKnoff – RMIT" :

    "Agent là một hệ thống máy tính được đặt trong các môi trường và nó tự chủ hành động trong môi trường đó để đạt được mục tiêu thiết kế của nó".

    Qua đó ta có thể nhận thấy Agent cũng là khái niệm chỉ con người , tổ chức, cũng có những mục tiêu cần đạt được, có hành động tự chủ thuộc về mình hay sự giao tiếp giữa các agent với nhau.

Agent là một hệ thống tính toán hành động tự chủ trong một môi trường và có khả năng cảm nhận môi trường và tác động tới môi trường đó.

- Các tính chất cơ bản của Agent [2] :

Tính chủ động (pro-activeness) là khi nhận thấy có sự thay đổi trong môi trường. Agent sẽ chủ động xác định chuối hành động cần thực thi và kích hoạt chuỗi hành động này.

Tính xã hội(social ability) là Agent không chỉ chủ dộng trong các hành động của mình nhằm đạt được mục tiêu của nó mà còn có khả năng tương tác lien lạc với các Agent khác ngay cả trong môi trường khác.

Tính tự trị(autonomous) là mỗi một Agent thì có một trạng thái riêng và nó hoàn toàn độc lập với các agent khác. Các agent khác cũng k thể truy cập vào trạng thái này ở đây thể hiện tính tự trị bên trong chính agent còn tự trị hành động tức là agent có sẵn nhiệm vụ của mình, nó có thể tự quyết định hành động tiếp theo của mình mà không cần bàn tay con người can thiệp.

Tính phản ứng(reactivity) là khả năng cảm nhận được sự thay đổi của môi trường mà nó đang hoạt động và có thể kịp thời đáp ứng với những sự thay đổi này. Điều này rất có ích cho các môi trường có tính thay đổi cao.

Tính di động(mobility) của agent thể hiện qua sự chuyển giao agent từ môi trường này qua môi trường khác.

- Có 2 loại di động :
  - Di động manh (String mobility) : khi Agent di chuyển thì nó có khả năng di chuyể cả mã chương trình và mã thi hành tới một môi trường khác

  - Di động yếu (Weak mobility) : khi Agent di chuyển thì nó chỉ có thể di chuyển mã chương trình giữa môi trường thi hành với nhau. Đôi khi mã nguồn có thể kèm theo một số dữ liệu khởi tạo nhưng trang thái thì hành thì không thể mang theo.

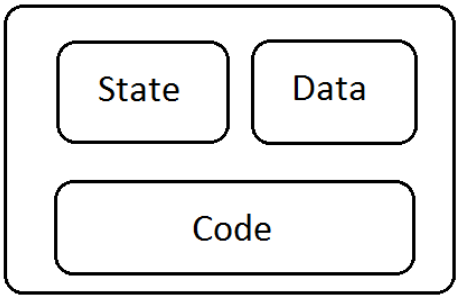
- Tính hướng đích(goal-orient) và tính chủ động (pro-activeness) là để đánh giá độ tự trị của một agent mạnh hay yếu.

b. Mobile Agent?

- Mobile Agent là thành phần phần mềm gồm mã chương trình, dữ liệu và trạng thái hoạt động.

- Mobile Agent là 1 dạng của mobile code (chương trình chuyển mã đến Client và thực thi tại đó).

Cấu trúc của Mobile Agent :

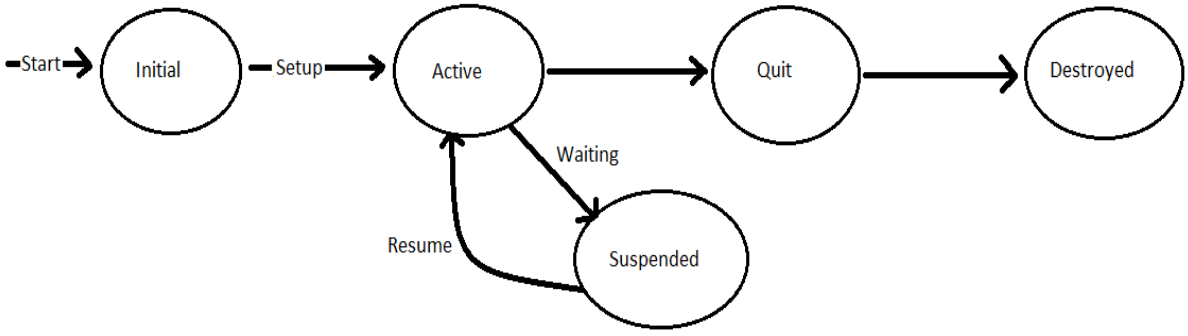


Hình 3 : Cấu trúc mobile agent

- Dữ liệu (data) : dữ liệu của agent chính là các giá trị của các biến trong lớp agent và chỉ có chính lớp agent đó mới có khả năng sử dụng được các dữ liệu này.

- Trạng thái (state) : Agent có khả năng cảm nhận được sự thay đổi của môi trường và đáp ứng kịp thời với sự thay đổi đó. Do vậy, agent có nhiều trạng thái khác nhau và ứng với mỗi trạng thái lại có những hành động khác nhau.

Dưới đây là hình ảnh mô tả các trạng thái Agent :



Hình 4 : Vòng đời của Agent

- Các trạng thái bên trên cũng mô tả vòng đời của một Agent :

  - Start : khi bắt đầu một Agent hệ thống sẽ khỏi tạo một con Agent để sẵn sang cho các hành động tiếp theo.

  - Initial : Agent được khởi tạo một giá trị AID xác định và một nhiệm vụ nhất định.

  - Active : Với nhiệm vụ được xác định từ bước Initial, tại đây Agent bắt đầu thực thi các hành vi được xác định sẵn để hoàn thành nhiệm vụ được giao. Trong quá trình thực thi , Agent có khả năng sẽ phải di chuyển từ môi trường này qua môi trường khác. Trước khi di chuyên, Agent sẽ tạm dừng lại quá trình thực thi, đóng gói toàn bộ thông tin hiện tại và bắt đầu di trú. Sau khi di trú xong, Agent lại quay trở lại kích hoạt các hành vi tạm dừng trước và tiếp tục tới khi nhiệm vụ hoàn thành.

  - Quit : kết thúc thực thi, nhiệm vụ hoàn thành.

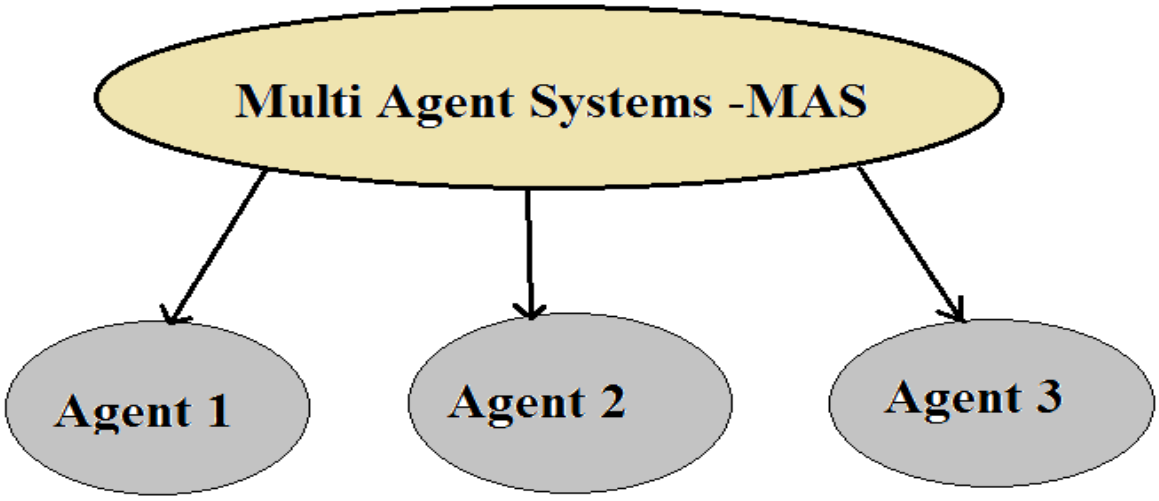
  - Destroy : Hủy Agent.

- Mã nguồn (code) : Gồm toàn bộ mã thực thi chương trình và được xây dựng bởi lập trình viên.

c. Hệ thống đa Agents

Một trong những đặc tính của Agent nêu bên trên đó là tính xã hội điều này thể hiện qua việc xây dựng một hệ thốn đa Agent.

Mỗi một Agent khi bắt đầu khởi tạo ra đã mang trong mình một nhiệm vụ xác định và hệ thống đa Agents là gồm rất nhiều Agent và mỗi Agent đều mang tròng mình một nhiệm vụ riêng nhưng mọi thứ đều nhằm đạt được mục tiêu cần đạt được của hệ thông, để dễ hiểu hơn chúng ta hãy liên tưởng hệ thống đa Agents như là một tổ chức. Một hệ thống gồm các Agent hoạt động là hệ thống đa Agent - Multi Agent Systems (MAS).



Hình 5 : Hệ thống đa agents

Trong mỗi một tổ chức thì luôn có một người quản lý để thúc đẩy và kiểm soát công việc của tổ chức. Tại hệ thống đa Agent, một Agent quản lý các Agent khác trong hệ thống được gọi là MAS. Agent này có nhiệm vụ quản lý thông tin và tham gia vào các quá trình trao đổi các Agent trong hệ thống hoặc hệ thống khác.

Đôi khi AMS có thể tham gia vào trạng thái "Active" của một Agent. Nó có thể tạo và sửa đổi thông tin của Agent này hay gửi thông điệp cho Agent để yêu cầu Agent làm nhiệm vụ mà nó yêu cầu. MAS là agent không thể thiếu và là duy nhất trong hệ thống.

d. JADE Platform

JADE (Java Agent Development Framework) là phần mềm mã nguồn mở được cài đặt hoàn toàn bằng ngôn ngữ java.

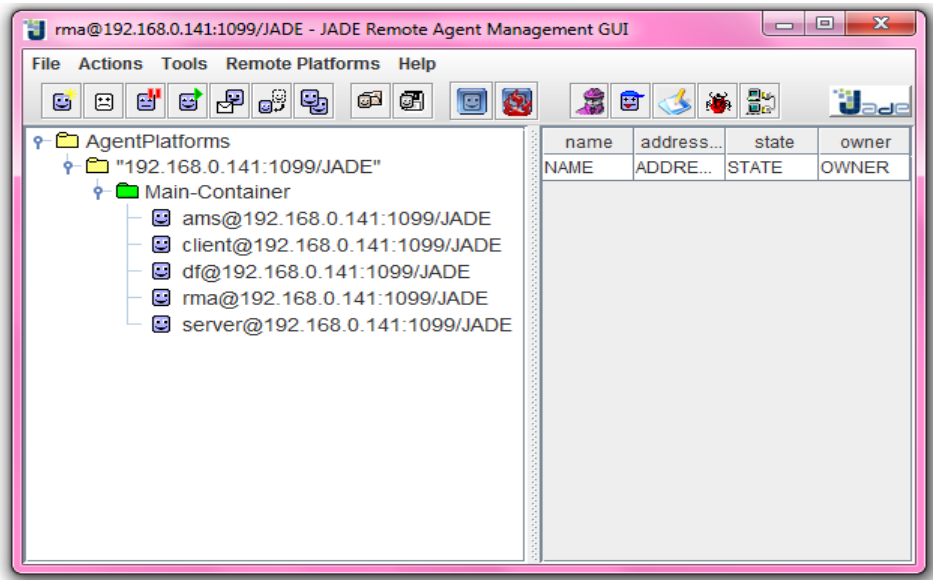
JADE là mã nguồn mở trong năm 2000 và được phân phối bởi Telecom Italia dưới giấy phép LGPL.

JADE còn là phần mềm trung gian nhằm hỗ trợ việc phát triển hệ thống đa Agents. Cấu trúc của nó bao gồm :

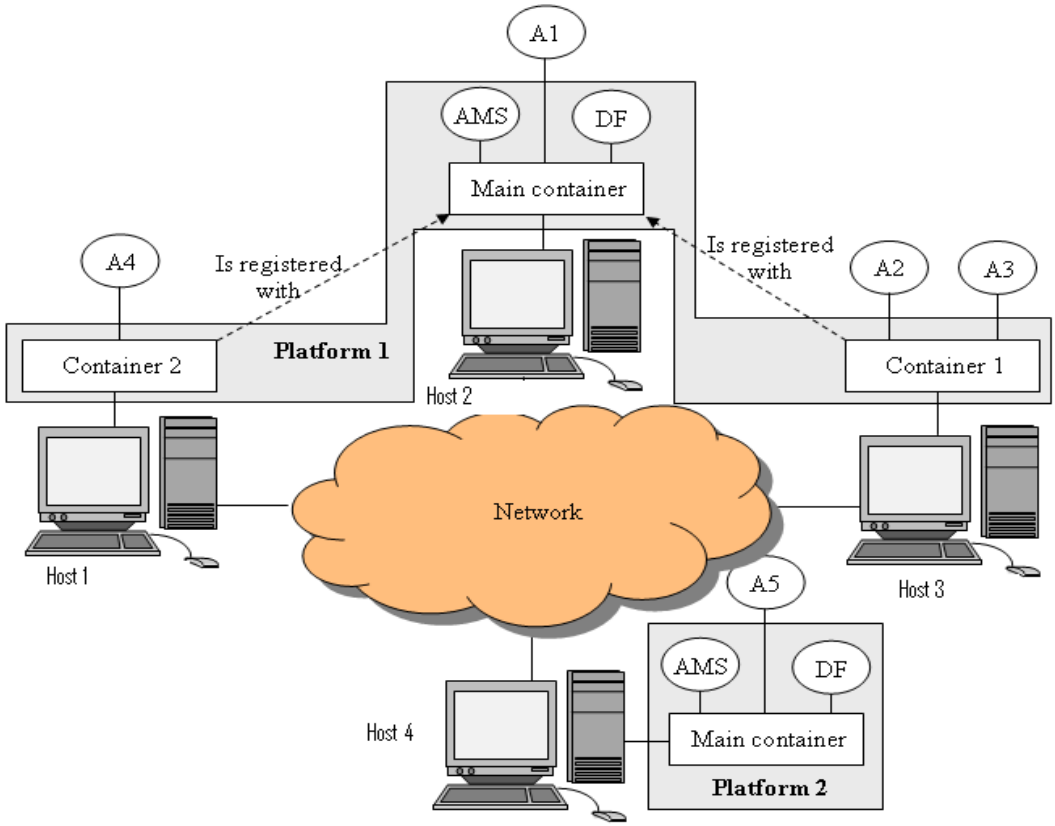
- Runtime environment (**Container**) : môi trường để các JADE agent sống và môi trường này phải chắc chắn rằng được kích hoạt tại máy chủ trước khi các agent trong đó hoạt động.

- Thư viện : gồm các lớp dành cho các lập trình viên sử dụng để phát triển agent.
- Công cụ đồ họa : quản lý và giám sát hành vi của các agent đang hoạt động.

Giao diện đồ họa quản lý của JADE sau khi được kích hoạt :



Hình 6: Giao diện đồ họa JADE



Hình 7 : Kiến trúc của JADE

- Khái niệm Container và Platform :

Như ở trên có ghi mỗi runtime environment là một container. Và mỗi container có thể gồm một hoặc nhiều agent trong đó. Một tập hợp gồm nhiều container được gọi là **Platform**. Trong mỗi platform sẽ có một container đặc biệt và được gọi là **Main container** container này sẽ luôn được kích hoạt. Các container còn lại sẽ phải biết thông tin của main container để có thể giao tiếp được với nó.

3. Lập trình

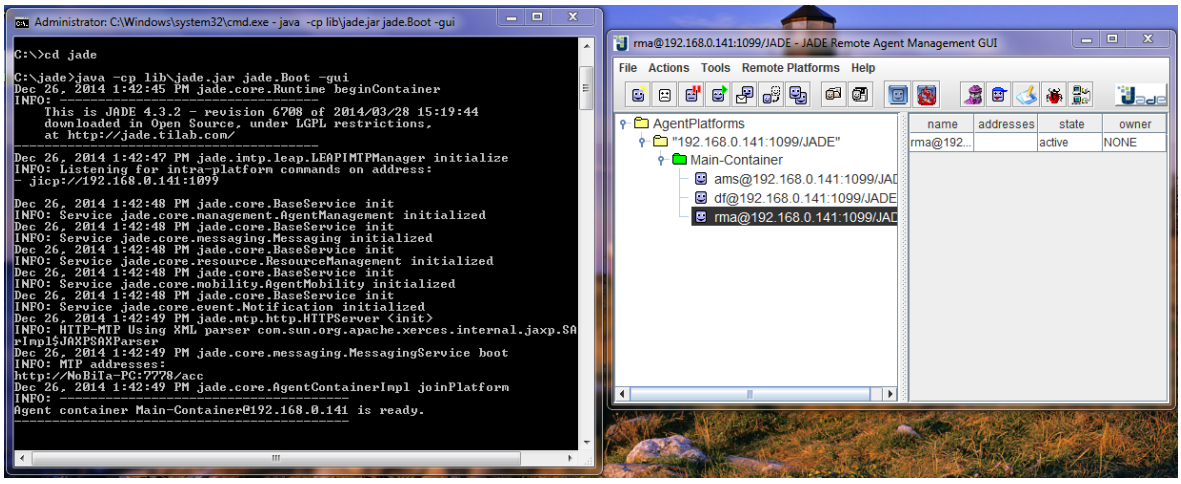
- Cài đặt JADE cho máy tính, tạo môi trường quản lý Agent :
  - Bước 1 : tải jade về từ trang chủ : http://jade.tilab.com/
  - Bước 2 : giải nén file jade và để vào ổ C:\

Sau khi giải nén thư mục jade có cấu trúc như sau :

```
jade/
     |---License
     |---classes
     |---demo
     |---doc/
     |      |---index.html
     |---lib/
     |      |---http.jar
     |      |---iiop.jar
     |      |---jade.jar
     |      |---jadeTools.jar
     |      |---commons-codec/
     |      |                |---commons-codec-1.3.jar
     |---src/
     |      |---demo
     |      |---examples
     |      |---FIPA
     |      |---jade
```

Hình 8: Cấu trúc thư mục jade

- Bước 3 : vào thư mục jade trong ổ C:\ bật command line và gõ "java –cp lib\jade.jar jade.Boot –gui"
  command line chạy và màn hình sẽ hiện thị như sau :



Như vậy là chúng ta đã cài đặt thành công JADE trên máy tính.

➢ Lập trình với JADE

- Agent có 3 loại hành vi :
  - Hành vi loại One-shot :

Hành vi loại One-shot là loại hành vi mà hành động tại phương thức action() của nó chỉ thực hiện duy nhất một lần. Các hành vi thuộc loại one-shot sẽ kế thừa từ lớp **jade.core.behaiours.OneShotBehaiour** và lớp này đã cài đặt sẵn phương thức done() bằng việc trả về giá trị là true.

Ví dụ :

```
public class MyOneShotBehaviour extends OneShotBehaviour{
    public void action(){
        System.out.println("Finish");
    }
}
```

Hình 9: Hành vi loại One-Short

- Hành vi loại Cylic :

Hành vi loại Cylic là loại hành vi sẽ không bao giờ hoàn thành. Tức là phương thức action() sẽ được thực hiện công việc giống nhau mỗi lần được gọi tới. các hành vi loại Cylic sẽ kế thừa từ lớp **jade.core.behaiours.CyclicBehaiour** lớp này đã cài sẵn phương thức done() trả về giá trị false (không kết thúc).

Ví dụ :

```
public class MyCyclicBehaviour extends CyclicBehaviour{
    public void action(){
        System.out.println("Continue");
    }
}
```

Hình 10 : Hành vi loại Cylic

- Hành vi loại Generic :

  Hành vi loại Generic là loại hành vi thực hiên các công việc khác nhau tùy thuộc vào đầu vào của trạng thái.

Ví dụ :

```
public class MyThreeStepBehaviour extends Behaviour{
    private int step = 0;
    public void action(){
        switch(step){
            case 0:
                System.out.println("step = 0");
                step++;
                break;
            case 1:
                System.out.println("step = 1");
                step++;
                break;
            case 2:
                System.out.println("step = 2");
                step++;
                break;
        }
    }

    public boolean done(){
        return step == 3;
    }
}
```

Hình 11: Hành vi loại Generic

➢ Lập lịch công việc cho Agent tại một thời điểm nhất định :

Trong JADE có cung cấp cho chúng ta hai lớp **WakerBehaiour** và **TickerBehaour (**trong lớp jade.core.behaviours)để cài đặt các hành vi và xác định nhiệm vụ cần thực thi vào một thời điểm xác định.

- **WakerBehaiour** :

  Các phương thức action() và done() đều được cài đặt sẵn. Chúng ta chỉ cần phương thức trừu tượng **handleElapsedTimeout()**. Bên trong phương thức này sẽ viết công việc mà sau một khoảng thời gian nào đó từ lúc bắt đầu nó sẽ thực thi và kết thúc hành vi.

Ví du :

```
public class MyAgent extends Agent{
    protected void setup(){
        addBehaviour(new WakerBehaviour(this, 10000){
            protected void handleElapsedTimeout(){
                System.out.println("finish");
            }
        });
    }
}
```

Hình 12: Sử dụng lớp WakerBehaviour

Màn hình sẽ in ra dòng chữ “finish” sau 10 giây kể từ khi Agent này được kích hoạt.

- **TickerBehaviour** :

Cũng như WakerBehaviour phương thức action() và done() đã có sẵn ta chỉ thêm phương thức trừu tượng **ontick()** và hành động trong phương thức này sẽ được lặp đi lặp lại sau mỗi khoảng thời gian được cài đặt sẵn cho tới khi nào Agent kết thúc. TickerBehaviour là không bao giờ hoàn thành.

Ví dụ :

```
public class MyAgent extends Agent{
    protected void setup(){
        addBehaviour(new TickerBehaviour(this, 10000) {
            int i = 0;
            @Override
            protected void onTick() {
                System.out.println("time : " + i);
                i++;
            }
        });
    }
}
```

Hình 13: Sử dụng lớp TickerBehaviour

Màn hình sẽ in ra “time 1” “time 2”…. Cứ sau 10 giây nó lại được gọi và in ra dòng chữ tới khi Agent kết thúc.

- Giao tiếp giữa các Agent

  Trong việc giao tiếp giữa các Agent với nhau chúng ta sẽ sử dụng lớp ACLMessage có sẵn trong JADE.

  - Gửi lời nhắn :

    Điều đầu tiên khi muốn gửi lời nhắn tới một agent khác chúng ta phải điền thông tin vào các trường của đối tượng trong lớp ACLMessage. Kế tiếp, gọi phương thức send() của lớp Agent.
    Dưới đây là đoạn mã đơn giản gửi tới Agent “chat2” với lời nhắn “Today it’s raining”

```
public void sendMessage(){
    ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
    msg.addReceiver(new AID("chat2", AID.ISLOCALNAME));
    msg.setContent("Today it's raning");
    send(msg);
}
```

Hình 14: Gửi lời nhắn sử dụng ACLMessage

  - Nhận lời nhắn :

    JADE runtime tự động lưu lời nhắn được gửi đến từ các Agent khác vào hàng đợi của Agent nhân sau đó Agent này sẽ nhận lấy lời nhắn từ hang đợi của nó bằng phương thức receive(). Nếu không có lời nhắn nào được lưu phương thức sẽ trả về NULL.

    Ví dụ :

```
public class MyCyclicBehaviour extends CyclicBehaviour{
    public void action(){
        ACLMessage request = myAgent.receive();
        if (request != null) {
            System.out.print("Request : " + request.getContent());
        }else
            block();
    }
}
```

Hình 15: Nhận lời nhắn sử dụng ACLMessage

IV. Dự kiến kết quả đạt được

Dựa trên khóa luận của anh Trương Văn Hưng – Biểu diễn trực quan các liên kết web tôi sẽ áp dụng Agetn vào để xử lý bài toán ứng với từng kích thước màn hình hiển thị thì ta phải có sắp xếp phù hợp. Server sẽ bớt phải làm những công việc dư thừa và đơn giản hơn.

V. Tài liệu tham khảo

1. Nguyễn Văn Lân, "Mô hình Proxy động sử dụng Agent phần mềm"

2. Lê Thị Hồng Hạnh, "Kiến trúc phần mềm dựa trên Agent".

3. "JadeGateway", [Online]. Available : http://jade.tilab.com/doc/tutorials/JadeGateway.pdf

4. John Wiley & Sons, "Developing multi-agent systems with JADE".

5. "Lập trình hướng Agent", [Online]. Available: http://doc.edu.vn/tai-lieu/lap-trinh-huong-agent-22501/

Ý kiến đánh giá:

……
……
……
……
……
……
……
……

Điểm số: ……. Điểm chữ: ……

Hà Nội, ngày tháng năm 20
Giảng viên đánh giá
(Ký, ghi rõ họ tên)