|  |
| --- |
| ĐẠI HỌC QUỐC GIA HÀ NỘI  **TRƯỜNG ĐẠI HỌC CÔNG NGHỆ**    **Nhóm 7:** Lê Văn Hùng  Vương Thị Mỵ  Phan Văn Nam  Bùi Thị Mai  Nguyễn Thị Thùy Dương  **TÌM HIỂU HỆ ĐA TÁC TỬ VỚI NỀN TẢNG JADE**  **Giảng viên hướng dẫn**: TS. Nguyễn Thị Huyền Châu  **Hà Nội - 2016** |

**MỤC LỤC**

[ĐẶT VẤN ĐỀ 1](#_Toc444964645)

[CHƯƠNG I: CƠ SỞ LÍ THUYẾT 2](#_Toc444964646)

[I.1. Tác tử 2](#_Toc444964647)

[I.1.1. Khái niệm tác tử 2](#_Toc444964648)

[I.1.2. Các đặc điểm tính chất của tác tử 3](#_Toc444964649)

[I.1.3. Phân biệt tác tử và đối tượng 3](#_Toc444964650)

[I.1.4. Phân loại tác tử 4](#_Toc444964651)

[I.2. Hệ đa tác tử (Multi Agents System – MAS) 5](#_Toc444964652)

[I.2.1. Khái niệm 5](#_Toc444964653)

[I.2.2. Vấn đề phối hợp của các tác tử trong hệ đa tác tử 6](#_Toc444964654)

[I.2.3. Đánh giá số nền tảng đa tác tử [6] 7](#_Toc444964655)

[I.2.4. Ứng dụng của hệ đa tác tử 10](#_Toc444964656)

[I.3. JADE framework 10](#_Toc444964657)

[I.3.1. Giới thiệu 10](#_Toc444964658)

[I.3.2. Kiến trúc JADE 11](#_Toc444964659)

[I.3.2. Các thành phần JADE 12](#_Toc444964660)

[I.3.2. Sự giao tiếp giữa các tác tử trong JADE 14](#_Toc444964661)

[I.3.3. Các hành vi của tác tử 17](#_Toc444964662)

[I.4. Dịch vụ trang vàng (Yellow page service) trong JADE 21](#_Toc444964663)

[CHƯƠNG II: PHÂN TÍCH THIẾT KẾ PHẦN MỀM 24](#_Toc444964664)

[II.1. Yêu cầu bài toán 24](#_Toc444964665)

[II.2. Kiến trúc 24](#_Toc444964666)

[II.3. Mô hình ca sử dụng của hệ thống 25](#_Toc444964667)

[CHƯƠNG III: CÀI ĐẶT VÀ TRIỂN KHAI 26](#_Toc444964668)

[III.1 Cài đặt JADE 26](#_Toc444964669)

[III.2. Lập trình JADE 28](#_Toc444964670)

[III.3. Mô phỏng một hệ thống nhà thông minh 28](#_Toc444964671)

[CHƯƠNG 4: KẾT LUẬN 29](#_Toc444964672)

**DANH MỤC HÌNH ẢNH**

[Hình 1: Tác tử và môi trường của nó 2](#_Toc444964625)

[Hình 2: Mô hình phân loại tác tử 4](#_Toc444964626)

[Hình 3: Mô hình hệ đa tác tử 6](#_Toc444964627)

[Hình 4: Mối quan hệ giữa các thành phần kiến trúc chính trong JADE 11](#_Toc444964628)

[Hình 5: Mô hình UML mô tả mối quan hệ giữa các thành phần kiến trúc chính 12](#_Toc444964629)

[Hình 6: Mô hình truyền thông điệp không đồng bộ JADE 14](#_Toc444964630)

[Hình 7: Minh họa quá trình phía gửi gửi thông điệp cho phía nhận 15](#_Toc444964631)

[Hình 8: Minh họa việc phía gửi gửi tin nhắn đến phía nhận và đồng thời phía nhận gửi tin nhắn trả lời cho phía gửi 16](#_Toc444964632)

[Hình 9: Ví dụ về kiểu hành vi thực hiện một lần 18](#_Toc444964633)

[Hình 10: Ví dụ kiểu hành vi thực hiện mãi mãi 18](#_Toc444964634)

[Hình 11: Ví dụ về kiểu hành vi tổng quát 19](#_Toc444964635)

[Hình 12: Ví dụ về kiểu hành vi đánh thức 20](#_Toc444964636)

[Hình 13: Ví dụ về kiểu hành vi thực hiện có chu kì 20](#_Toc444964637)

[Hình 14: Dịch vụ trang vàng 21](#_Toc444964638)

[Hình 15: Kiến trúc nhà thông minh mô phỏng 25](#_Toc444964639)

[Hình 16: Minh họa tải framework JADE tại trang chủ 26](#_Toc444964640)

[Hình 17: Cấu trúc thư mục JADE sau khi giải nén 26](#_Toc444964641)

[Hình 18: Mô tả cách thêm các thư viện JADE để cấu hình một project 27](#_Toc444964642)

[Hình 19: Minh họa cách cấu hình để chạy một project JADE 28](#_Toc444964643)

[Hình 20: Mô tả cách cấu hình các tham số trong JADE để chạy một chương trình 28](#_Toc444964644)

DANH MỤC BẢNG BIỂU

[Bảng 1: Kết quả thực nghiệm đối với các thông số 9](#_Toc444714868)

[Bảng 2: Kết quả đánh giá trên phương diện khi hoạt động 9](#_Toc444714869)

[Bảng 3: Kết quả so sánh thời gian yêu cầu 9](#_Toc444714870)

**DANH SÁCH CÁC TỪ VIẾT TẮT**

JADE **J**ava **a**gent **de**velopment Framework

FIPA **F**oundation for **I**ntelligent **P**hysical **A**gents

API **A**pplication **P**rograming **I**nterface

# ĐẶT VẤN ĐỀ

Khi mà các chương trình phần mềm ngày đang trở nên ngày càng phức tạp, không gian bộ nhớ máy tính ngày càng lớn thì những người lập trình cần biết thêm các phương pháp lập trình mới để có thể tổ chức mã nguồn của họ tốt hơn. Từ phương pháp cổ điển như lập trình hướng thủ tục cho đến lập trình hướng đối tượng có thể sẽ không còn phù hợp với nhiều chương trình hiện tại bởi vì các chương tình hiện tại hướng đến sự tương tác giữa các thành phần, sự phân tán khi đó nếu sử dụng các phương pháp cũ có vẻ như sẽ rất khó khăn trong việc lập trình, rồi bảo trì hay tiến hóa, … Lập trình hướng tác tử là một phương pháp tiếp cận mới cho người lâp trình. Tác tử có vẻ như là một cụm từ khá mới bởi có thể đây là lần đầu tiên xuất hiện trong suy nghĩ của nhiều người. Tuy nhiên, ở nhiều nước trên thế giới khái niệm này đã xuất hiện từ khá lâu đời từ những năm 1990. Lập trình hướng tác tử được xem như là sự trừu tượng hóa mức độ cao hơn so với lập trình hướng đối tượng. Chính vì sự khá lạ lẫm với khái niệm này, cũng như mong muốn tìm hiểu một phương pháp lập trình, phương pháp phát triển phần mềm mới liên quan đến tác tử, cũng như mong muốn tìm hiểu cách xây dụng một hệ đa tác tử thì có gì mới nên nhóm chúng em lựa chọn chủ đề “Hệ đa tác tử với JADE”. Trong đó, JADE là một framework được chúng em lựa chọn để phát triển ứng dụng liên quan đến hệ đa tác tử.

# CHƯƠNG I: CƠ SỞ LÍ THUYẾT

## I.1. Tác tử

Chương này trước hết giới thiệu các khái niệm về tác tử, tổng quan các công nghệ tác tử, kiến trúc tác tử, các ngôn ngữ lập trình và các công cụ phát triển. Tiếp theo sẽ mô tả các đặc tả của FIPA- tập các tiêu chuẩn phổ biến nhất và được chấp nhận rộng rãi cho phát triển các nền tảng và ứng dụng đa tác tử.

JADE là một nền tảng tuân theo các đặc tả FIPA và hơn nữa nó còn mở rộng mô hình FIPA trong một số lĩnh vực như tác tử cho thiết bị di động, tác tử cho dịch vụ web.

### I.1.1. Khái niệm tác tử

Thuật ngữ tác tử (Agent) hay tác tử phần mềm đã được sử dụng rộng rãi và xuất hiện trong nhiều lĩnh vực nghiên cứu như trí tuệ nhân tạo, cơ sở dữ liệu, các tài liệu về hệ điều hành và mạng máy tính.

Tác tử là một hệ thống máy tính được đóng gói nằm trong một số môi trường có khả năng hành động độc lập và linh hoạt trong môi trường đó nhằm đáp ứng mục tiêu thiết kế của nó. [1]

**MÔI TRƯỜNG**

**TÁC TỬ**

Sensor thu nhận tin hiệu

Phản ứng lại

Hình 1: Tác tử và môi trường của nó

Hình 1 mang đến một cái nhìn trừu tượng về tác tử. Trong hình này, chúng ta thấy hành động phản ứng lại môi trường được tạo ra bởi chính tác tử để ảnh hưởng lại đối với môi trường của chính nó. Trong hầu hết các lĩnh vực phức tạp hợp lý, một tác tử sẽ không hoàn toàn kiểm soát được môi trường của nó mà tác tử chỉ có thể kiểm soát tốt nhất một phần môi trường ảnh hưởng đến nó. Từ góc nhìn của các tác tử, điều này có nghĩa là cùng một hành động được thi hành trong hai hoàn cảnh giống hệ nhau có thể cho kết quả hoàn toàn khác nhau và trong trường hợp cụ thể kết quả đó có thể là kết quả không mong muốn. Tuy nhiên, thông thường một tác tử sẽ có sẽ một tập các hành động sẵn có, tập các hành động này mô tả khả năng có thể có của tác tử, tuy nhiên các hành động trong tập này đều được thi hành trong mọi tình huống. Ví dụ: Hành động nâng một cái bàn chỉ có thể thực hiện được nếu cân nặng của cái bàn đủ nhỏ để tác tử có thể nâng được nó hoặc một ví dụ khác là hành động mua một chiếc xe Ferrari, kết quả của hành động này có thể thất bại như tác tử không đủ tiền để mua chiếc xe đó.

### I.1.2. Các đặc điểm tính chất của tác tử

Tác tử mang một số tính chất sau [5]:

* Tính tự chủ: Các tác tử đóng gói một số trạng thái (không thể truy cập từ các tác tử khác) và ra quyết định làm cái gì đều dựa trên những trạng thái này mà không có sự can thiệp trực tiếp của con người hay những thứ khác.
* Tính phản ứng: Tác tử được đặt trong một môi trường (có thể là thế giới vật chất, hoặc một người dùng thông qua một giao diện người dùng hoặc một tập hợp các tác tử khác, môi trường Internet hoặc môi trường được kết hợp từ những môi trường này) và có thể phản hồi một cách kịp thời với những thay đổi xảy ra trong môi trường đó.
* Tính chủ động: Các tác tử không đơn giản là chỉ hành động để phản ứng lại môi trường mà còn có thể thực hiện các hành vi có mục đích một cách chủ động.
* Khả năng xã hội: Các tác tử tương tác với các tác tử khác thông qua một số loại ngôn ngữ giao tiếp tác tử (agent communication language – ACL) và thường có khả năng tham gia vào các hoạt động xã hội (như giải quyết vấn đề hợp tác và đàm phán) để đạt được mục tiêu của chúng.

Xét một ví dụ cụ thể sau: Tưởng tượng rằng một hệ thống lái máy bay tự động tự kiểm soát một chiếc máy bay, mục đích là hạ cánh an toàn xuống một số sân bay. Chúng ta hy vọng hệ thống lên kế hoạch làm thế nào để đạt được điều này (có thể là chuẩn bị kế hoạch từ trước, chứ không phải khi cần mới bắt đầu chuẩn bị), kể cả kế hoạch cho các mục tiêu nhỏ hơn (ví dụ như khi đạt đến độ cao 30 000 feet thì tiến về phía bắc với tốc độ …) đây chính là tính chủ động. Chúng ta mong muốn hệ thống thực thi đúng như kế hoạch nhưng không phải thực thi một cách mù quáng. Như vậy, trong trường hợp bất khả kháng (ví dụ, một sự thay đổi trong điều kiện thời tiết, một lỗi trong máy bay, một yêu cầu kiểm soát không lưu), chúng ta hy vọng hệ thống để đáp ứng với tình hình mới cho phù hợp, có thể phản ứng lại trong thời gian cho phép chứ không phải mất hàng giờ để cân nhắc xem cần làm gì tiếp theo, đây chính là tính phản ứng (reactiveness). Chúng ta mong muốn hệ thống lái máy bay tự động có thể phối hợp tác với các bộ điều khiển hàng không và các máy bay khác để đạt được mục đích của mình đây chính là khả năng xã hội.

### I.1.3. Phân biệt tác tử và đối tượng

Đối tượng là một thực thể của thế giới thực, chúng có các trạng thái và các phương thức của mình. Đối tượng được xem như là bị động bởi các hành vi của nó chỉ được thực hiện khi một thực thể bên ngoài gửi một thông điệp đến nó (truyền tham số thông qua việc gọi phương thức).

Tác tử là các thực thể tự trị bởi chúng có thể biết được trách nhiệm của mình. Tác tử là những thực thể có tính tương tác, chúng sử dụng nhiều dạng thông điệp dể tương tác với nhau. Những thông điệp này có thể hỗ trợ việc gọi phương thức, cũng như là thông báo cho các tác tử những sự kiện đặc biệt, hỏi một cái gì đó của tử. Bởi sự trự trị của mình nên các tác tử có thể khởi tọ sự tương tác và hồi đáp thông điệp theo mọi cách chúng muốn.

Ở một phương diện khác, chúng ta có thể nhận thấy hướng đối tượng có thể giúp chúng ta giải quyết các vấn đề của hướng tác tử. Chắc hẳn chúng ta trước khi biết tới lập trình hướng đối tượng cũng đều biết qua về lập trình hướng thủ tục, chia chương trình thành các hàm con hoặc chia hệ thống thành các modun cho dễ quản lí, bảo trì, … tất cả những gì lập trình hướng đối tượng làm được đều có thể làm được bằng lập trình hướng thủ tục. Thậm chí, các ngôn ngữ lập trình như C hoặc Cobol có thể sử dụng để lập trình hướng đối tượng. Vậy tại sao chúng ta không sử dụng ngôn ngữ lập trình C mà phải cần đến các ngôn ngữ như C++ hay Java. Câu trả lời ở việc chúng ta sử dụng những gì chúng ta biết để mang lại một cách nghĩ khác về hệ thống và cài đặt chúng. Tác tử chính là một sự tiến hóa từ hướng đối tượng.

### I.1.4. Phân loại tác tử

Việc phân loại các tác tử dựa trên 3 yếu tố: Tự trị, học hỏi và hợp tác. [7]

Hình 2: Mô hình phân loại tác tử

*Tác tử giao diện*: là tác tử thi hành nhiệm vụ của nó chú trọng vào tính tự trị và tinhs học hỏi. Chúng hỗ trợ và cung cấp sự hỗ trợ cho một người dùng học sử dụng một ứng dụng cụ thể như bảng tính. Các tác tử quan sát các hành động thực hiện bởi người dùng và cố học những cái mới một cách ngắn gọn sau đó cố gắng đề xuất cách tốt hơn khi làm cùng một công việc.

*Tác tử hợp tác:* Mục đích của tác tử hợp tác là liên kết các tác tử riêng rẽ lại với nhau vì thế tạo ra các quần thể tác tử có khả năng hoạt động tốt hơn là các tác tử riêng rẽ.

*Tác tử thông tin*: Tác tử này xuất hiện do sự bùng nổ thông tin trên Word Wide Web. Mục đích của loại tác tử này là giúp chúng ta quản lí, thao tác hoặc đối chiếu các thông tin từ nhiều nguồn phân tán. Kiểu tác tử này có một chút tương đồng với tác tử giao diện, tuy nhiên một điểm cần chú ý là tất cả các loại tác tử này đều bắt đầu tại cùng một thời điểm. Mục đích của tác tử thông tin là định nghĩa những gì chúng làm, còn mục đích của tác tử giao diện là định nghĩa chúng là cái gì.

*Tác tử phản ứng*: là tác tử thực hiện và phản hồi lại môi trường khi có sự kích thích phản ứng nào đó xuất hiện.

*Tác tử lai:* là những tác tử mà sự cấu tạo của nó là sự kết hợp của hai hoặc nhiều triết lí tác tử. Những triết lí này có thể là di động, giao diện, thông tin, hợp tác, …

*Tác tử di động*: Một tác tử phần mềm là một tác tử phần mềm di động nếu nó có thể di chuyển từ máy tính này sang máy tính khác trong một môi trường mạng không đồng nhất.

## I.2. Hệ đa tác tử (Multi Agents System – MAS)

### I.2.1. Khái niệm

Hệ đa tác tử là một tập hợp các tác tử tương tác với nhau bằng cách trao đổi thông điệp. Trong trường hợp tổng quát nhất, các tác tử trong hệ đa tác tử sẽ thay thế cho người dùng hoặc người sở hữu để thực hiện những mục tiêu rất khác nhau. Chính vì thế để các tác tử có thể tương tác với nhau thì đòi hỏi các tác tử cần có khả năng hợp tác, phối hợp và thương lượng với nhau theo cách mà con người vẫn hợp tác, phối hợp, thương lượng với nhau trong cuộc sống hàng ngày. [8]

Dựa trên khái niệm về hệ đa tác tử, chúng ta có thể dễ dàng thấy được ưu điểm của hệ đa tác tử chính là sức mạnh và khả năng mở rộng. Sự mạnh mẽ thể hiện ở chỗ việc điều khiển và trách nhiệm được chia đều cho các tác tử trong hệ thống. Nếu một hay nhiều tác tử bị hư hỏng thì hệ thống vẫn có thể xử lí được. Khả năng mở rộng của hệ đa tác tử bắt nguồn từ sự modun hóa của nó. Chúng ta có thể dễ dàng thêm các tác tử mới vào trong hệ thống thay vì là thêm chức năng cho hệ thống. [9]

Hình 3: Mô hình hệ đa tác tử

quan hệ tổ chức

Môi trường

Phạm vi ảnh hưởng

sự tương tác

Tác tử

### I.2.2. Vấn đề phối hợp của các tác tử trong hệ đa tác tử

Một trong những vấn đề chính đối với hệ đa tác tử là làm thế nào để các tác tử phối hợp với nhau một cách hiệu quả. Bởi vì trong hệ đa tác tử, các tác tử giải quyết các công việc nhỏ, nhưng các công việc nhỏ này lại phụ thuộc lẫn nhau, sự phụ thuộc thể hiện ở việc cạnh tranh tài nguyên, hoặc sự liên quan giữa các công việc mà từng tác tử đảm nhiệm. Mối quan hệ giữa các tác tử phát sinh từ hai tình huống cơ bản liên quan đến việc chia công việc mà cả hệ đa tác tử cần giải quyết thành các công việc nhỏ được giải quyết bởi từng tác tử. Tình huống thứ nhất là các công việc nhỏ có thể giống hệt nhau hoặc trùng lặp với nhau nhưng các tác tử khác nhau lại lựa chọn cách thức hoặc dữ liệu khác nhau để giải quyết. Ví dụ một ứng dụng đánh giá tình huống phân tán, các công việc nhỏ có thể trùng lặp khi mà các tác tử khác nhau phân tích dữ liệu từ các cảm biến khác nhau từ các nguồn tài nguyên độc lập bởi có sự trùng lặp giữa các vùng cảm biến. Tình huống thứ hai là khi cách thực hiện các công việc nhỏ có những ràng buộc nhất định. Ví dụ trong một ứng dụng hệ chuyên gia phân tán liên quan đến việc thiết kế của một cổ vật, mỗi tác tử phải chịu trách nhiệm thiết kế một bộ phận khác nhau, có những ràng buộc khi thiết kế các bộ phận này mà các tác tử phải tuân thủ bởi nếu các tác tử thực hiện một cách độc lập thì các thành phần này không thể ghép lại với thành sản phẩm tổng thể được. Tùy theo tính chất sự phụ giữa các công việc nhỏ, sự tương tác giữa các tác tử trong hệ đa tác tử có thể phức tạp và thường đòi hỏi một cuộc đối thoại nhiều bước. Ví dụ một công việc pj rất khó để một tác tử có thể giải quyết nếu như không có một tác tử khác giải quyết công việc pi trước đó hoặc việc giải quyết công việc pi có thể giúp tác tử giải quyết công việc pj. Những kiểu tương tác như vậy đã được minh họa trong các hệ thống đa tác tử thương mại được mở ra gần đây để phục hồi dịch vụ của mạng lưới giao thông điện tử liên quan đến các tác tử cho việc phát hiện lỗi, chuẩn đoán, cô lập lỗi, và cấu hình mạng. Chúng ta xem xét ví dụ về hai tác tử chuyên gia trong hệ thống thực hiện các dạng chuẩn đoán lỗi khác nhau. Mỗi tác tử hoạt động đồng thời, sử dụng cac thuật toán khác nhau để chuẩn đoán và thông tin mà mỗi tác tử sử dụng cũng không giống nhau. Cả hai tác tử có thể mắc sai lầm nhưng nhìn chung những sai lầm này sẽ không giống nhau. Hai tác tử sẽ tương tác với nhau bằng cách trao đổi một phần kết quả để tập trung vào quá trình tìm kiếm chuẩn đoán địa phương hướng tới những khu vực triển vọng của mạng lưới giao thông nơi là nguồn gốc lỗi xuất hiện và rời bỏ những nơi không phải là nguồn gốc xuất hiện lỗi. Chúng cũng trao đổi kết quả cuối cùng để làm tăng sự tin tưởng trong kết quả chuẩn đoán cuối cùng xem chúng có đồng ý với kết quả chuẩn đoán hay không. Nếu như các tác tử này không đồng ý thì sẽ có một sự tương tác phức tạp hơn được bảo đảm (có thể là sự đàm phán) để tìm ra nguyên nhân của sự bất đồng và sau đó đưa ra một chuẩn đoán mới dựa trên sự nhau này. Như vậy, nhờ sự phối hợp với nhau, các tác tử không chỉ đưa ra giải pháp mới có chất lượng tốt hơn mà còn hoàn thành công việc nhanh hơn. [13]

### I.2.3. Đánh giá số nền tảng đa tác tử [6]

Ngày nay, có rất nhiều nền tảng đa tác tử hỗ trợ xây dựng các hệ thống đa tác tử phức tạp. Tùy từng yêu cầu khác nhau của hệ thống mà lựa chọn các nền tảng cho phù hợp. Sau đây, em xin giới thiệu về một số nền tảng đa tác tử:

**JADE** là một nền tảng phần mềm đơn giản hóa quá trình triển khai hệ thống đa tác tử thông qua một tầng giữa (middle ware) và tuân thủ đặc tả của FIPA. Nền tảng tác tử JADE được lập trình bằng ngôn ngữ Java nhằm mục đích cao hiệu năng của hệ thống tác tử phân tán.

**JATLite** được phát triển bởi Stanford Center for Design Research là một công cụ để tạo các hệ thống dựa trên tác tử. JATLite được thiết kế cho phép người dùng tạo các tác tử phần mềm mới có thể giao tiếp mạnh mẽ thông qua Internet.

**SkeletonAgent** được phát triển bởi “Systems Complex and Adaptive Laboratory”, là một công cụ phần mềm. Các khái niệm về tác tử được gói gọn trong một tập các thư viện tái sử dụng. Kiến trúc của AkeletonAgent được thiết kế cho phép việc tích hợp và hợp tác.

**ZEUS**: được phát triển bởi “BT Laboratories in the Advanded Application & Technolory Department” là một mô hình tiện lợi phát triển các ứng dụng đa tác tử một cách nhanh chóng.

Rất khó để có thể so sánh các nền tảng đa tác tử ở trên, bởi vì mỗi một nền tảng đều có những ưu và nhược điểm khác nhau. Để so sánh, chúng ta dựa trên hai phương diện là việc xây dựng phần mềm (building software) và trong khi chạy nền tảng (runtime).

**Trên phương diện xây dựng phần mềm**: Gồm các yếu tố sau:

*Thời gian phát triển*: bao gồm nỗ lực bỏ ra để thiết kế và lập trình tác tử đầu tiên và cho toàn bộ hệ đa tác tử. Đối với tác tử đầu tiên, những giai đoạn sau sẽ được cân nhắc:

* Phân tích và thiết kế: Có một vài thông số được đưa ra để đánh giá giai đoạn này như số giờ người kĩ sư bỏ ra để phân tích và thiết kế hệ đa tác tử như mong muốn đối với từng framework, thời gian cần thiết để để tìm hiểu các framework khác có liên quan như các API lập trình, …
* Sinh mã nguồn cho tác tử đầu tiên: Là thời gian cần thiết để người kĩ sư cài đặt một tác tử mới. Công sức phải bỏ ra khi tích hợp với các hệ thống khác
* Sự tích hợp với mã nguồn bên ngoài: Nỗ lực bỏ ra để tích hợp với mã nguồn đã có từ trước của tác tử. Do đó, thời gian yêu cầu để chỉnh sửa các tác tử tổng quát được cung cấp bởi framework để xây dựng một tác tử cụ thể nên được lo lường.
* Kiểm thử và tìm lỗi: Thời gian cần để kiểm thử và tìm các lỗi cho những vấn đề có thể xuất hiện cho đến khi hệ thống làm việc một các đúng đắn (số lượng tác tử tối thiểu trong tập các tác tử cần thiết để làm việc)

*Tính sử dụng lại của phần mềm*: Số dòng mã nguồn mà các lớp Java mới hoặc số lượng lớp được sử dụng để xây dụng tác tử mới trong hệ thống.

**Trên phương diện khi hoạt động**:

* Đánh giá hiệu năng: Thông số này sử dụng để kiểm tra xem một hệ đa tac tử đơn giản với số lượng tác tử ít nhất mà hệ thống có thể hoạt động một cách chính xác. Một vài thông số để đánh giá tiêu chí này như: Thời gian mà người dùng phải chờ đợi đến khi hệ đa tác tử kết thúc tác vụ của nó. Điều này có thể phụ thuộc vào sự phức tạp của tác vụ mà tác tử phải làm. Hoặc số lượng các thông điệp trao đổi giữa các tác tử trong khi chạy.
* Khả năng mở rộng: Số lượng các tác tử sẽ tăng lên khi cài đặt một hệ đa tác tử để đánh giá khả năng mở rộng của hệ thống như thế nào.

Từ những thông số đã đưa ra chúng ta có bảng so sánh các framework đa tác tử như sau:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | JADE (hours) | Zeus (hours) | JATLite | Skeleton |
| Phân tích và thiết kế | 28 | 63 | 25 | **10** |
| Sinh mã nguồn (Với tác tử đầu tiên) | **12.5** | **12.5** | 25 | 30 |
| Tích hợp mã nguồn (Với tác tử đầu tiên) | 12.5 | 14.5 | 15 | 30 |
| Kiểm thử và tìm lỗi | **6** | 7 | 10 | 15 |
| Sinh mã nguồn (Đối với hệ đa tác tử) | 2.5 | **2** | 3 | 3.5 |
| Tích hợp mã nguồn (Đối với hệ đa tác tử) | **2** | **2** | **2** | 2.5 |
| Dòng mã nguồn | 1897 | **1354** | 3199 | 1447.5 |
| Lớp mới (new class) | **9** | 13 | 11 | 20 |
| Lớp sử dụng lại (reused class) | **11** | **11** | 13 | 14 |

Bảng 1: Kết quả thực nghiệm đối với các thông số

Bảng 1 đưa ra sự đánh giá trên phương diện xây dựng phần mềm. Đơn vị cho các tiêu chí như phân tích và thiết kế, sinh mã nguồn, tích hợp mã nguồn, kiểm thử và tìm lỗi là số giờ. Với các tiêu chí như dòng mã nguồn thì đơn vị là số dòng, với tiêu chí lớp mới thì đơn vị là số lớp mới được sinh ra. Cuối cùng tiêu chí lớp sử dụng lại chính là số lớp được sử dụng lại. Dựa vào bảng 1, chúng ta thấy JADE và ZEUS cho kết quả tốt nhất.

Bảng 2 ghi lại kết quả của việc đánh giá trên phương diện khi hoạt động thông qua việc so sánh: thời gian yêu cầu cho mỗi thông điệp. JADE và Skeleton cho kết quả tốt nhất

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Số tác tử | JADE (thời gian yêu cầu/thông điệp) | JATLite (thời gian yêu cầu/thông điệp) | Skeleton (thời gian yêu cầu/thông điệp) | Zeus (thời gian yêu cầu/thông điệp) |
| 1 | 113.8/35.2 | 255.1/38.9 | 136.3/35.7 | 146.7/39.3 |
| 2 | 378.7/77/3 | 468/75.4 | 312.8/65.2 | 383.6/74.8 |
| 3 | 659.6/108.4 | 946.3/113 | 412.6/98 | 716.7/144.2 |
| 6 | 1323.9/185.1 | 1093.1/253.3 | 989.7/190 | 1272.2/203.8 |

Bảng 2: Kết quả đánh giá trên phương diện khi hoạt động

Bảng 3 là kết quả thời gian hồi đáp thu được khi thực hiện cùng một truy vấn cho tất cả các tác tử

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Số tác tử | JADE | JATLite | Skeleton | Zeus |
| 1 | **113.8** | 225.1 | 136.3 | 146.7 |
| 6 | 1323.9 | 1093.1 | **989.7** | 1272.2 |
| 10 | **879.67** | 1126 | 881.34 | 898.7 |
| 20 | 1607 | **1544.7** | 1558.2 | 1585.7 |
| 30 | 2603.7 | 1762.3 | **1689.6** | 1763.3 |

Bảng 3: Kết quả so sánh thời gian yêu cầu

### I.2.4. Ứng dụng của hệ đa tác tử

Một vài ứng dụng phổ biến của hệ đa tác tử trong thực tế như:

* Bộ điều khiển máy bay Orbital Communications Adaptor (OCA) trong trung tâm điều khiển trạm vũ trụ quôc tế của NASA đã sử dụng các hệ thống máy tính khác nhau để uplink (giao tiếp từ mặt đất đến một vệ tinh), downlink (giao tiếp từ một vệ tinh đến mặt đất), sao lưu, lưu trữ và chuyển dữ liệu đến Trạm không gian quốc tế (International Space Station – ISS) hoặc chuyển dữ liệu từ Trạm không gian quốc tế đến nơi khác trong thời gian thực. [10]
* Ứng dụng đa tác tử trong lĩnh vực an ninh mạng. [11]
* Ứng dụng hệ đa tác tử trong thu thập thông tin, hợp tác giữa các tác tử di động, thông tin và giao tiếp giữa các cảm ứng (sensor). [12]

## I.3. JADE framework

### I.3.1. Giới thiệu

Telecom Italia (viết tắt là CSELT) đã phát triển những phần mềm đầu tiên, cuối cùng trở thành nền tảng JADE vào cuối năm 1998. Với quan điểm là để cung cấp các dịch vụ cho người phát triển ứng dụng và để dễ dàng sử dụng được và truy cập được cho cả những người phát triển lâu năm và người mới có ít hoặc không có chút kiến thức nào về những đặc tả của FIPA, JADE đặc biệt nhấn mạnh vào sự đơn giản và tiện dụng của các phần mềm API.[3]

JADE đã trở thành mã nguồn mở từ năm 2000 và được phân phối bởi Telecom Italia, đảm bảo tất cả các quyền cơ bản để tạo thuận lợi cho việc sử dụng phần mềm có trong các sản phầm thương mại: quyền làm bản sao của phần mềm và phân phối các bản sao, quyền được truy cập mã nguồn, và quyền được thay đổi mã và thực hiện các cải tiến của nó.

JADE có một website, http://jade.tilab.com, từ đó các phần mềm, tài liệu, mã nguồn ví dụ, và rất nhiều thông tin về cách sử dụng của JADE đều có sẵn. Dự án hoan nghênh sự tham gia của cộng đồng mã nguồn mở với nhiều cách thức để tham gia và đóng góp cho dự án, chúng đều được chi tiết hóa trên trang web.

Khi JADE lần đầu tiên được công bố bởi Telecom Italia, nó đã được sử dụng hầu như chỉ bởi cộng đồng FIPA nhưng khi tích hợp các chức năng lại vượt xa các chi tiết kỹ thuật FIPA. Do đó nó đã được sử dụng bởi một cộng đồng các nhà phát triển được phân phối trên toàn cầu.

Một trong những phần mở rộng của lõi JADE được cung cấp bởi LEAP, một dự án tài trợ một phần bởi Ủy ban châu Âu đã góp phần đáng kể từ năm 2000 và 2002 nhằm hướng JADE tới Java Micro Edition và môi trường mạng không dây. Công việc này được dẫn dắt bởi Giovanni Caire. Ngày nay, nó được dùng như một JADE run-time cho các nền tảng J2ME-CLDC và J2ME-CDC, và nó được sử dụng để giải quyết các vấn đề và thách thức đặt ra trong viễn thông di động, đây được coi là một trong những tính năng hàng đầu của JADE.

### I.3.2. Kiến trúc JADE



Hình 4: Mối quan hệ giữa các thành phần kiến trúc chính trong JADE

Hình 2 mô tả các thành phần kiến trúc chính trong nền tảng JADE. Một nền tảng JADE bao gồm các vùng chứa tác tử (agents container) được phân tán qua mạng. Các tác tử sống trong các vùng chứa, các vùng chứa là các tiến trình Java để chạy JADE và tất cả các dịch vụ cần cho việc lưu trữ và thực thi tác tử. Trong số các vùng chứa thì có một vùng chứa đặc biệt là vùng chứa chính (main container) đây là nơi khởi động của nền tảng JADE. Đây là khung chứa đầu tiên được chạy, tất cả các khung chứa khác phải đăng kí với khung chứa chính. Trong khung chứa chính có một thành phần là bảng khung chứa (Container table – CT) chính là nơi mà đăng kí các đối tượng giao chiếu và địa chỉ truyền thông của tất cả các nút khung chứa trong nền tảng.

Bảng mô tả tác tử toàn cụ (Global agent descriptor table - GADT) là nơi đăng kí của tất các tác tử trong khung chứa, các thông tin đăng kí bao gồm trạng thái hiện tại và vị trí của tác tử.

ASM và DF là hai tác tử đặc biệt cung cấp việc quản lí tác tử, các dịch vụ trang vàng, trang trắng.



Hình 5: Mô hình UML mô tả mối quan hệ giữa các thành phần kiến trúc chính

### I.3.2. Các thành phần JADE

JADE gồm các gói sau:

* *jade.core* là các lõi của hệ thống. Nó bao gồm lớp *Agent*, đây là lớp cơ bản mà tất các lớp phải kế thừa khi lập trình ứng dụng. Bên cạnh đó, lớp *Behaviour* được chứa trong các gói con *jade.core.behaviours*. Lớp *Behaviour* thực hiện nhiệm vụ, hoặc ý định của một agent. Những nhiệm vụ hoặc ý định là các đơn vị hoạt động một lgic có thể được tạo theo những cách khác nhau để thực hiện các nhiệm vụ phức tạp và có thể được thực hiện đồng thời. Người lập trình ứng dụng cần xác định hành vi của các tác tử bằng cách lập trình các hành vi và sự thực tương tác của các hành vi này giữa các tác tử.
* Gói con *jade.lang.acl* được cung cấp để xử lý ngôn ngữ giao tiếp tác tử (Agent Communication Language) dựa theo chuẩn FIPA.
* Gói *jade.content* chứa một tập các lớp để hỗ trợ người dùng định nghĩa các bản thể (ontologies) và nội dung ngôn ngữ. Một hướng dẫn riêng biệt mô tả làm thế nào để sử dụng sự JADE để hỗ nội dung thông tiệp. Trong *jade.content.lang.sl* các SL codec, gồm bộ phân tích cú pháp và các bộ mã hóa.
* Gói *jade.domain* chứa tất cả những lớp Java mà đại diện cho các tác tử quản lý được định nghĩa bởi chuẩn FIPA, cụ thể là các tác tử AMS và DF. Tác tử ASM và DF cung cấp chu kỳ sống, các dịch vụ trang trắng và trang vàng. Các gói con *jade.domain.FIPAAgentManagement* chứa FIPA-Agent-Management Ontology và tất cả các lớp đại diện cho các khái niệm của nó. Các gói con *jade.domain.JADEAgentManagement* chứa các phần mở rộng của JADE cho việc quản lí tác tử (Agent – Management) (ví dụ như đánh hơi tin nhắn, kiểm soát vòng đời của các agent ...), bao gồm cả bản thể và tất cả các lớp đại diện cho các khái niệm của nó. Các gói con *jade.domain.introspection* chứa các khái niệm được sử dụng cho việc đàm thoại giữa các công cụ JADE (ví dụ như Sniffer và Introspector) và hạt nhân JADE. Các gói *jade.domain.mobility* phụ có chứa tất cả các khái niệm sử dụng để giao tiếp về tính di động (mobility).
* Các gói *jade.gui* chứa một bộ các lớp tổng quát hữu ích để tạo GUI để hiển thị và sửa đổi định danh tác tử (Agent-Identifiers), các mô tả tác tử (Agent Descriptions), các thông điệp ngôn ngữ giao tiếp giữa các tác tử (ACLMessages), …

Các gói *jade.mtp* chứa một giao diện (interface) Java mà mỗi giao thức chuyển thông điệp nên thực hiện để có thể được dễ dàng tích hợp với nền tảng JADE, và thực hiện một tập hợp các giao thức này.

Gói *jade.proto* là gói chứa các lớp để mô hình các giao thức tương tác chuẩn (Ví dụ: Các giao thức theo chuẩn FIPA như fipa-request, fipa-query, fipa-contract-net, fipa-subscribe, ...) và các lớp để giúp người lập trình ứng dụng tạo ra các giao thức của riêng mình.

Các gói FIPA chứa các module IDL xác định bởi FIPA cho IIOP dựa trên việc truyền tin nhắn.

Cuối cùng, gói *jade.wrapper* cung cấp bao đóng cho các chức năng mức độ cao của JADE cho phép người sử dụng JADE như một thư viện, các ứng dụng Java bên ngoài khởi động các tác tử JADE và các vùng chứa tác tử (container agents)

**AMS**: đại diện quyền lực trong một flatform, là agent duy nhất có thể thực hiện hoạt động quản lý bắt đầu, tạm đừng hoặc kết thúc toàn bộ flatform, (agent bình thường có thể yêu cầu hoạt động như vậy tới AMS).

**DF**: cung cấp dịch vụ Page Yellow, nơi các agent có thể tạo ra các dịch vụ nó cung cấp và tìm các agent khác cung cấp các dịch vụ nó cần.

### I.3.2. Sự giao tiếp giữa các tác tử trong JADE

Sự giao của tác tử có lẽ là tính năng cơ bản nhất của JADE và được cài đặt tuân theo các thông số kĩ thuật FIPA. Mô hình truyền thông dựa trên việc truyền thông điệp không đồng bộ (Asynchronous Message Passing). Do đó, mỗi tác tử có một hộp thư hay gọi chính xác hơn là hàng đợi tin nhắn tác tử, nơi JADE trong khi chạy đã đẩy thông điệp được gửi bởi các tác tử khác vào. Bất cứ khi nào một thông điệp được gửi vào trong hộp thư, các tác tử nhận sẽ được thông báo. Tuy nhiên, khi nào hoặc nếu, các tác tử lấy các thông điệp từ hàng đợi để xử lí thì còn phụ thuộc sự lựa chọn thiết kế của người lập trình. Quá trình này được mô tả trong hình 6.

JADE chạy phân tán

Chuẩn bị thông điệp gửi đến tác tử 2

Lấy thông điệp từ hàng đợi thông điệp và xử lí nó

Đẩy thông điệp vào hàng đợi thông điệp của tác tử 2

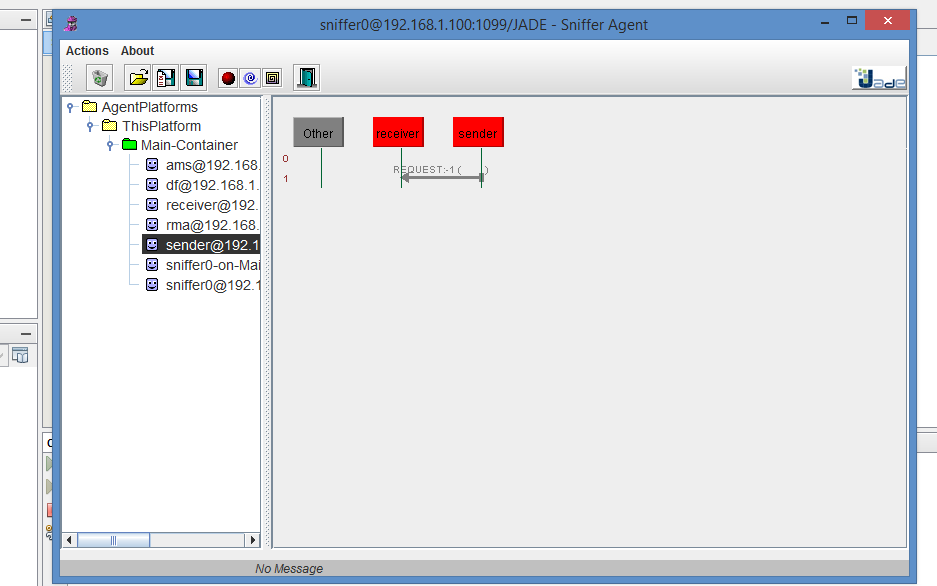
Hình 6: Mô hình truyền thông điệp không đồng bộ JADE

Các định dạng đặc biệt của thông điệp trong JADE phải phù hợp với định nghĩa của cấu trúc tin nhắn FIPA – ACL. Mỗi thông điệp bao gồm các trường sau:

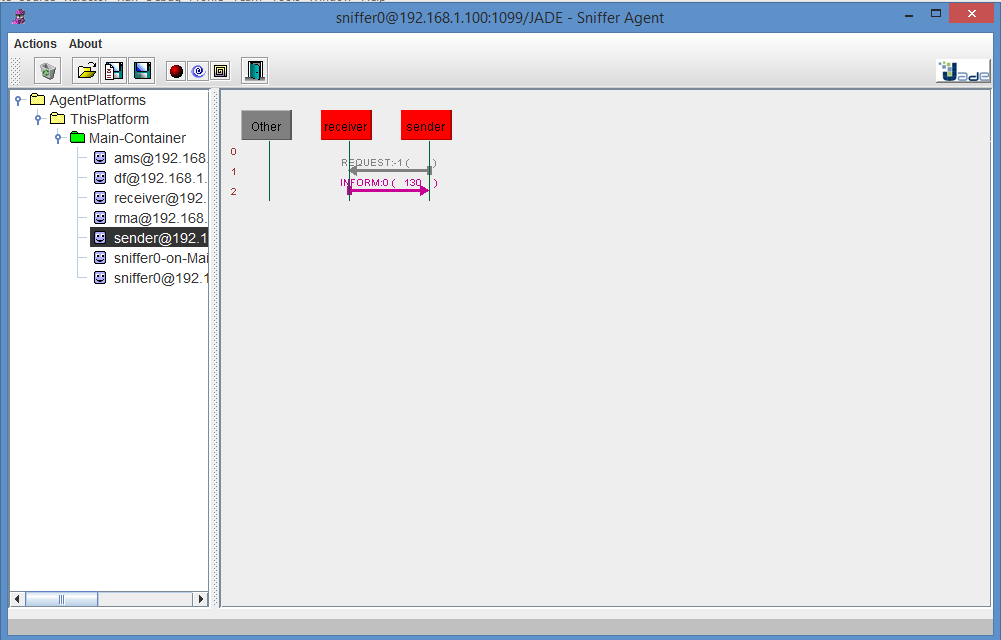
* Tác tử gửi thông điệp
* Danh sách các tác tử nhận.
* Các cách thức giao tiếp (hay còn gọi là biểu hiện - performative) cho thấy ý định của phía gửi bằng cách gửi tin nhắn. Ví dụ, nếu các biểu hiện là yêu cầu (REQUEST), phía gửi muốn người nhận để thực hiện một hành động, nếu nó là thông báo (INFORM) người gửi muốn người nhận phải nhận thức được một thực tế, nếu nó là một đề xuất (PROPOSE) hay một gọi đề xuất (Call for proposal – CFP) thì phía gửi muốn tham gia vào một cuộc đàm phán.
* Các nội dung có chứa các thông tin thực tế được trao đổi bằng thông điẹp (Ví dụ: hành động được thực hiện trong một thông điệp REQUEST, hoặc sự việc mà người gửi muốn tiết lộ trong một thông điệp INFORM, …)
* Ngôn ngữ thể hiện nội dung cho thấy cú pháp sử dụng để thể hiện nội dung. Cả người gửi và người nhận phải có khả năng mã hóa và phân tích cách diễn tả phù hợp với cú pháp này để các giao tiếp có hiệu quả.
* Bản thể (ontology) chỉ ra ý nghĩa của những kí hiệu được sử dụng trong nội dung. Cả người gửi và người nhận phải mô tả cùng một ý nghĩa với các kí hiệu để các giao tiếp có hiệu quả.
* Một số trường bổ sung được sử dụng để kiểm soát một số cuộc hội thoại song song và để xác định thời gian chờ để nhận được một câu trả lời như là định danh cuộc hội thoại (conversation-id), hồi đáp – với (reply-with), hồi đáp cho và hồi đáp bởi (in-reply-to and reply-by).

Một tin nhắn trong JADE được thực hiện như một đối tượng của lớp jade.lang.acl.ACLMessage cung cấp và thiết lập các phương pháp để truy cập tất cả các trường theo quy định của các định dạng ACL. Tất cả biểu hiện định nghĩa trong đặc tả FIPA được ánh xạ như hằng số trong các lớp ACLMessage.

Quá trình gửi và nhận thông điệp được minh họa như hình 6 và hình 7. Ở hình 6, phía gửi gửi một thông điệp dạng “REQUEST” cho phía nhận. Ở hình 7, phía gửi gửi một tin nhắn dạng “REQUEST” cho phía nhận, sau đó phía nhận gửi lại một tin nhắn dạng nhắc nhở “INFORM” cho phía gửi.



Hình 7: Minh họa quá trình phía gửi gửi thông điệp cho phía nhận

  
Hình 8: Minh họa việc phía gửi gửi tin nhắn đến phía nhận và đồng thời phía nhận gửi tin nhắn trả lời cho phía gửi

#### I.3.2.1. Gửi thông điệp

Việc gửi thư tới tác tử khác cũng đơn giản như việc điền vào các trường của một đối tượng ACLMessage và sau đó gọi phương thức send() của lớp Agent. Đoạn mã dưới đây tạo ra một thông điệp dạng INFORM để báo cho một tác tử có biệt danh là Peter với nôi dung “Today it’s raining”:

ACLMessage msg = new ACLMessage(ACLMessage.INFORM);

msg.addReceiver(new AID("Peter", AID.ISLOCALNAME));

msg.setLanguage("English");

msg.setOntology("Weather-forecast-ontology");

msg.setContent("Today it’s raining");

send(msg);

#### I.3.2.2. Nhận thông điệp

JADE trong khi chạy tự động đưa các thông điệp vào hàng đợi tin nhắn của tác tử nhận ngay khi thông điệp đến. Một tác tử có thể lấy thông điệp từ hàng đợi thông điệp của mình bằng phương thức receive(). Phương pháp này trả về thông điệp đầu tiên trong hàng đợi thông điệp, hoặc không có gì nếu hàng đợi thông điệp trống rỗng một cách lập tức.

ACLMessage msg = receive();

if (msg != null) {

// Xử lí thông điệp

}

#### I.3.2.3. Lưa chọn thông điệp từ hàng đợi thông điệp

Khi thông điệp được gửi đến hàng đợi thông điệp tác tử có thể không nhất thiết lựa chọn thông điệp đầu tiên trong hàng đợi mà tác tử lựa chọn dạng thông điệp mà nó muốn nhận. Muốn vậy, phía gửi gói thông điệp mà nó muốn gửi đến phía nhận trong một khuôn (template) thông qua phương thức:

MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.ACCEPT\_PROPOSAL);

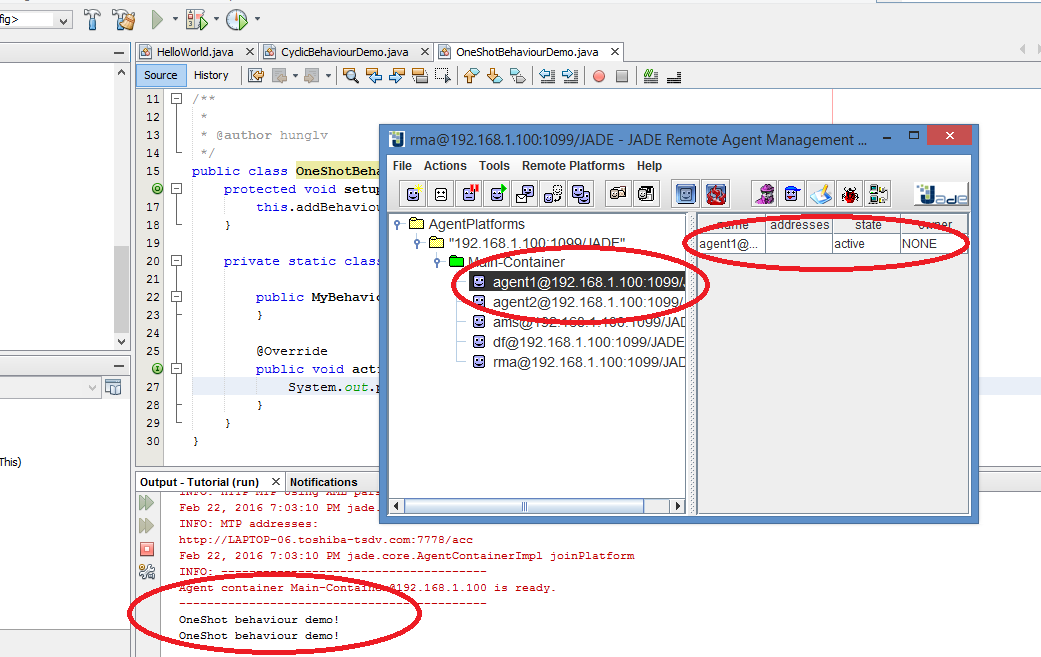
ACLMessage.ACCEPT\_PROPOSAL là việc quy định dạng thông điệp muốn gửi đi, phương thức MessageTemplate.MatchPerformative() đưa thông điệp muốn gửi đi vào một cái khuôn. Khi thông điệp đến hàng đợi của phía gửi, phía gửi sẽ lựa chọn từ hàng đợi thông điệp cái thông điệp đầu tiên có dạng ACLMessage.ACCEPT\_PROPOSAL.

### I**.3.3. Các hành vi của tác tử**

Hành vi là một công việc mà tác tử có thể thực thi. Các tác tử có rất nhiều loại hành vi, các hành vi của mỗi tác tử đều có đặc điểm chung là đều thực hiện hai phương thức chính là phương thức action() và phương thức done(), các hành vi của tác tử đều được cài đặt như một đối tượng của lớp kế thừa từ lớp *jade.core.behaviours.Behaviour*. Tuy nhiên, trong nội dung của môn học em xin trình bày bốn loại hành vi chính hay được sử dụng là: Hành vi thực hiện một lần (OneShotBehaviour), hành vi thực hiện mãi mãi (CyclicBehavior), hành vi đánh thức (WakerBehaviour), hành vi thực hiện có chu kì (TickerBehavior).

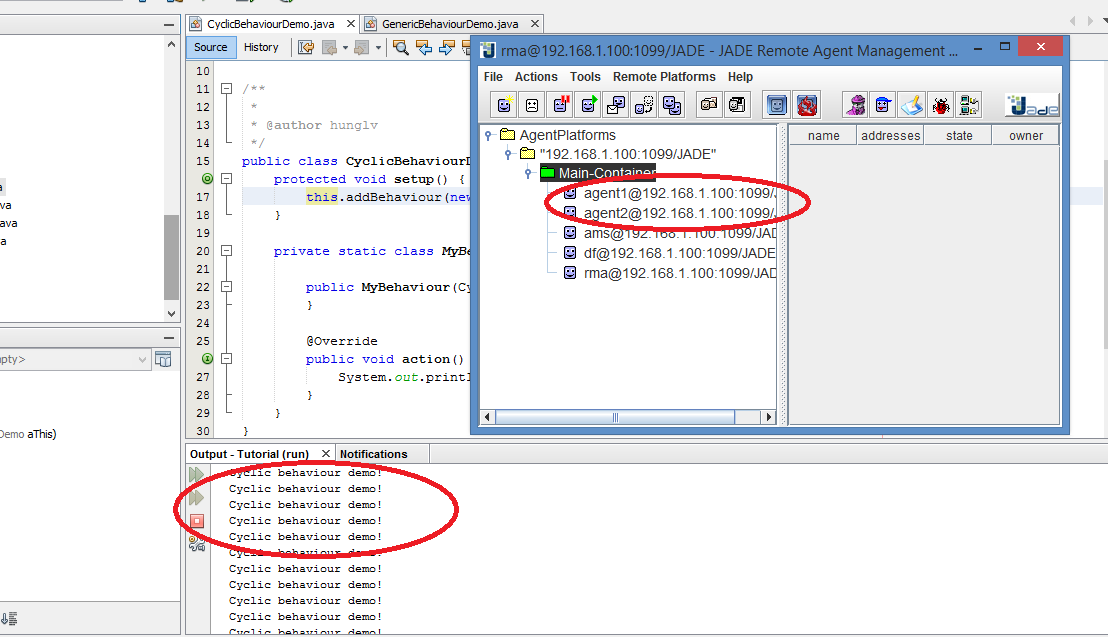
Hành vi thực hiện một lần (OneShotBehaviour): Đây là kiểu hành vi chỉ thực hiện một lần duy nhất nên phương thức action() chỉ thực hiện duy nhất 1 lần. Vì thế giá trị trả về của phương thức done() mặc định là true. Kiểu hành vi này thuộc lớp jade.core.behaviours.OneShotBehaviour.

Ví dụ về kiểu hành vi thực hiện một lần được trình bày ở hình 9:



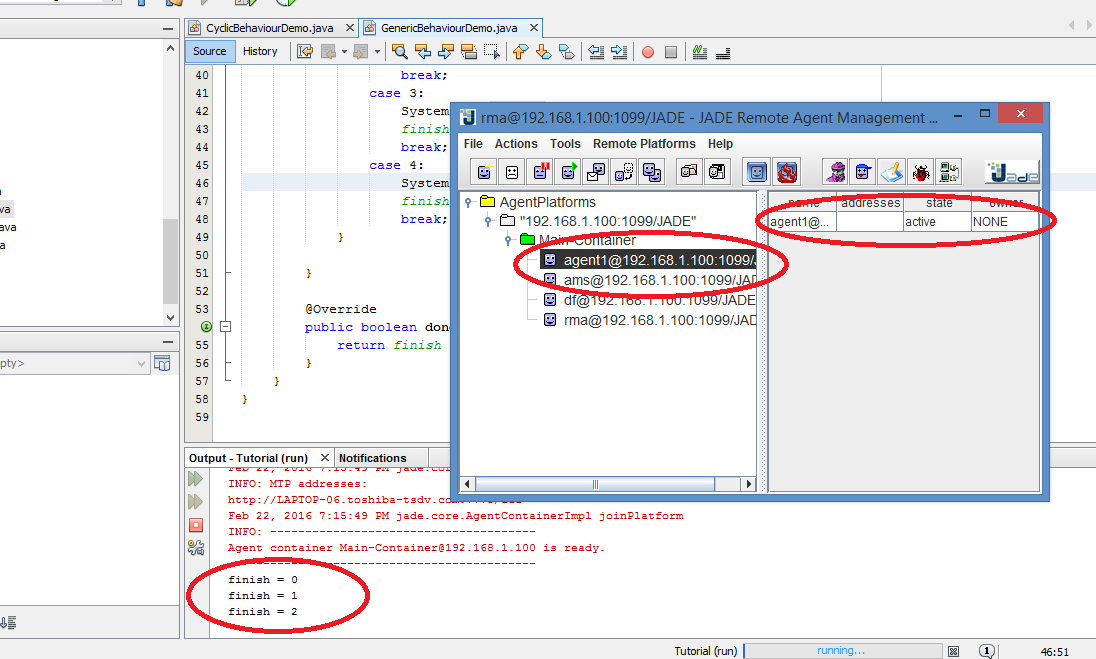
Hình 9: Ví dụ về kiểu hành vi thực hiện một lần

Hành vi thực hiện mãi mãi (CyclicBehaviour): Đây là kiểu hành vi thực hiện mãi mãi, vì thế phương thức action() thực thi cùng một hành vi một cách liên tục. Vì thế kết quả trả về của phương thức done() luôn luôn là false. Kiểu hành vi này thuộc lớp jade.core.behaviours.CyclicBehaviour. Ví dụ về kiểu hành vi thực hiện mãi mãi được mô tả như ở hình 10.



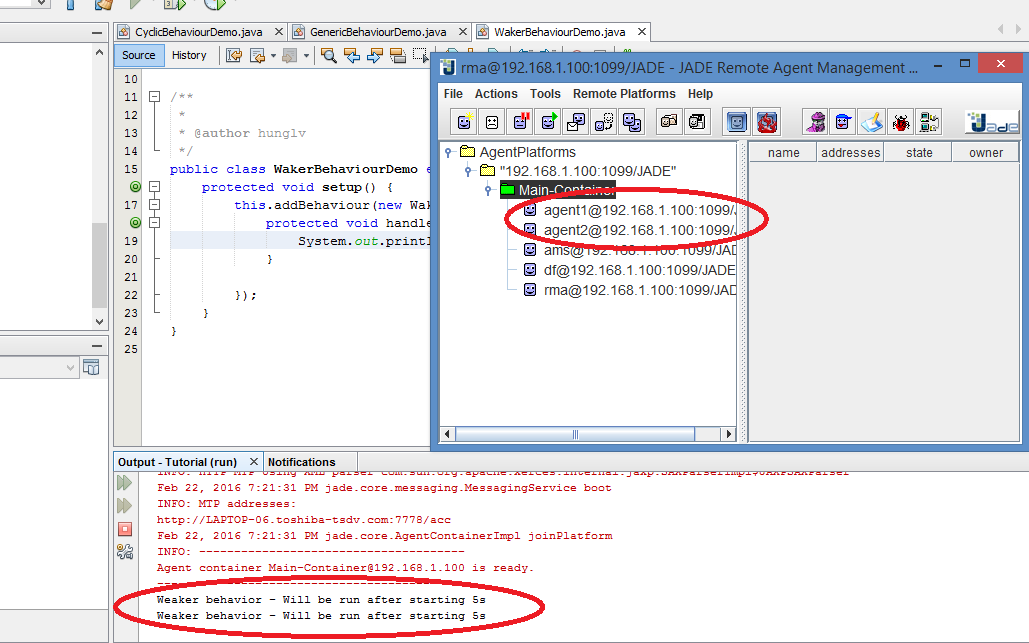
Hình 10: Ví dụ kiểu hành vi thực hiện mãi mãi

Hành vi tổng quát (GenericBehaviour): Đây là loại hành vi gắn trạng thái kích hoạt hành động và thực thi các hành động khác nhau phụ phụ thuộc vào trạng thái của giá trị. Kiểu hành vi tổng quát này sẽ kết thúc khi thỏa mãn một điều kiện nào đó mà chúng ta định nghĩa trong phương thức done(). Các hành động trong phương thức action() sẽ được thực thi phụ thuộc vào giá trị của biến trạng thái. Hình 11 là ví dụ minh họa về kiểu hành vi tổng quát.



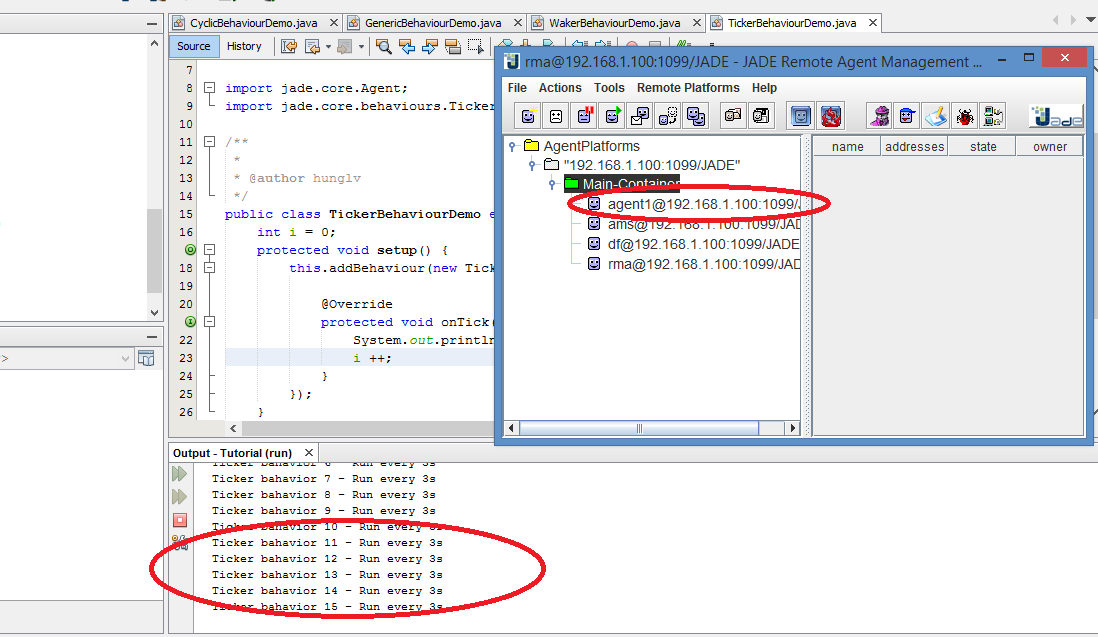
Hình 11: Ví dụ về kiểu hành vi tổng quát

Hành vi đánh thức (WakeBehaviour): Đây là hành vi được thực thi duy nhất một lần sau một khoảng thời gian xác đinh vì thế phương thức action() và phương thức done() đã thực sự được cài đặt vì thế phương thức handleElapsedTimeout() được cài đặt như một lớp con và được thực thi sau một khoảng thời gian xác định (dưới dạng ms). Kiểu hành vi này thuộc lớp jade.core.behaviours.WakerBehaviour. Ví dụ về kiểu hành đánh thức được mô tả ở hình 12.



Hình 12: Ví dụ về kiểu hành vi đánh thức

Hành vi thực hiện có chu kì (TickerBehaviour): Đây là loại hành vi thực hiện mãi mãi trừ phi phương thức done() được thực thi. Phương thức action() và done() đã được cài đặt sẵn vì vậy phương thức onTick() được thực hiện định kì với một khoảng thời gian nhất định (đặc tả dưới dạng ms). Kiểu hành vi này thuộc lớp jade.core.behaviours.TickerBehaviour. Ví dụ về kiểu hành vi thực hiện có chu kì được mô tả ở hình 13.



Hình 13: Ví dụ về kiểu hành vi thực hiện có chu kì

### I.4. Dịch vụ trang vàng (Yellow page service) trong JADE

#### I.4.1. Tác tử DF (Directory Facilitator)

**Dịch vụ trang vàng**

**Yellow pages service**

Tác tử 1: - Dịch vụ X

- Dịch vụ Y

Tác tử 2: - Dịch vụ Z

Tác tử 3: - Dịch vụ W

- Dịch vụ K

- Dịch vụ H

Công bố dịch vụ mà tác tử cung cấp

Tìm kiếm dịch vụ được yêu cầu

Hình 14: Dịch vụ trang vàng

Một dịch vụ trang vàng cho phép các tác tử công bố mô tả của một hay nhiều dịch vụ mà chúng cung cấp để các tác tử khác có thể dễ dàng tìm kiếm và sử dụng chúng. Hình 9, mô tả một cái nhìn khái quát về dịch vụ trang vàng, các tác tử 1, tác tử 2, tác tử 3 là những tác tử cung cấp dịch vụ, chúng sẽ công bố dịch vụ mà chúng cung cấp lên dịch vụ trang vàng, các tác tử 4, tác tử 5, tác tử 6 là những tác tử cần sử dụng dịch vụ, chúng sẽ tìm kiếm trên trang dịch vụ trang vàng loại dịch vụ mà chúng cần để sử dụng. Mọi tác tử có thể vừa tìm kiếm các dịch vụ và vừa có thể sử dụng các dịch vụ. Đăng ký, hủy đăng ký, thay đổi và tìm kiếm các dịch vụ có thể được thực hiện bất cứ lúc nào trong suốt thời gian sống của tác tử. Dịch vụ trang vàng trong JADE, phù hợp với đặc tả quản lí tác tử FIPA (FIPA Agent Management), được cung cấp bởi một tác tử đặc biệt gọi là DF (Directory Facilitator). Mỗi nền tảng tuân thủ chuẩn FIPA nên chưa một tác tử DF mặc định (tên gọi là “df @ <platform-name>”). Các tác tử DF khác có thể được triển khai nếu cần thiết và một số tác tử DF (bao gồm cả các mặc định) có thể liên lại với nhau để cung cấp một danh mục dịch vụ trang vàng phân tán.

#### I.4.2. Tương tác với tác tử DF

Vì DF là một tác tử, nó có thể tương tác như với bất kỳ tác tử nào khác bằng cách trao đổi các tin nhắn ACL sử dụng một ngôn ngữ nội dung thích hợp (ví dụ như ngôn ngữ SL0) và một bản thể phù hợp (ví dụ bản thể FIPA-agent-management) theo như định nghĩa của chuẩn FIPA. Để đơn giản hóa các tương tác, JADE cung cấp lớp jade.domain.Service mà nó có thể công bố và tìm kiếm các dịch vụ thông qua việc gọi một số phương thức.

**Công bố dịch vụ:** Một tác tử muốn công bố một hoặc nhiều dịch vụ phải cung cấp DF với một mô tả bao gồm AID riêng của mình, một danh sách các dịch vụ được cung cấp và tùy chọn là danh sách ngôn ngữ và bản thể (ontologies) mà các tác tử khác phải sử dụng để tương tác với nó. Mỗi mô tả dịch vụ được cung cấp phải bao gồm kiểu dịch vụ, tên dịch vụ, các ngôn ngữ và bản thể yêu cầu sử dụng dịch vụ và một tập hợp các thuộc tính dịch vụ cụ thể dưới hình thức cặp khóa-giá trị. Các lớp DFAgentDescription, ServiceDescription và Property đều là các lớp trừu tượng, được định nghĩa trong gói jade.domain.FIPAAgent.

Để cung cấp một dịch vụ tác tử phải tạo ra một mô tả thích hợp (một thực thể của lớp DFAgentDescription) và gọi phương thức tĩnh register() của lớp DFService. Ví dụ:

protected void setup() {  
...  
// Register the book-selling service in the yellow pages  
DFAgentDescription dfd = new DFAgentDescription();  
dfd.setName(getAID());  
ServiceDescription sd = new ServiceDescription();  
sd.setType("Book-selling");  
sd.setName(getLocalName()+"-Book-selling");  
dfd.addServices(sd);  
try {  
DFService.register(this, dfd);  
}  
catch (FIPAException fe) {  
fe.printStackTrace();  
}  
...  
}

Khi kết thúc một tác tử nên hủy bỏ đăng kí sử dụng dịch vụ với dịch vụ trang vàng.

protected void takeDown() {  
// Deregister from the yellow pages  
{  
DFService.deregister(this);  
}  
catch (FIPAException fe) {  
fe.printStackTrace();  
}  
...  
}

**Tìm kiếm dịch vụ**: Một tác tử có nhu cầu tìm kiếm các dịch vụ phải cung cấp cho tác tử DF với một mô tả mẫu (template description). Kết quả của việc tìm kiếm là một danh sách của tất cả các mô tả phù hợp với các mẫu đã cung cấp. Theo đặc tả của FIPA, một mô tả phù hợp với mẫu nếu tất cả các trường được đặc tả trong mẫu có cùng giá trị với tất cả các trường trong mô tả dịch vụ. Ví dụ:

public class BookBuyerAgent extends Agent {  
// The list of known seller agents  
private Vector sellerAgents;  
protected void setup() {  
...  
// Update the list of seller agents every minute  
addBehaviour(new TickerBehaviour(this, 60000) {  
protected void onTick() {  
// Update the list of seller agents  
**DFAgentDescription template = new DFAgentDescription();  
ServiceDescription sd = new ServiceDescription();  
sd.setType("Book-selling");  
template.addServices(sd);  
try {  
DFAgentDescription[] result = DFService.search(myAgent,  
template);**  
sellerAgents.clear();  
for (int i = 0; i < result.length; ++i) {  
sellerAgents.addElement(result[i].getName());  
}  
}  
catch (FIPAException fe) {  
fe.printStackTrace();  
}  
}  
} );  
...

Trong ví dụ trên chúng ta thấy việc tìm kiếm được lặp đi lặp lại một lần mỗi phút kể từ khi tác tử bán tự động có thể xuất hiện và biến mất trong hệ thống.

# CHƯƠNG II: PHÂN TÍCH THIẾT KẾ PHẦN MỀM

### II.1. Yêu cầu bài toán

Dường như đối với mỗi chúng ta, những người dân chuyên về công nghệ, khái niệm nhà thông minh đã không còn xa lạ một chút nào. Nhà thông minh (tiếng Anh là "Smart Home") hoặc hệ thống nhà thông minh là một ngôi nhà/ căn hộ được trang bị hệ thống tự động tiên tiến dành cho điều khiển đèn chiếu sáng, nhiệt độ, truyền thông đa phương tiện, an ninh, rèm cửa, cửa và nhiều tính năng khác nhằm mục đích làm cho cuộc sống ngày càng tiện nghi, an toàn và góp phần sử dụng hợp lý các nguồn tài nguyên. Trong thời gian nghiên cứu về hệ đa tác tử, dựa trên những ưu điểm, các tính năng nổi trội mà hệ đa tác tử mang lại, nhóm chúng em đã nghĩ đến việc mô hình một chiếc nhà thông minh bởi hệ đa tác tử. Các tác tử sẽ đóng vai trò như những thiết bị trong nhà, điều phối hoạt động dựa theo thông tin về môi trường mà tác tử cảm biến cung cấp.

### II.2. Kiến trúc

Hệ thống sẽ gồm 3 thành phần chính:

* Tác tử cảm biến (Đóng vai trò như một thiết bị cảm biến, thu nhận các thông tin về môi trường và gửi cho các thiết bị trong nhà. Vì chương trình demo chỉ mang tính chất mô phỏng nhà thông minh nên tác tử cảm biến sẽ sinh ra các số ngẫu nhiên đại diện cho các thông tin của môi trường.
* Tác tử thiết bị: Gồm nhiều thiết bị điện trong gia đình. Vì ở mức độ chỉ mô phỏng một chiếc nhà thông minh nên nhóm chúng em sẽ minh họa bằng 6 thiết bị chính gồm: Điều hòa, quạt, bóng đèn, ti vi, tủ lạnh, máy giặt.
* Tác tử quản lí: Đây là một tác tử đóng vai trò quan sát hiện trạng của căn nhà, hiển thị lên một giao diện để cho người dùng biết về trạng thái của căn nhà.

Mô hình kiến trúc nhà thông minh được minh họa ở hình 15.

Hình 15: Kiến trúc nhà thông minh mô phỏng

Tác tử cảm biến

Điều hòa

Ti vi

Tủ lạnh

Bóng đèn

Quạt

Tác tử quản lí

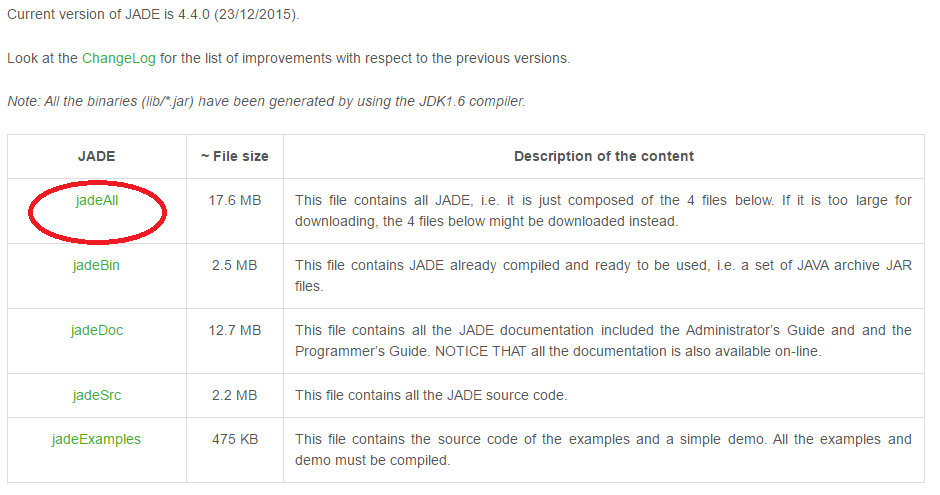
Máy giặt

### II.3. Mô hình ca sử dụng của hệ thống

# CHƯƠNG III: CÀI ĐẶT VÀ TRIỂN KHAI

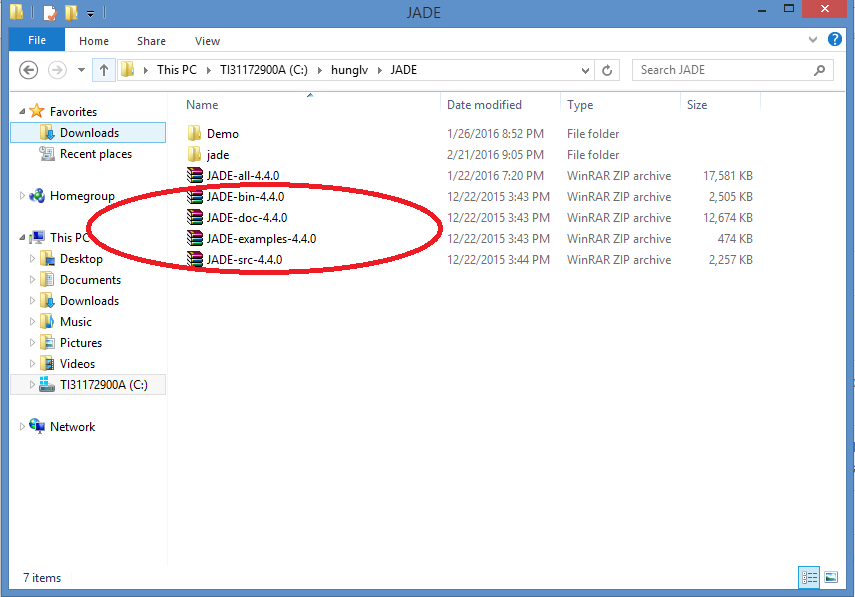
## III.1 Cài đặt JADE

Trước tiên, chúng ta tải framework JADE tại trang chủ sau: <http://jade.tilab.com/>.



Hình 16: Minh họa tải framework JADE tại trang chủ

Sau khi tải xong JADE, chúng ta tiến hành giải nén. Cấu trúc thư mục JADE sau khi giải nén được minh họa ở mình 17.

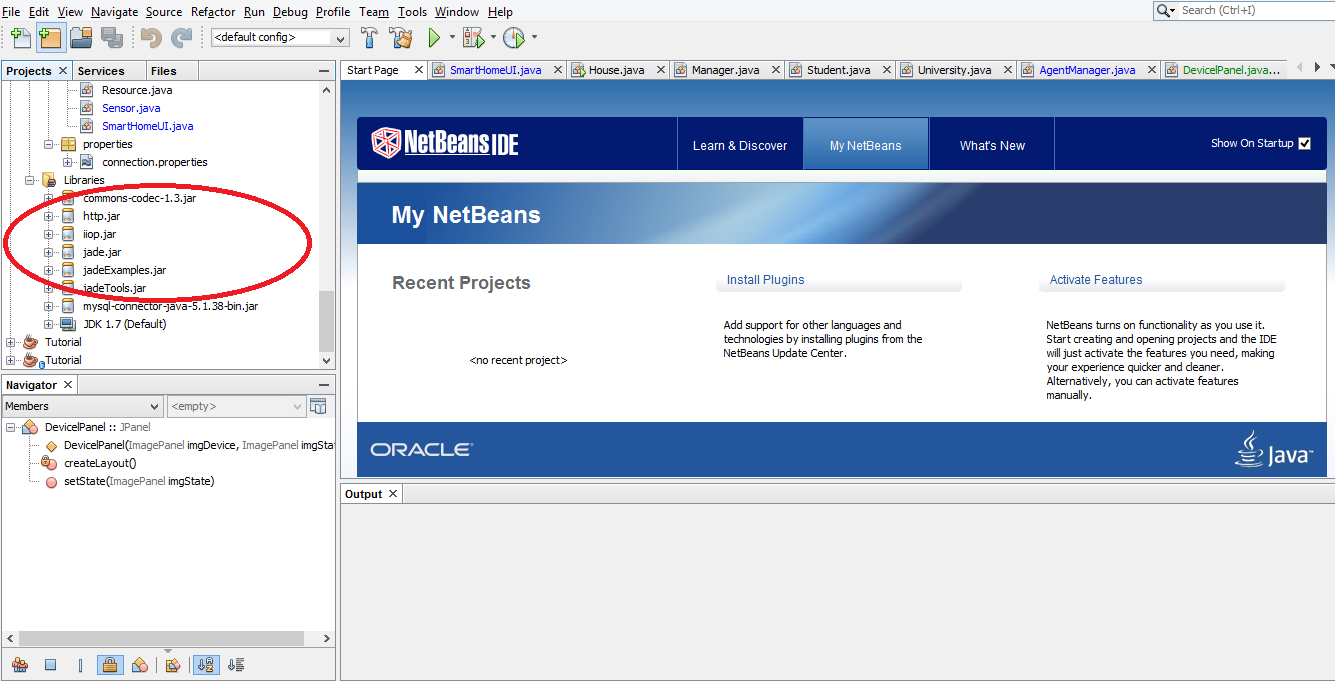


Hình 17: Cấu trúc thư mục JADE sau khi giải nén

JADE gồm 4 thư mục con

* jadeBin: Chứa JADE đã được biên dịch và sẵn sàng sử dụng
* jadeDoc: Chứa các tài liệu JADE bao gồm tài liệu hướng dẫn quản trị và hướng dẫn lập trình.
* jadeSrc: Chứa tất cả mã nguồn của JADE
* jadeExamples: Chứa mã nguồn các chương trình ví dụ và demo đơn giản.

Để lập trình với JADE, IDE mà nhóm chúng em chọn là NetBean, để NetBean có thể lập trình được JADE chúng ta thực hiện như sau: Khởi động Netbean và tạo một project mới, thêm các thư viện sau vào phần “Set Project Configuration” trong của project vừa tạo: http.jar, iiop.jar, jade.jar, jadeTools.jar, jadeExamples.jar, commons-codec-1.3.jar.



Hình 18: Mô tả cách thêm các thư viện JADE để cấu hình một project



Hình : Minh họa cách cấu hình để chạy một project JADE

Cấu cấu hình để chạy một hệ đa tác tử:

* Main class: **jade.Boot**
* Arguments: **“-gui ‘Tên tác tử’:’Lớp thực thi tác tử’.**

Ví dụ: “-gui agent1:behaviour.TickerBehaviourDemo”



Hình : Mô tả cách cấu hình các tham số trong JADE để chạy một chương trình

## III.2. Lập trình JADE

Khi tạo một lớp tác tử mới phải kế thừa từ lớp Agent của lớp *jade.core.Agent,* phần mã chương trình sẽ được viết trong phương thức setup() và chúng ta cần lưu ý các bước cấu hình ở phần III.1 đều đã được thực hiện.

Vì JADE được viết bằng ngôn ngữ Java nên cú pháp lập trình đều giống với cú pháp của Java, chúng ta có thể tài liệu hướng dẫn và mã nguồn ví dụ trong tệp JADE đã tải về từ trang chủ.

## III.3. Mô phỏng một hệ thống nhà thông minh



Hình 21: Giao diện chương trình khi hoạt động

Hình 21 là giao diện chương trình khi hoạt động, chúng ta thấy hình ảnh của 6 thiết bị đúng như đã đề cập trong mục II.1. Dưới mỗi hình ảnh là một nút trạng thái minh họa việc thiết bị hoạt động (mở) hay không hoạt động (tắt). Mục đích của chương trình cho thấy cách các tác tử tương tác với nhau để giúp hệ thống nhà thông minh hoạt đông.

# CHƯƠNG 4: KẾT LUẬN

Dựa trên những kiến thức đã tìm hiểu được về tác tử, hệ đa tác tử và JADE framework đã giúp chúng em có được một hướng đi mới về xây dựng hệ thống. Khác với trước kia, việc xây dựng hệ thống chủ yếu dựa trên

**TÀI LIỆU THAM KHẢO**

[1]Nicholas R. Jennings and Michael Wooldridge, “Agent-Oriented Software Engineering”, Department of Electronic Engineering Queen Mary & Westfield College University of London London E1 4NS, United Kingdom, pp 5.

[2] Michael Wooldridge, “Agent-Based Software Engineering”, Mitsubishi Electric Digital Library Group, 18th Floor, Centre Point, 103 New Oxford Street London WC1A 1EB, United Kingdom. September 19, 1997.

[3] John Wiley&Sons, “Developing Multi-Agent Systems with JADE”, John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex PO19 8SQ, England. Copyright 2007.

[4] Giovanni Caire , 2009. Jade programing for beginers.pdf. TILAB, formerly CSELT

[5] Loạt bài hướng dẫn lập trình hướng agent

<https://www.youtube.com/watch?v=VHt9A9Jn1Jk&list=PL6_53ptjaqs4zs-xSK7SwabqC-CjubpIV>

[6] David Camacho and Ricardo Aler, “Software and performance measures for evaluating mutlti-agent frameworks, Universidad Carlos III de Marid, Computer Science Department, Avenida de la.

[7] Qusay H. Mahmoud, “Software Agents: Characteristics and Classification”, School of Computing Science, Simon Fraser University, Burnaby, BC. Canada V5A 1S6.

[8] Michael Wooldridge, “An Introduction to MultiAgent Systems”, John Wiley and Sons, 2002

[9] Dr. Mevludin Glavic, “Agents and Multi-Agent Systems: A Short Introduction for Power Engineers”, University of Liege Electrical Engineering and Computer Science Department”. May, 2006, pp. 8

[10] Maarten Sierhuis, William J. Clancey, Ron J.J. van Hoof and Brian L. Anderson, “NASA’s OCA Mirroring System An application of multiagent systems in Mission Control”. NASA Ames Research Center.

[11] Munindar P. Singh, “Cybersecurity as an Application Domain for Multiagent Systems”. Department of Computer Science North Carolina State University Raleigh, NC 27695-8206, USA

[12] Jonas Buchli, “Mobile Robots Moving Intelligence”, Pro Literatur Verlag, pp 376

[13] Victor R. Lesser, “Cooperative Multiagent Systems: A Personal View of the State of the Art”. Computer Science Department University of Massachusetts at Amherst