MỤC LỤC

DANH MỤC HÌNH VẼ

MỞ ĐẦU

Trong lĩnh vực công nghệ phần mềm có nhiều phương pháp tiếp cận để xây dựng phần mềm. Trong đó, xây dựng phần mềm dựa trên tác tử là hướng tiếp cận mới và đem lại nhiều lợi ích, đặc biệt trong một số ứng dụng chuyên biệt.

Xuất phát từ yêu cầu thực tế đó em đã chọn đề tài "Kiến trúc phần mềm dựa trên tác tử ".

Đồ án bao gồm 3 chương :

Chương 1: Tác tử và đa tác tử.

Giới thiệu tổng quan kiến thức về tác tử và đa tác tử.

Chương 2: Nền tảng JADE.

Trong chương này đồ án trình bày những đặc điểm cơ bản kiến trúc JADE và các yếu tố liên quan.

Chương 3: Kiến trúc phần mềm dựa trên tác tử và ứng dụng.

Ứng dụng. và thực ngiệm.

CHƯƠNG 1: TÁC TỬ VÀ ĐA TÁC TỬ

1.1. Tác tử và hệ đa tác tử

1.1.1. Giới thiệu về tác tử và hệ đa tác tử

Chương này trước hết giới thiệu các khái niệm về tác tử [1] [2] [3], 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 [1] [7] - 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 [1] [8] 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.

1.1.2. Định nghĩa về tác tử

Thuật ngữ *"tác tử"* 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.

Mặc dù cho đến nay vẫn chưa có một định nghĩa thống nhất về tác tử nhưng tất cả các định nghĩa đều có chung một điểm rằng một tác tử, về bản chất, là một phần mềm máy tính đặc biệt có thể tự chủ và cung cấp một interface có khả năng tương thích với một hệ thống bất kì và/hoặc cư xử như là một tác tử con người hay đại diện cho một số client để thực thi các đích cho riêng mình.

Mặc dù một đa tác tử có thể chỉ cần dựa trên một tác tử đơn lẻ để làm việc trong một môi trường và tương tác với người dùng của nó khi cần thiết, tuy nhiên các đa tác tử thường bao gồm nhiều tác tử. Những hệ thống đa tác tử (MAS: Multiagent System) có thể sử dụng để mô hình hóa các hệ thống phức tạp bao gồm các tác tử với các mục tiêu chung hoặc riêng. Những tác tử có thể tương tác với nhau một cách gián tiếp (qua tác động lên môi trường) hoặc trực tiếp (thông qua giao tiếp và thương lượng). Các tác tử có thể quyết định hợp tác để cùng có lợi hoặc có thể cạnh tranh để phục vụ cho mục tiêu của mình.

- Như vậy, tác tử có tính tự chủ, vì nó hoạt động mà không có sự can thiệp trực tiếp của con người hoặc các hệ thống khác và có khả năng kiểm soát được hành động và trạng thái bên trong của mình.

- Tác tử có tính xã hội, vì nó tương tác với con người hoặc các tác tử khác để hoàn thành nhiệm vụ của mình.
- Tác tử có tính phản ứng, bởi vì nó nhận thức được môi trường và đáp ứng một cách kịp thời với những thay đổi xảy ra trong môi trường.
- Tác tử có tính hướng đích, vì nó không chỉ đơn giản là hoạt động để phản ứng với môi trường của nó mà còn có khả năng thể hiện hoạt động hướng đích một cách chủ động.
- Tác tử có thể có tính di động, với khả năng di chuyển giữa các node trong một mạng máy tính.
- Tác tử có thể có tính trung thực nghĩa là luôn cung cấp sự thật.
- Tác tử có thể tốt bụng, luôn cố gắng thực hiện những gì được yêu cầu.
- Tác tử có thể sáng suốt, luôn hoạt động hướng đến để đạt được mục tiêu và không bao giờ ngăn cản việc đạt được mục tiêu của mình.

1.1.3. Các kiểu kiến trúc của tác tử.

Kiến trúc tác tử là cơ chế nằm bên dưới các thành phần tự chủ nhằm hỗ trợ hành vi của tác tử trong thế giới thực, môi trường động và môi trường mở. Trong thực tế, những nỗ lực ban đầu trong lĩnh vực tính toán dựa trên tác tử tập trung vào sự phát triển của các kiến trúc tác tử thông minh và đã đưa ra khá nhiều kiểu kiến trúc. Vì vậy, kiến trúc tác tử có thể được chia thành bốn chính nhóm: dựa trên logic, có tính phản ứng, BDI và phân lớp.

1.1.3.1. Những kiến trúc dựa trên logic

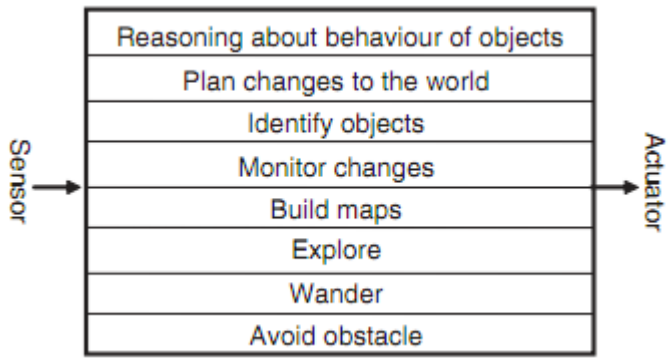
Những kiến trúc dựa trên logic (logic-based) lấy nền tảng từ kỹ thuật dựa trên tri thức truyền thống trong đó một môi trường được thể hiện và hoạt động bằng cách sử dụng các cơ chế lập luận. Ưu điểm của cách tiếp cận này là tri thức của con người được biểu diễn bởi các ký hiệu và vì thế mà việc mã hóa trở nên dễ dàng hơn và cũng làm cho con người hiểu logic hoạt động của nó dễ dàng hơn. Nhược điểm là rất khó để biên dịch thế giới thực thành những mô tả hình tượng một cách chính xác và đầy đủ. Hơn nữa việc biểu diễn và xử lý dưới dạng các kí hiệu có thể mất nhiều thời gian để có được kết quả và thường là được đưa ra quá muộn, không còn có ích nữa.

1.1.3.2. Phản ứng

Những kiến trúc có tính phản ứng (reactive) thực thi quá trình đưa ra quyết định khi ánh xạ trực tiếp tình huống sang hành động và được dựa trên một cơ chế kích thích - phản ứng được tạo ra bởi dữ liệu của thiết bị cảm biến. Không giống như những kiến trúc dựa trên logic, chúng không có bất kì mô hình biểu diễn tri thức và vì thế, không tận dụng được các kiểu lập luận phức tạp nào. Kiến trúc có tính phản ứng nổi tiếng nhất là kiến trúc gộp của Brooks [7]. Những ý tưởng chính mà dựa trên đó Brooks đã tìm ra kiến trúc này là:

- Một cách ứng xử thông minh có thể được tạo ra mà không cần biểu diễn rõ ràng và lập luận được cung cấp bởi các kỹ thuật của trí tuệ nhân tạo.
- Thông minh là một tính chất riêng biệt của những hệ thống phức tạp.

Kiến trúc gộp xác định các tầng của các máy hữu hạn trạng thái – các máy được kết nối với thiết bị cảm biến – các thiết bị truyền thông tin theo thời gian thực (một ví dụ của kiến trúc gộp được thể hiện trong hình 1.1). Các tầng này tạo thành sự phân cấp các hành vi của tác tử trong đó, mức độ thấp nhất được điều khiển ít hơn so với mức độ cao hơn trong ngăn xếp, vì thế việc ra quyết định được đưa ra thông qua những hành vi hướng đích. Những tác tử được thiết kế gộp hiểu được điều kiện và hành động, nhưng không đưa ra được kế hoạch.



Hình 1.1. Kiến trúc gộp

Điểm mạnh của phương pháp tiếp cận này là nó có thể thực thi tốt hơn trong những môi trường động, cũng như chúng thường được thiết kế đơn giản hơn so với những tác tử dựa trên logic.

Tuy nhiên, nhược điểm là những tác tử có khả năng phản ứng không áp dụng được khi những mô hình là kết quả tác động của môi trường của chúng. Do đó, các dữ liệu có thể không đủ để xác định một hành động thích hợp và thiếu các

trạng thái của tác tử khiến cho hầu như không thể thiết kế các tác tử có thể học hỏi từ kinh nghiệm.

1.1.3.3. BDI

Các kiến trúc BDI (Belief, desire, intention) [5] là những kiến trúc tác tử phổ biến nhất. Chúng có nguồn gốc triết học và dựa trên lý thuyết logic. Lý thuyết này dựa trên những quan điểm về tinh thần của niềm tin, mong muốn và dự định bằng cách sử dụng logic hình thức. Một trong những kiến trúc BDI nổi tiếng nhất là hệ thống lập luận theo thủ tục (PRS – Procedural Reasoning System). Kiến trúc này dựa trên 4 kiểu dữ liệu chính: Lòng tin (beliefs), tác vụ (desires), ý định (intentions) và kế hoạch (plans) và một bộ phận phiên dịch (xem hình 2.2). Trong hệ thống PRS, lòng tin biểu diễn những thông tin mà tác tử có về môi trường của nó, có thể không đầy đủ hoặc không chính xác. Tác vụ biểu diễn những tác vụ được phân công cho tác tử và tương ứng là những mục tiêu, hoặc là mục đích mà nó sẽ hoàn thành. Ý định thể hiện những mong muốn mà tác tử cần phải đạt được. Cuối cùng, kế hoạch chỉ rõ một vài quá trình của hành động mà tác tử sẽ phải làm để đạt được mục đích. Bốn cấu trúc dữ liệu này được quản lý bởi bộ phận phiên dịch tác tử chịu trách nhiệm cập nhật lòng tin từ những quan sát từ môi trường, sinh ra những tác vụ mới dựa trên cơ sở của các lòng tin mới, và lựa chọn trong tập những tác vụ hiện tại một vài tập con để hoạt động, chúng được gọi là ý định. Cuối cùng, bộ phận phiên dịch phải lựa chọn một hành động để thực thi dựa trên cơ sở của những ý định hiện tại của tác tử và tri trức về mặt thủ tục.

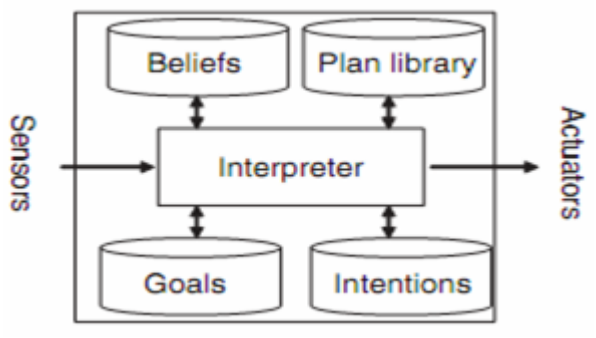
1.1.3.4. Kiến trúc phân lớp

Kiến trúc phân tầng (layered architecture) cho phép hành vi của tác tử vừa mang tính phản xạ vừa có tính kế hoạch. Để có được sự linh hoạt này, các hệ thống con được sắp xếp thành các tầng của một hệ thống phân cấp nhằm thích ứng với cả hai loại hành vi của tác tử.

Có hai loại luồng điều khiển trong một kiến trúc phân lớp: phân lớp ngang và phân lớp dọc.

Trong phân lớp nằm ngang, các lớp kết nối một cách trực tiếp với đầu vào của sensor và đầu ra của hành động về cơ bản là có mỗi tầng hoạt động giống như một tác tử. Điểm mạnh chính của cách phân lớp này là sự dễ dàng trong thiết kế bởi vì nếu tác tử t cần n loại hành vi khác nhau, thì kiến trúc chỉ yêu cầu n tầng. Tuy nhiên, bởi vì mỗi tầng đều bị ảnh hưởng bởi tác tử, nên không cần có một chức

năng trung gian hòa giải để kiểm soát các hành động. Sự phức tạp khác là một lượng lớn các tương tác có thể xảy ra giữa những tầng ngang - mn (với m là số lượng hành động tại mỗi tầng).



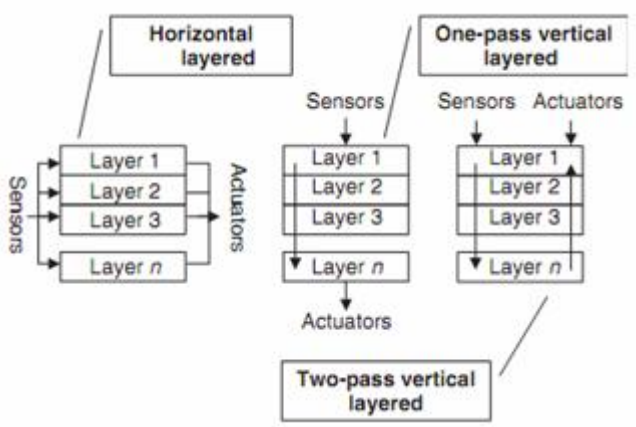
Hình 1.2. Kiến trúc PRS

Một kiến trúc phân lớp dọc loại trừ một số vấn đề trên vì đầu vào của sensor và đầu ra của hành động được giải quyết phần lớn tại mỗi tầng. Kiến trúc phân lớp dọc có thể được chia nhỏ thành những kiến trúc điều khiển một chiều và hai chiều. Trong kiến trúc một chiều, luồng điều khiển đi từ tầng đầu, tầng nhận dữ liệu từ các sensor, xuống đến tầng cuối, tầng sinh ra đầu ra của hành động (xem Hình 1.2). Trong kiến trúc hai chiều, luồng dữ liệu đi lên xuyên qua các tầng và điều khiển, tiếp đó lại có luồng dữ liệu trở về theo thứ tự ngược lại (xem hình 1.3). Điểm mạnh chủ yếu của kiến trúc phân lớp dọc là sự tương tác giữa các tầng được làm giảm đáng kể còn m2 (n-1). Nhược điểm là kiến trúc này phụ thuộc vào tất cả các tầng và không chấp nhận lỗi, vì thế nếu một tầng lỗi, toàn bộ hệ thống sẽ lỗi.

1.1.4. Giao tiếp và phối hợp

1.1.4.1. Giao tiếp

Một trong những thành phần chính của những hệ thống đa tác tử là giao tiếp. Trong thực tế, các tác tử cần có khả năng giao tiếp với người dùng, với tài nguyên hệ thống, và với tác tử khác nếu chúng cần hợp tác, cộng tác, đàm phán…Cụ thể, các tác tử tương tác với tác tử khác bằng cách sử dụng một vài ngôn ngữ giao tiếp đặc biệt, được gọi là những ngôn ngữ giao tiếp tác tử, dựa trên lý thuyết lời nói hành động và đem lại sự phân biệt giữa hành động giao tiếp và ngôn ngữ nội dung.



Hình 1.3. Luồng dữ liệu và luồng điều khiển trong kiến trúc phân lớp

Ngôn ngữ giao tiếp tác tử đầu tiên là KQML. KQML được phát triển vào đầu những năm 1990 là một phần của dự án ARPA của chính phủ Mỹ. Nó là một ngôn ngữ và giao thức để trao đổi thông tin và tri thức, xác định nhiều động từ biểu hiện và cho phép nội dung thông điệp được thể hiện trong một ngôn ngữ giống logic đầu tiên được gọi là KIF. Hiện nay, ngôn ngữ giao tiếp agent được nghiên cứu và sử dụng nhiều nhất là FIPA ACL, nó kết hợp nhiều khía cạnh của KQML. Đặc điểm chính của FIPA ACL là khả năng sử dụng những ngôn ngữ nội dung khác nhau và sự quản lý các cuộc hội thoại thông quan các giao thức tương tác được xác định trước.

1.1.4.2. Phối hợp

Phối hợp là một tiến trình mà trong đó, các tác tử tham gia nhằm đảm bảo rằng một cộng đồng các tác tử đơn lẻ hành động một cách chặt chẽ . Có khá nhiều lý do lý giải tại sao nhiều tác tử cần phối hợp với nhau:

(1) Các mục đích của các tác tử có thể gây ra sự xung đột giữa các hành động của tác tử.

(2) Các mục đích của các tác tử có thể phụ thuộc lẫn nhau.

(3) Các tác tử có thể có những khả năng và tri thức khác nhau.

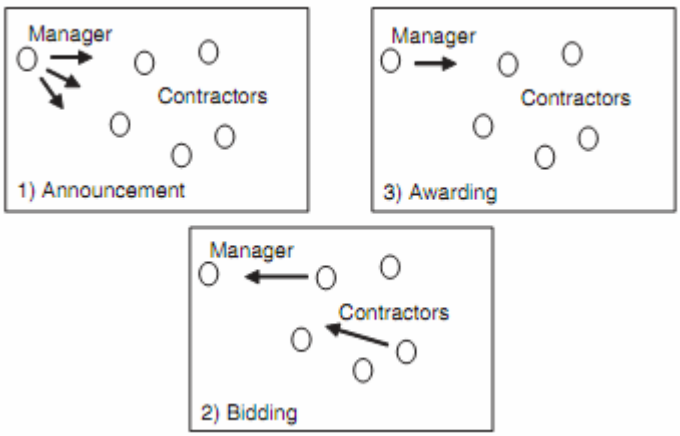
(4) Các mục đích của các tác tử có thể nhanh chóng đạt được nếu có sự cộng tác giữa các tác tử khác nhau.

Sự phối hợp giữa các tác tử có thể được điều khiển với nhiều phương pháp tiếp cận khác nhau bao gồm cơ cấu tổ chức (Organizational structuring), lập hợp đồng (contracting), lập kế hoạch và đàm phán.

Cơ cấu tổ chức cung cấp một nền tảng để hoạt động và tương tác thông qua việc định nghĩa các vai trò, đường truyền thông và các mối quan hệ về quyền hạn. Cách đơn giản nhất để đảm bảo hành vi rõ ràng và giải quyết xung đột là kết hợp một nhóm với một tác tử có một quan điểm rộng về hệ thống, qua đó tạo thành một cơ cấu tổ chức hoặc cấu trúc phân cấp.

Một kỹ thuật phối hợp quan trọng dùng cho việc phân bổ nhiệm vụ và phân bổ tài nguyên giữa các tác tử và xác định cơ cấu tổ chức là giao thức mạng hợp đồng (contract net protocol). Phương pháp tiếp cận này dựa trên một cơ cấu thị trường phân quyền, mà trong đó, các tác tử có thể đảm nhiệm hai vai trò, quản lý và đấu thầu. Những tiền đề cơ bản của thể thức phối hợp này là nếu một tác tử không thể giải quyết một vấn đề được giao khi chỉ sử dụng nguồn lực hoặc chuyên môn của mình, nó sẽ phân rã vấn đề thành các vấn đề con và cố gắng tìm các tác tử sẵn sàng khác với nguồn lực/chuyên môn cần thiết để giải quyết những vấn đề con này.

Một phương pháp tiếp cận khác coi vấn đề phối hợp các tác tử là vấn đề lập kế hoạch. Để ngăn chặn những hành động hoặc tương tác xung đột hay không phù hợp, các tác tử có thể xây dựng một kế hoạch chi tiết hóa toàn bộ những hành động và tương tác trong tương lai để đạt được mục đích và bổ sung thêm các kế hoạch hoặc lập lại kế hoạch. Lập kế hoạch đa tác tử có thể tập trung, hoặc là phân tán.



Hình 1.4. Các pha của giao thức mạng hợp đồng

Đàm phán có thể là một kỹ thuật đáng tin cậy nhất để phối hợp các tác tử. Cụ thể, đàm phán là quá trình giao tiếp của một nhóm các tác tử để đạt được một thỏa thuận chấp nhận lẫn nhau về một vấn đề nào đó. Đàm phán có thể mang tính cạnh tranh hoặc hợp tác tuỳ thuộc vào hành vi của các tác tử có liên quan.

1.1.5. Ngôn ngữ lập trình và công cụ

Ngôn ngữ lập trình, nền tảng và các công cụ phát triển của hệ thống đa tác tử là thành phần quan trọng mà có ảnh hưởng đến việc ứng dụng rộng rãi các công nghệ tác tử. Trong thực tế, sự thành công của hệ thống đa tác tử phần lớn là phụ thuộc vào sự sẵn có của công nghệ (tức là ngôn ngữ lập trình, thư viện phần mềm và các công cụ phát triển) để cho phép thực thi các khái niệm và các kỹ thuật đã hình thành cơ sở cho hệ thống đa tác tử.

Hệ thống tác tử có thể được cài đặt bằng cách sử dụng một loại ngôn ngữ lập trình nào đó. Cụ thể, ngôn ngữ hướng đối tượng được coi là một phương tiện phù hợp, vì khái niệm về tác tửkhông khác nhiều so với từ khái niệm đối tượng. Trong thực tế, các tác tử chia sẻ nhiều tính chất với các đối tượng như đóng gói (encapsulation), và đôi khi có cả kế thừa (inheritance) và truyền thông điệp (message passing). Tuy nhiên, các tác tử cũng khác với các đối tượng ở một số điểm chính: tính tự chủ (autonomous) (nghĩa là chúng có thể tự quyết thực hiện hay không thực hiện một hành động theo yêu cầu từ các tác tử khác); chúng có thể có hành vi linh hoạt; và mỗi tác tử của một hệ thống có thể điều khiển luồng của riêng mình.

Ngôn ngữ lập trình hướng tác tử là một loại ngôn ngữ lập trình mới. Nó tập trung vào những đặc điểm chính của hệ thống đa tác tử. Tối thiểu, một ngôn ngữ lập trình hướng tác tử phải bao gồm một vài cấu trúc tương ứng với một tác tử, nhưng nhiều ngôn ngữ lập trình cũng cung cấp các cơ chế để hỗ trợ các thuộc tính bổ sung của tác tử như niềm tin (beliefs), mục đích (goals), kế hoạch (plans), vai trò (roles) và quy tắc (norms).

Ngày nay, một số ngôn ngữ hướng tác tử đã xuất hiện. Một số được thiết kế từ đầu, trực tiếp mã hóa một số lý thuyết về tác tử, trong khi một số khác mở rộng ngôn ngữ đã có để phù hợp với tính chất riêng biệt của tác tử. Ngoài ra, một số ngôn ngữ mang quan điểm lập trình hoàn toàn có tính chất khai báo hoặc có tính chất bắt buộc. Ví dụ điển hình là FLUX và ngôn ngữ Jack Agent.

Nền tảng là phương tiện chính cho phép phát triển các hệ thống đa tác tử. Hầu hết chúng cung cấp một phương tiện để triển khai nhiều hệ thống tác tử trên các phần cứng và hệ điều hành khác nhau, thường là cung cấp một chương trình trung gian (midleware) để hỗ trợ thực thi và các hoạt động cần thiết của chúng như giao tiếp (communication) và phối hợp (coordination). Một số nền tảng có mục đích chung là cung cấp các chức năng theo các chuẩn FIPA để hỗ trợ cộng tác giữa nhiều

hệ thống tác tử khác nhau. Ngoài ra, một số nền tảng cũng có mục tiêu hỗ trợ các loại phần cứng, mạng truyền thông và kiến trúc tác tử, ví dụ như JADE và một số hỗ trợ các loại tác tử đặc biệt, ví dụ như các tác tử điện thoại di động.

Một đặc điểm quan trọng mà các hệ thống đa tác tử nên cung cấp là khả năng hỗ trợ sự tương tác giữa các hệ thống phần mềm kế thừa từ các hệ thống trước. Do đó, sự sẵn sàng tích hợp các công cụ phần mềm với các công nghệ khác có thể là chìa khóa dẫn đến thành công của chúng. Internet là một trong các lĩnh vực ứng dụng quan trọng nhất và là phương tiện truyền thông quan trọng nhất mà nhiều hệ thống đa tác tử có thể sử dụng để cung cấp khả năng tương tác giữa các hệ thống phần mềm kế thừa. Do vậy, rất nhiều công trình nghiên cứu và phát triển hiện nay hướng đến việc cung cấp các kỹ thuật và công cụ phần mềm thích hợp cho việc tích hợp các hệ thống đa tác tử với các công nghệ web như Web Service và Semantic Web.

1.1.6. Tác tử di động

1.1.6.1. Thế nào là tác tử di động

Theo các định nghĩa chuẩn, các tác tử di động [4] có tất cả đặc tính của một tác tử thông thường (như tính tự chủ, phản ứng, hướng đích và tính xã hội), nhưng thêm vào đó chúng có khả năng di chuyển – chúng có thể di chuyển giữa các platform để thực hiện các nhiệm vụ được giao. Từ quan điểm của hệ thống phân tán, một tác tử di động là một chương trình với một thực thể duy nhất, có thể di chuyển code, dữ liệu và trạng thái của nó giữa các máy đã được nối mạng. Để đạt được điều này, các tác tử di động có thể tạm dừng quá trình thực thi của chúng tại bất kỳ thời điểm nào và tiếp tục tại một vị trí khác. Chúng ta có thể đặt các tác tử di động trong mối quan hệ với các cách tiếp cận cổ điển khác:

- Client – server: phương pháp tiếp cận được sử dụng rộng rãi nhất, các dịch vụ được cung cấp bởi một máy chủ và được phục vụ cho một hoặc nhiều máy khách – thường là từ xa.
- Thực thi từ xa: một thành phần gửi mã đến thành phần khác để thành phần đó thực thi từ xa do quyết định của chính nó, một yêu cầu từ thành phần từ xa hoặc thậm chí có thể là một phần của một giao ước đã tồn tại trước đó. Sau mỗi lần thực thi, thành phần thực hiện thường trả lại bất kỳ kết quả nào tới thành phần ban đầu (thành phần đã gửi mã tới).

- Các tác tử di động: một thành phần tự gửi chính bản thân nó (hoặc cả thành phần khác,nếu được phép) tới một máy chủ từ xa để thực thi. Thành phần này chuyển cả code, dữ liệu và có thể toàn bộ trạng thái của nó. Động lực có thể tương tự như trường hợp trên (thực thi từ xa), nhưng thông thường nhất là bởi quyết định của chính thành phần (tức là tác tử di động), nó muốn di chuyển tới một vị trí thay thế.

Một tác tử di động, bao gồm 3 thành phần: code, trạng thái và dữ liệu.Code là phần của tác tử sẽ được thực thi khi nó di chuyển tới một platform khác. Trong trường hợp đơn giản nhất, chỉ có một code đơn nhất. Trạng thái là môi trường thực thi dữ liệu của tác tử,bao gồm bộ đếm chương trình và ngăn xếp tác vụ. Thành phần này chỉ được tìm thấy ở các tác tử "di chuyển mạnh" .Dữ liệu bao gồm các biến mà các tác tử sử dụng, như tri thức, định danh tập tin...Trong "di chuyển yếu" thành phần này thực sự cần thiết vì code tác tử được cấu tạo như một máy trạng thái và để duy trì thông tin trạng thái đòi hỏi phải có các biến này.

1.1.6.2. Một số ưu điểm và nhược điểm của tác tử di động

Đã có nhiều cuộc tranh luận về những ưu điểm và nhược điểm khác nhau của các tác tử điện thoại di động, thường so với người anh em họ của nó không phải di động. Một số trong những lợi thế điển hình là:

- Tiến trình độc lập và không đồng bộ: Một khi nó đã di chuyển đến một nền tảng mới, các tác tử không có liên hệ với chủ sở hữu của mình để thực hiện nhiệm vụ của nó. Nó chỉ có thể cần phải gửi lại kết quả. Điều này đặc biệt hữu ích khi xem xét các thiết bị di động với nguồn lực hạn chế; tác tử có thể được di chuyển đến máy khác để thực hiện các nhiệm vụ phức tạp và định kỳ trở lại kết quả.
- Sự chịu lỗi : Nó có thể giải quyết và hỗ trợ với điều kiện lỗi bằng cách di chuyển để thay thế nền tảng khi vấn đề được phát hiện. Tương tự, nếu một điểm đến di cư là xuống, một trung gian có thể được lựa chọn như là một máy chủ lưu trữ tạm thời. Điều này làm cho chúng rất thích hợp cho những môi trường thù địch, không thân thiện.
- Các ứng dụng lớn: các tác tử di động rất thích hợp cho các ứng dụng mà cần quá trình lớn lượng dữ liệu từ xa.Điện thoại di động các tác tử có thể di chuyển các dữ liệu, chứ không phải ngược lại mà nhiều trường hợp là một lựa chọn hiệu quả hơn nhiều.

Nhưng các tác tử điện thoại di động cũng có một số nhược điểm.

- Khả năng mở rộng và hiệu suất: Mặc dù các tác tử điện thoại di động giảm tải mạng, nó cũng có xu hướng tăng cường xử lý tải vì chúng thường được lập trình với ngôn ngữ thông dịch và cũng thường cần phải thực hiện nghiêm ngặt các tiêu chuẩn tương hợp có thể phải chịu xử lý dữ liệu chi phí chung.
- Tính linh động và tiêu chuẩn hóa: tác tử không có thể hoạt động nếu họ không làm theo tiêu chuẩn giao tiếp chung. Thông qua các tiêu chuẩn này, FIPA thường là cần thiết, đặc biệt là liên nền tảng di động.
- Bảo mật: Việc sử dụng các tác tử điện thoại di động có thể mang về vấn đề an ninh. Bất kỳ mã điện thoại di động cung cấp một mối đe dọa tiềm năng và phải được chứng thực một cách cẩn thận trước khi gọi.

1.1.6.3. Di chuyển mạnh và di chuyển yếu

Trong các hệ thống tác tử di động có thể chia thành 2 loại di chuyển cơ bản: di chuyển mạnh và di chuyển yếu.

Di chuyển mạnh thì thường phức tạp hơn. Đó là trường hợp sự thực thi của một tác tử ổn định, sự di chuyển diễn ra, sau đó sự thực thi được khởi động lại ngay từ chỉ dẫn tiếp theo. Kỹ thuật này đòi hỏi phải bảo vệ và lưu lại trạng thái của tác tử trong suốt quá trình di chuyển. Việc cài đặt kỹ thuật này có thể phức tạp bởi nó đòi hỏi truy cập các tham số nội bộ của tác tử - thường chỉ dành cho hệ điều hành; và rất phụ thuộc cấu trúc.

Mặt khác, di chuyển yếu không gửi trạng thái của tác tử, do đó đơn giản hơn nhiều. Sự thực thi của tác tử luôn khởi động lại từ đầu mã. Loại di chuyển này đòi hỏi tác tử được cài đặt như một máy trạng thái hữu hạn để trạng thái được duy trì.

1.1.6.4. Quá trình di chuyển

Một hành trình xác định địa điểm mà một tác tử điện thoại di động phải đến để hoàn thành một tập hợp các nhiệm vụ. Hai các loại hình cơ bản của hành trình có thể phân biệt:

- Hành trình tĩnh được xác định tại thời điểm tạo tác tử mà không có bất kỳ khả năng sửa đổi trong khi tác tử thực hiện.
- Hành trình năng động được xác định trong khi thực hiện đại diện theo nhu cầu và ham muốn.

1.1.7. Tạo tác tử

Tạo một tác tử trong jade chỉ đơn giản là định nghĩa một lớp extends lớp jade.core.agent và cài đạt phương thức setup() như ví dụ dưới đây:

```
import jade.core.Agent;
public class HelloWorldAgent extends Agent {
   protected void setup() {
       // Printout a welcome message
System.out.println("Hello World. I’m an agent!");
}
}
```

Lớp HelloWorldAgent ở trên đại diện cho một loại tác tử chính xác hơn là một lớp thông thường biển thị cho một đối tượng. Một vài thể hiện của lớp HelloWorldAgent có thể chạy lúc run-time. Không giống như đối tượng java thông thường được xử lý qua tham chiếu của chúng, một tác tử luôn luôn được thể hiện bởi JADE run-time và tham chiếu của nó không bao giờ được đặt ngoài tác tử chính nó (tất nhiên là trừ khi các tác tử đó rõ ràng). Các tác tử không bao giờ tương tác qua lời gọi các phương thức mà là tương tác bằng cách trao đổi các thông điệp không đồng bộ, sẽ được giới thiệu ở mục sau.

Phương thức setup() có mục đích để gộp các khởi tạo của tác tử. Thông thường công việc chính xác của các tác tử được thực hiện bên trong các hành vi “behaviours”. Ví dụ về các hoạt động điển hình mà tác tử thực hiện trong hàm setup() của nó là : đưa ra một GUI, mở kết nối đến cơ sở dữ liệu, đăng ký các dịch vụ nó cung cấp trong mục các trang vàng và bắt đầu khởi tạo các behaviours. Tốt nhất là không nên xây dựng hàm khởi tạo trong lớp tác tử và thực hiện tất cả các khởi tạo trong phương thức setup(). Điều này là vì tại thời điểm xây dựng , tác tử vẫn chưa được liên kết với JADE run-time phía dưới và vì vậy một vài phương thức kế thừa từ lớp Agent có thể không làm việc một cách chắc chắn.

1.1.8. Ứng dụng hệ thống đa tác tử

Các hệ thống đa tác tử đang ngày càng được sử dụng rộng rãi trong rất nhiều ứng dụng, từ các hệ hống tương đối nhỏ để hỗ trợ cá nhân đến các hệ thống mở, phức tạp, và đặc biệt quan trọng dành cho các ứng dụng công nghiệp.

Ứng dụng công nghiệp là những ứng dụng rất quan trọng cho các hệ thống đa tác tử bởi vì chúng là nơi mà công nghệ đa tác tử đầu tiên được thử nghiệm và

chứng minh tiềm năng ban đầu của nó. Các ví dụ về việc áp dụng các hệ thống đa tác tử trong lĩnh vực công nghiệp bao gồmkiểm soát tiến trình, chẩn đoán hệ thống, dịch vụ vận tải, và quản lý mạng.

Một trong những lĩnh vực ứng dụng quan trọng nữa của hệ thống đa tác tử là quản lý thông tin. Thật ra, Internet đã chứng tỏ là một miền lý tưởng cho các hệ thống đa tác tử nhờ bản chất phân tán tự nhiên của nó và khối lượng thông tin sẵn có đang ngày càng tăng lên mạnh mẽ.Ví dụ, các tác tử có thể được sử dụng để tìm kiếm và lọc thông tin. Internet đã thúc đẩy việc sử dụng công nghệ tác tử trong lĩnh vực quản lý tiến trình nghiệp vụ và thương mại điện tử.

Giao thông và vận tải cũng là một lĩnh vực quan trọng, nơi mà bản chất phân tán của các tiến trình giao thông và vận tải và sự độc lập mạnh mẽ giữa các thực thể có liên quan trong các tiến trình đó làm cho các hệ thống đa tác tử trở thành một công cụ có giá trị cho việc thực hiện các giải pháp thương mại thực sự có hiệu quả.

Các hệ thống viễn thông là một lĩnh vực ứng dụng đã sử dụng thành công các hệ đa tác tử .Trong thực tế, các hệ thống viễn thông là các mạng lưới lớn và phân tán gồm các thành phần được kết nối với nhau. Những thành phần đó cần phải được theo dõi và quản lý trong thời gian thực. Các hệ thống viễn thông cũng hình thành nên cơ sở của một thị trường cạnh tranh, nơi các công ty viễn thông và các nhà cung cấp dịch vụ nhắm đến để phân biệt chính họ với đối thủ cạnh tranh của họ bằng cách cung cấp các dịch vụ tốt hơn, nhanh hơn hoặc đáng tin cậy hơn. Vì vậy, các hệ đa tác tử được sử dụng cả trong việc quản lý các mạng lưới phân tán lẫn cho việc cài đặt các dịch vụ viễn thông tiên tiến.

1.2. Nền tảng tác tử vật ký và tác tử thông minh

Phần này sẽ giới thiệu một cách tổng quát về lịch sử phát triển,mục tiêu và nội dung chính của đặc tả FIPA [2].

JADE là sự thi hành của các đặc tả FIPA nên nó phụ thuộc nhiều vào các ý tưởng được đưa ra trong các đặc tả và được mở rộng hơn.Tuy nhiên các đặc tả FIPA không được thể hiện hoàn toàn trong JADE vì JADE có mở rộng thêm một số khu vực so với đặc tả.

1.2.1. FIPA lịch sử và mục tiêu

FIPA được thành lập vào năm 1996 là một hiệp hội phi lợi nhuận quốc tế để phát triển một bộ sưu tập của các tiêu chuẩn liên quan đến công nghệ phần mềm tác tử. Các thành viên ban đầu gồm một nhóm các tổ chức nghiên cứu và công nghiệp

đã xây dựng một bộ luật hướng dẫn xây dựng các đặc tả chuẩn, hợp pháp cho các công nghệ tác tử. Lúc đó các tác tử phần mềm đã rất nổi tiếng trong cộng đồng nghiên cứu nhưng cho đến nay chỉ nhận được sự chú ý hạn chế từ các doanh nghiệp thương mại vượt ra ngoài quan điểm thăm dò. Các tập đoàn đã đồng ý để sản xuất tiêu chuẩn đó sẽ hình thành nền tảng của một ngành công nghiệp mới bằng cách sử dụng trên một số lượng lớn ứng dụng.

Cốt lõi của FIPA là tập hợp các nguyên tắc sau đây:

- Tác tử công nghệ cung cấp một mô hình mới để giải quyết các vấn đề cũ và mới.
- Một số nghệ nhân đã đạt đến một mức độ đáng kể của sự trưởng thành.
- Để được sử dụng một số công nghệ tác tử yêu cầu tiêu chuẩn hóa.
- Tiêu chuẩn chung của các công nghệ đã được chứng minh là có thể, và để cung cấp hiệu quả kết quả của các diễn đàn tiêu chuẩn hóa khác.
- Các tiêu chuẩn của các cơ bên trong của tác tử không phải là mối quan tâm chính mà là cơ sở hạ tầng và ngôn ngữ cần thiết cho hướng mở.

FIPA bước đầu đã được thiết lập với một nhiệm vụ năm năm để xác định các lĩnh vực được lựa chọn của hệ thống đa tác tử.Nhiệm vụ này là vô thời hạn gia hạn vào năm 2001. Cho đến cuối năm 2005 FIPA đã được điều chỉnh bởi một bầu thành viên Hội đồng quản trị chịu trách nhiệm hướng dẫn chiến lược và quản lý nhiệm vụ hành chính chính thức.

Đến nay một số các thành tựu chính của FIPA như sau:

- Tập chuẩn đặc tả hỗ trợ quá trình giao tiếp giữa các tác tử và các dịch vụ chính ở tầng trung gian.
- Một kiến trúc trừu tượng cung cấp cái nhìn hoàn thiện thông qua các chuẩn FIPA 2000.
- Một ngôn ngữ giao tiếp agent rõ ràng và được sử dụng nhiều (FIPA-ACL), kèm theo một tập các ngôn ngữ nội dung (ví dụ như FIPA-SL) và một tập các giao thức chính áp dụng từ quá trình trao đổi thông điệp đơn giản cho đến các quá trình giao dịch phức tạp .
- Một số công cụ tác tử thương mại và nguồn mở, như JADE, ngày nay đã được coi là công nghệ mã nguồn mở tuân theo FIPA được sử dụng rộng rãi.

- Ngoài FIPA, có nhiều dự án đã hoàn thành như dự án Agentcities đã tạo một mạng lưới các platform tuân theo FIPA và các dịch vụ ứng dụng tác tử.
- Một mở rộng của UML chuyên về tác tử là AUML đã được đề xuất.

1.2.2. Các khái niệm cốt lõi FIPA

Trong quá trình phát triển của FIPA, nhiều ý tưởng liên quan tới tác tử được đề xuất. Một số ý tưởng trở thành chuẩn, một số khác được phát triển nhưng chưa hoàn thành, số còn lại bị thất bại bởi nhiều nguyên nhân nào đó. Các ý tưởng này đều xoay quanh một số khía cạnh chính là giao tiếp giữa tác tử, quản lý tác tử, và kiến trúc tác tử. Mục này sẽ thảo luận các khái niệm chính liên quan tới các khía cạnh đó.

1.2.2.1. Giao tiếp giữa các tác tử

Các tác tử về cơ bản là một dạng của các tiến trình lập trình phân tán và vì vậy tuân theo khái niệm cổ điển của mô hình tính toán phân tán bao gồm 2 phần: thành phần (component) và kết nối (connector). Thành phần là người tiêu dùng, người sản xuất và người trung gian truyền các thông điệp được trao đổi thông qua kết nối. Các chuẩn như ISO và IETF mang cách tiếp cận hướng mạng (network-oriented) trong việc phát triển các ngăn xếp giao thức được phân lớp (layeredprotocol stack) chiếm đa số trong các giao tiếp máy tính (computer communication) chúng ta biếtngày nay như Mô hình tham chiếu OSI và mô hình TCP/IP. Cả 2 đều được sử dụng thông qua các interface của các dịch vụ phần mềm cài đặt các giao thức.

FIPA – ACL dựa trên lý thiết lời nói hành động (speech act theory). Lý thuyết này chỉ ra rằng các thông điệp biểu diễn các hành động, hoặc các hành động giao tiếp – cũng được biết đến như là các hành động lời nói (speech act) hoặc biểu hiện (performative).

Một số hoạt động được sử dụng chung nhất là cho biết (inform), yêu cầu(request), đồng ý (agree), không hiểu (not understood) và từ chối (refuse). Chúng ghi lại những dạng thường gặp nhất trong giao tiếp cơ bản. Nó được xác định trong các chuẩn FIPA để được tuân theo một cách đầy đủ khi agent nhận bất kì thông điệp giao tiếp hành động FIPA – ACL nào và ít nhất là đáp ứng bằng một thông điệp not-understood nếu thông điệp nhận được không thể xử lý được.

Dựa vào các kiểu giao tiếp hành động này, FIPA định nghĩa một tập các giao thức tương tác, mỗi cái bao gồm một chuỗi các giao tiếp hành động để kết hợp các

hành động đa thông điệp,như mạng hợp đồng (contract net) dành cho việc thiết lập các thoả thuận và các kiểu đấu giá. Bên trong cấu trúc của mỗi thông điệp, FIPA – ACL không uỷ thác việc sử dụng một ngôn ngữ cụ thể nào để biểu diễn nội dung, mặc dầu nhiều đặc tả dành cho một số kiểu biểu diễn nội dung như FIPA – SL, FIPA – KIF và FIPA – RDF đã được đề xuất.

1.2.2.2. Các lớp con của FIPA

Như đã đề cập ở trước, ngăn xếp giao tiếp FIPA có thể được phân chia thành một số lớp con trong lớp ứng dụng ngăn xếp OSI hoặc TCP/IP. Chúng được trình bày chi tiết như dưới đây:

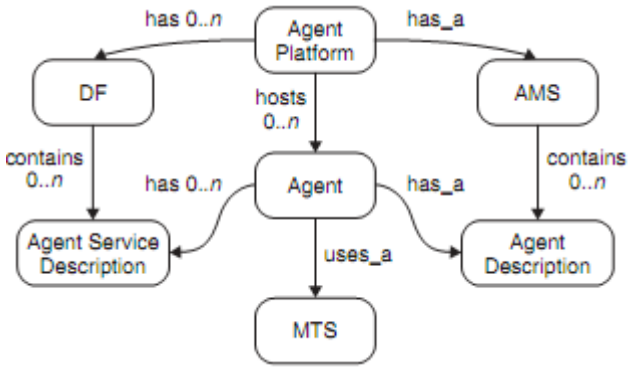
- *Sub-layer 1* (Transport): trong mô hình giao thức phân lớp FIPA – ACL, giao thức lớp con thấp nhất là giao thức vận chuyển. FIPA định nghĩa các giao thức vận chuyển thông điệp (message transport protocol) như IIOP (IIOP, 1999), WAP (WAP) và HTTP (HTTP) .
- *Sub-layer 2* (Encoding): Ngoài việc gửi các thông điệp được mã hoá theo từng byte, FIPA còn định nghĩa một vài cách biểu diễn thông điệp sử dụng cho các cấu trúc dữ liệu ở mức cao bao gồm XML, String và Bit – Efficient. Bit – Efficient dự kiến sẽ sử dụng khi giao tiếp qua các kết nối băng thông thấp.
- *Sub-layer 3* (Messaging): Trong FIPA, cấu trúc thông điệp được xác định độc lập với việc mã hoá để khuyến khích sự linh hoạt. Khía cạnh quan trọng ở mức này là các tham số chính cần thêm vào payload hoặc nội dụng được trao đổi, ví dụ người gửi và người nhận, kiểu thông điệp, thời gian đáp ứng. Một ví dụ của cấu trúc thông điệp FIPA – ACL .
- *Sub-layer 4 (Ontology):* thuật ngữ này chứa trong payload hoặc nội dung của một thông điệp FIPA có thể được tham chiếu một cách rõ ràng sang mô hình khái niệm chuyên về một ứng dụng cụ thể (application-specific) hoặc bản thể (ontology). Mặc dù về bản chất FIPA cho phép sử dụng các ontology khi biểu diễn nội dung thông điệp, nhưng nó không chỉ ra bất kì cách biểu diễn nào cho các ontology hoặc cung cấp các ontology cho một lĩnh vực cụ thể nào. Nó có thể tham chiếu đến các *ontology* dựa trên web nếu được yêu cầu.
- *Sub-layer 5* (Content expression): Dữ liệu thật của các thông điệp FIPA có thể có một dạng nào đó, nhưng FIPA đã định nghĩa các hướng dẫn sử dụng các công thức và vị từ logic chung, và các phép tính đại số để kết hợp và lựa

chọn các khái niệm. Ngôn ngữ thường được sử dụng nhất cho việc biểu diễn nội dung là FIPA – SL, ví dụ của các công thức logic bao gồm: not, or, implies (kéo theo), equiv… và ví dụ của các toán tử đại số như any và all.

- *Sub-layer 6* (Communicative act): việc phân loại thông điệp đơn giản là chia chúng thành các loại: action hay performative. Ví dụ: inform, request và agree.
- *Sub-layer 7* (Interaction protocol or IP): các thông điệp hiếm khi được trao đổi một cách riêng biệt mà thường được hình thành một số chuỗi tương tác. FIPA định nghĩa một số giao thức tương tác chỉ ra các chuỗi trao đổi thông điệp điển hình như request, nó miêu tả một nhóm thông điệp tham gia vào việc tạo một yêu cầu tới các agent khác và phản hồi là agree hoặc refuse.

1.2.2.3. Sự quản lý tác tử

Ngoài giao tiếp, khía cạnh cơ bản thứ hai của các hệ thống tác tử được đề cập trong các đặc tả FIPA ban đầu là quản lý agent: một nền tảng chuẩn trong đó các tác tử tuân theo FIPA có thể tồn tại, hoạt động và được quản lý. Nó thiết lập mô hình tham chiếu logic cho việc tạo, đăng kí, định vị, giao tiếp, di trú và hoạt động của các tác tử. Mô hình tham chiếu quản lý tác tử gồm các thành phần được mô tả trong Hình 1.5



Hình 1.5. Minh họa mô hình tham chiếu quản lý agent

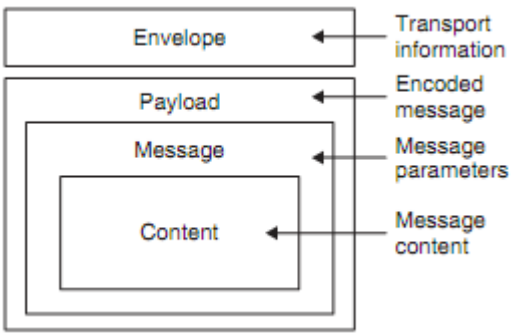
Agent Platform (AP): cung cấp cơ sở hạ tầng vật lý trong đó tác tử được triển khai. AP bao gồm các cơ chế, các hệ điều hành, các thành phần FIPA quản lý tác tử, các tác tử và phần mềm hỗ trợ. Thiết kế cụ thể bên trong của AP không được miêu tả ở đây. Một AP đơn có thể trải rộng trên nhiều máy tính, các tác tử cư trú trên đó cũng không phải đặt trên cùng một host.

Agent: một tác tử là một tiến trình có sử dụng máy tính. Nó nằm trong AP và thường cung cấp một hoặc nhiều dịch vụ có sử dụng máy tính. Những dịch vụ này có thể được xuất bản dưới một bản miêu tả dịch vụ. Một tác tử phải có ít nhất một đối tượng sở hữu nó và phải hỗ trợ ít nhất một khái niệm để xác định cái nào có thể được miêu tả bởi FIPA Agent Identifier (AID). AID là cái dùng để gán nhãn cho một tác tử để nó có thể được phân biệt một cách rõ ràng. Một tác tử có thể được đăng ký một số địa chỉ vận chuyển để có thể liên hệ.

Directory Facilitator (DF): DF là một thành phần tùy chọn của AP. Nó cung cấp các dịch vụ của các trang vàng cho các tác tử khác. Nó duy trì một danh sách các tác tử đúng đắn, hoàn chỉnh và hợp thời và phải cung cấp các thông tin phổ biến nhất về tác tử trong thư mục của nó trên cơ sở không có sự phân biệt đối xử giữa tất cả các tác tử đã được chứng thực. AP có thể hỗ trợ nhiều DF mà những DF này có thể đăng ký cùng với nhau để hình thành một liên đoàn.

Agent Management System (AMS): AMS là một thành phần bắt buộc của AP và chịu trách nhiệm quản lý các thao tác của AP, như tạo mới và xóa tác tử, và dự đoán việc đến và đi của tác tử. Mỗi tác tử phải đăng ký với một AMS để có được AID, cái mà sau đó được giữ lại làm thư mục cho mọi tác tử và trạng thái hiện tại của chúng (như active, suspended hay waiting) trong AP. Các miêu tả của tác tử sau đó có thể được sửa đổi bởi AMS trong giới hạn cho phép. Sau khi hủy đăng ký, AID có thể bị xóa và có thể dùng để phục vụ các tác tử khác đang yêu cầu nó. Chỉ một AMS đơn mới có thể tồn tại trong mỗi AP và nếu AP trải rộng trên nhiều máy, AMS cũng có quyền trên tất cả các máy đó.

Message Transport Service (MTS): là một dịch vụ được cung cấp bởi AP để vận chuyển các thông điệp FIPA-ACL giữa các tác tử trong một AP và giữa các tác tử trong các AP khác nhau. Các thông điệp cung cấp một tem vận chuyển chứa tập các tham số chi tiết như người nhận…Cấu trúc chung của thông điệp như sau:



Hình 1.6. Cấu trúc thông điệp FIPA

1.2.2.4. Kiến trúc trừu tượng

Ngoài việc giao tiếp và quản lý tác tử, giữa năm 2000 và 2002 một kiến trúc tác tử trừu tượng được tạo ra và chuẩn hóa làm phương tiện để tránh sự ảnh hưởng của nền tảng cài đặt lên đặc tả cốt lõi. Điều này có được bằng cách trừu tượng hóa các khía cạnh chính của các cơ chế quan trọng nhất như vận chuyển thông điệp và các dịch vụ trong thư mục thành một đặc tả thống nhất. Mục tiêu chính của cách này là để cho phép việc tạo ra các hệ thống được tích hợp một cách nhuần nhuyễn trong môi trường tính toán của chúng trong khi vẫn tương tác với các hệ thống tác tử đang cư trú trong các môi trường riêng biệt.

Các hệ tác tử được xây dựng theo các đặc tả ban đầu của FIPA 97 và FIPA 98 có thể tương tác với các hệ tác tử khác được xây dựng theo kiến trúc trừu tượng thông qua các cổng (gateway) vận chuyển với một số hạn chế. Kiến trúc FIPA2000 là một ánh xạ gần gũi hơn và cho phép tương tác một cách đầy đủ thông quang các gateway. Vì kiến trúc trừu tượng cho phép tạo nhiều thể hiện rõ ràng, nên nó cũng cung cấp các cơ chế cho phép các thể hiện đó có thể tương tác với nhau như chuyển đổi gateway cho cả việc vận chuyển và mã hóa.

1.2.3. Các liên quan đến FIPA và JADE

Nhớ rằng FIPA dựa trên nguyên lý là chỉ đặc tả các hành vi bên ngoài của các thành phần trong hệ thống, bỏ qua kiến trúc và chi tiết cài đặt bên trong. Điều này đảm bảo sự kết nối liền mạch giữa các nền tảng biên dịch đầy đủ. JADE tuân theo quan điểm này ở chỗ nó đảm bảo tính tương thích trọn vẹn với đặc tả FIPA2000 (truyền thông, quản lý và kiến trúc) – đặc tả này cung cấp một framework chuẩn trong đó các tác tử có thể tồn tại, vận hành và giao tiếp trong khi vẫn chấp nhận một kiến trúc bên trong thống nhất và độc quyền và chấp nhận cài đặt các dịch vụ và các tác tử chính.

JADE tất nhiên chỉ là một trong các nền tảng ứng dụng và dự án có tính cộng tác về tác tử tuân theo các chuẩn của FIPA. Việc tuân theo này đã được kiểm tra thông qua một số sự kiện:cuộc kiểm tra tính tương kết của FIPA năm 1999 và 2001, dự án Agentcities. Về mặt độ bao phủ của các chuẩn của FIPA, JADE cài đặt hoàn chỉnh đặc tả quản lý tác tử bao gồm các dịch vụ:AMS, DF, MTS và ACC.

CHƯƠNG 2: NỀN TẢNG JADE

Phần này cung cấp một cái nhìn tổng quan cơ bản của nền tảng JADE [1] [8] và các thành phần chính cấu thành kiến trúc của nó. Hơn nữa, nó mô tả cách thức để khởi động nền tảng với nhiều tùy chọn dòng lệnh.

2.1. JADE là gì?

JADE là một nền tảng phần mềm cung cấp lớp trung gian cơ bản các chức năng của các ứng dụng cụ thể và giúp đơn giản hóa việc thực hiện các ứng dụng phân tán khai thác các trừu tượng tác tử phần mềm.

2.2. Tóm tắt lịch sử

Những phần mềm phát triển đầu tiên, cuối cùng trở thành nền tảng JADE, đã được bắt đầu xây dựng bởi Telecom Italia (viết tắt là CSELT) cuối năm 1998, do cần sớm có sự xác nhận các đặc tả kỹ thuật FIPA.

JADE đã trở thành mã nguồn mở từ năm 2000 và được phân phối bởi Telecom Italia theo giấy phép LGPL. Giấy phép này đảm bảo tất cả các quyền cơ bản để tạo thuận lợi cho việc sử dụng phần mềm có trong các sản phẩm thương mại.

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

Để tạo điều kiện tham gia tốt hơn, tháng 5 năm 2003 Telecom Italia Lab và Motorola Inc, đã đưa ra một thỏa thuận hợp tác và thành lập Ủy Ban Quản Lý JADE, một tổ chức phi lợi nhuận của các công ty quan tâm đóng góp cho sự phát triển của JADE.

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. Ngày nay, nó được dùng như một JADE run-time cho các nền tảng J2ME-CLDC và J2ME-CDC, và 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.

2.3. JADE và mô hình tác tử

JADE là một nền tảng phần mềm cung cấp chức năng cơ bản cho tầng giữa, độc lập với các ứng dụng cụ thể và đơn giản hóa việc thực hiện của các ứng dụng phân tán – những ứng dụng khai thác sự trừu tượng của các tác tử.Một đặc điểm đầy ý nghĩa của JADE là nó thực thi

Một tác tử có tính tự chủ và hướng đích: một tác tử không thể cung cấp các callback hoặc tham chiếu đối tượng của chính nó tới các tác tử khác để làm giảm đi cơ hội điều khiển của các thực thể lên các dịch vụ của nó. Một tác tử phải có luồng thực thi của chính nó, sử dụng nó để điều khiển vòng đời của nó và tự chủ quyết định khi nào thực thi các hành động.

Các tác tử có thể nói không, và chúng được gắn kết lỏng lẻo: Việc giao tiếp không đồng bộ dựa trên thông điệp là hình thức giao tiếp cơ bản giữa các tác tử trong JADE; một tác tử muốn giao tiếp phải gửi thông điệp đến một điểm được xác định (hoặc thiết lập các điểm đến).Hơn nữa, dạng thức giao tiếp này cho phép bên nhận có quyền lựa chọn thông điệp sẽ xử lý hay loại bỏ, cũng như có quyền xác định các mức ưu tiên xử lý của chính nó.

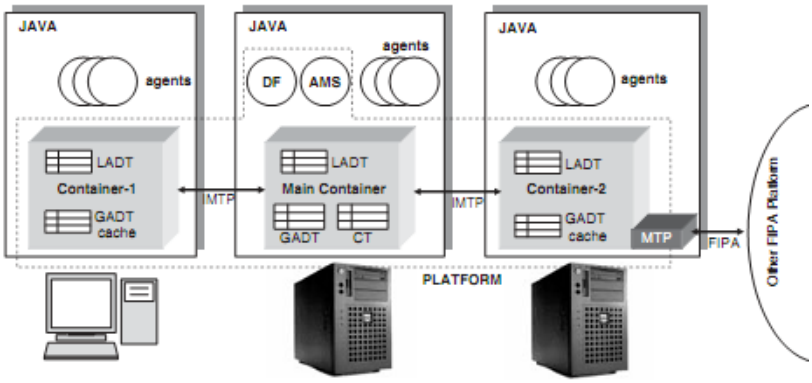
Hệ thống có kiểu Peer-to-Peer: mỗi tác tử được xác định bởi một tên toàn cục duy nhất. Nó có thể tham gia vào và rời khỏi một nền tảng máy chủ ở bất kỳ thời điểm nào và có thể nhận ra các tác tử khác thông qua cả 2 dịch vụ white-page và yellow-page cung cấp trong JADE bởi AMS và DF mà đã được định nghĩa bởi FIPA.Trên cơ sở những lựa chọn thiết kế này, JADE đã được cài đặt để cung cấp cho các nhà lập trình các chức năng cốt lõi sẵn sàng để sử dụng và dễ dàng để tùy biến.

2.4. Kiến trúc JADE

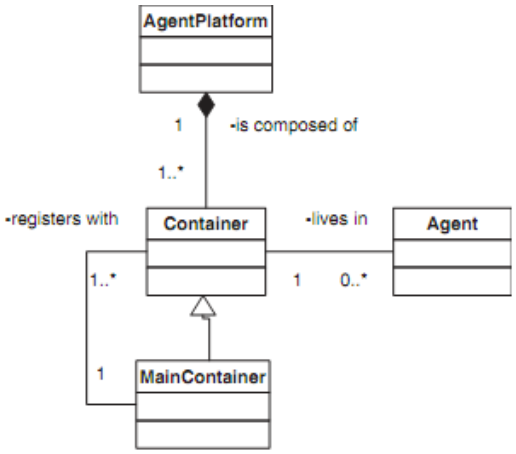
Hình 2.1 chỉ ra các thành phần kiến trúc chính của JADE platform. Một JADE platform bao gồm các khung chứa tác tử (agent containers), có thể được phân tán trên mạng. Các tác tử sống trong khung chứa là các tiến trình Java, cung cấp JADE run-time và tất cả các dịch vụ cần cho việc lưu trú và thực thi các tác tử. Có một khung chứa đặc biệt, được gọi là khung chứa chính (main container), thể hiện nét nổi bật của platform: Nó là khung chứa đầu tiên được khởi chạy và tất cả các khung chứa khác phải đăng kí để gia nhập vào khung chứa chính. Biểu đồ UML trong hình 2.2 miêu tả quan hệ giữa các thành phần kiến trúc chính trong JADE.

Người lập trình sử dụng tên logic để xác định các khung chứa; mặc định, khung chứa chính được gọi là "Main Container" trong khi các khung chứa khác có tên lần lượt là "Container-1", "Container-2",…Các dòng lệnh khác nhau đã sẵn có để thay đổi các tên mặc định đó. Main container có những nhiệm vụ đặc biệt sau:

- Quản lý bảng khung chứa (container table - CT), nơi đăng kí các tham chiếu của đối tượng và các địa chỉ giao dịch của tất các khung chứa có trong platform.
- Quản lý bảng miêu tả tác tử cục bộ (Global agent descriptor table -GADT), là nơi đăng kí của tất cả các tác tử trong platform, bao gồm cả trạng thái hiện tại và vị trí của chúng .
- Hosting AMS và DF, hai tác tử đặc biệt cung cấp việc quản lý tác tử (agent management) và dịch vụ trang trắng (white page service), và dịch vụ trang vàng măc định của platform (default yellow page service).



Hình 2.1. Các thành phần kiến trúc chính



Hình 2.2. Mối quan hệ giữa các yếu tố kiến trúc chính

Định danh của tác tử được chứa trong Agent Identifier (AID), gồm một tập các khe tuân thủ cấu trúc và ngữ nghĩa được đưa ra bởi FIPA. Các thành phần cơ bản nhất của AID là tên tác tử và địa chỉ của nó. Tên của tác tử là định danh toàn cục duy nhất mà JADE xây dựng bằng cách kết hợp nickname được định nghĩa bởi người dùng (được biết như tên cục bộ sử dụng trong giao tiếp intra-platform) với tên của platform. Địa chỉ của tác tử là địa chỉ giao dịch được kế thừa từ platform, mỗi địa chỉ platform tương ứng với một điểm cuối MTP (Message Transport Protocol), nơi các thông điệp theo chuẩn FIPA có thể được gửi và nhận. Người lập trình tác tử cũng được phép thêm các địa chỉ giao vận riêng vào AID, khi họ muốn tự cài đặt MTP. Khi khung chứa chính được khởi chạy, hai tác tử đặc biệt được tự động khởi tạo và được bắt đầu bởi JADE, vai trò của chúng được định nghĩa bởi chuẩn quản lý tác tử của FIPA (FIPA Agent Management standard):

- Hệ thống quản lý tác tử (Agent Management System -AMS) là tác tử quản lý toàn bộ platform. Nó là điểm kết nối cho tất cả các tác tử muốn tương tác để truy cập trang trắng của platform cũng như để quản lý chu trình sống của chúng. Mọi tác tử phải đăng kí với AMS (được thực hiện một cách tự động bởi JADE lúc tác tử khởi tạo) để có một AID hợp lệ.
- Directory Facilitator (DF) là tác tử triển khai dịch vụ trang vàng, được sử dụng bởi các tác tử khi chúng muốn đăng kí các dịch vụ của chúng hoặc tìm kiếm các dịch vụ có sẵn khác. JADE DF cũng chấp nhận các đặc tả từ các tác tử với mong muốn nhận thông báo bất cứ khi nào có một dịch vụ được đăng kí hay sửa đổi. Nhiều DF có thể được bắt đầu đồng thời để phân tán dịch vụ trang vàng tới nhiều miền khác nhau. Các DF này có thể được hợp nhất thành liên đoàn nếu cần thiết, bằng cách thiết lập các đăng kí (cross-registration) với một DF khác (là DF cho phép truyền bá các yêu cầu của tác tử tới toàn bộ liên đoàn đó.

2.5. Những đặc điểm cơ bản của JADE

2.5.1. Cài đặt nhiệm vụ cho tác tử.

Như được trình bày việc cài đặt tác tử thể hiện trong các hành vi (behaviour). Một behaviour đại diện cho một nhiệm vụ mà tác tử có thể thực hiện và được cài đặt như một đối tượng của một lớp kế thừa jade.core.behaviours.Behaviour. Để một tác tử thực thi nhiệm vụ được cài đặt trong đối tượng behaviour, behaviour phải được add vào tác tử bằng phương thức

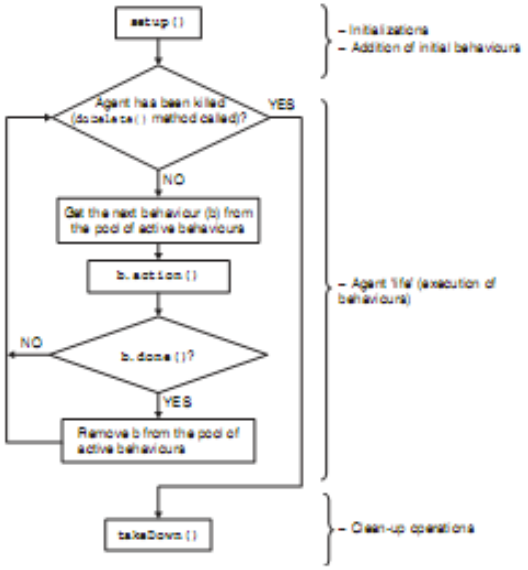
addBehaviour() của lớp Agent. Các Behaviour có thể được add vào bất kì thời gian nào khi tác tử bắt đầu (trong phương thức setup()) hoặc từ trong các behaviour khác. Mỗi lớp kế thừa Behaviour phải cài đặt hai phương thức abstract. Phương thức action() định nghĩa các hoạt động được thực hiện khi behaviour thực thi. Phương thức done() trả một giá trị boolean chỉ ra behaviour được hoàn thành hay chưa và được xóa khỏi luồng hành vi của tác tử đang thực thi.

2.5.1.1. Lập lịch và thực thi Behaviour

Một tác tử có thể thực thi đồng thời vài behaviour. Tuy nhiên, điều quan trọng cần lưu ý là việc lập lịch của các behaviour trong tác tử không có sự ưu tiên (giống threads của java), nhưng có sự hợp tác với nhau. Điều này có nghĩa khi một behaviour được lập lịch cho việc thực thi phương thức action() của nó được gọi và chạy cho đến khi trả về.

Mô hình này hiện có một số lợi thế:

- Nó chấp nhận một luồng Java đơn giản bằng tác tử. Điều này rất quan trọng trong môi trường giới hạn về nguồn lực như điện thoại di động.
- Nó cung cấp cải thiện hiệu suất trong việc chuyển hành vi nhanh hơn so với chuyển luồng Java.
- Nó loại bỏ tất cả các vấn đề đồng bộ giữa các behaviour đồng thời truy cập vào cùng tài nguyên từ tất cả các behaviours được thực thi bởi cùng Java thread. Điều này cũng làm nâng cao hiệu suất.
- Khi chuyển đổi behaviour xảy ra, tình trạng của tác tử không bao gồm bất kì thông tin ngăn xếp nào. Điều này cho phép việc thực hiện liên tục một số tính năng nâng cao quan trọng, chẳng hạn như lưu lại trạng thái của tác tử trong bộ lưu trữ lâu dài, hoặc chuyển các tác tử đến container khác để thực thi từ xa (tác tử di động).
- Các bước thực hiện của luồng tác tử được mô tả trong Hình 2.3. Điều quan trọng cần chú ý là một behaviour như phần dưới đây sẽ giải quyết trước bất kì behaviour khác đang được thực thi bởi phương thức action() của nó và không trả về. Khi không có các behaviour để thực thi, thread của tác tử sẽ sleep để đỡ tốn thời gian CPU.Luồng này sẽ được đánh thức trở lại một khi có một behaviour để thực thi.



Hình 2.3. Luồng thực thi của tác tử

2.5.1.2. One-shot behaviour, cyclic behavior và generic behaviour

Có 3 kiểu behaviour chính sẵn có trong JADE như sau:

(1) "One -shot" behaviours được thiết kế để kết thúc một giai đoạn thực thi. Phương thức action() chỉ được thực thi một lần. Lớp jade.core.behaviours.OneShotBehaviour đã cài đặt phương thức done() return "true" và thuận lợi khi mở rộng để cài đặt các one-shot behaviour mới.

(2) "Cyclic" behaviours được thiết kế không bao giờ kết thúc. Phương thức action() thực hiệncác operation cùng lúc mỗi khi được gọi. Lớp Jade.core.behaviours.CyclicBehaviour đã cài đặt phương thức done() return "false" và thuận lợi khi mở rộng để cài đặt các cyclicbehaviour mới.

(3) Các behaviours được nhúng vào ba trạng thái và thực thi các operation khác nhau phục thuộc vào giá trị trạng thái. Chúng kết thúc khi một điều kiện nhất định được đáp ứng. JADE cũng cung cấp khả năng hợp tác với nhau của các behaviours để tạo ra các behaviour phức tạp. Tính năng này, đặc biệt thuận lợi khi cài đặt các nhiệm vụ phức tạp.

2.5.1.3. Bổ sung thêm về hành vi của tác tử

Tất cả các behaviours đều kế thừa các phương thức onStart() và onEnd() từ lớp Behaviour. Các phương thức này được thực thi chỉ một lần trước khi gọi phương thức action() và sau khi phương thức done() trả về true. Chúng nhằm thực hiện các nhiệm vụ đặc biệt để khởi tạo và chấm dứt các operation. Không giống với

các phương thức action() và done() được khai báo abstract, chúng cài đặt mặc định rỗng cho phép người phát triển cài đặt chúng theo ý họ muốn.

Một behaviour có thể bị hủy ở bất cứ thời gian nào khi gọi phương thức removeBehaviour() trong lớp Agent. Do đó nếu behavior bị hủy sử dụng phương thức removebehaviour(), phương thức onEnd() của nó không được gọi. Mỗi behaviour có một biến gọi là "myAgent" trỏ đến tác tử được thực thi behaviour. Cuối cùng điều quan trọng cần ghi nhớ là một đối tượng Behaviour đã được thực thi, nếu nó thực thi lần thứ hai, nó cần gọi phương thức reset() trước tiên. Nếu không làm điều này có thể dẫn đến kết quả không mong muốn.

2.5.1.4. Lập lịch cho các hành vi của tác tử

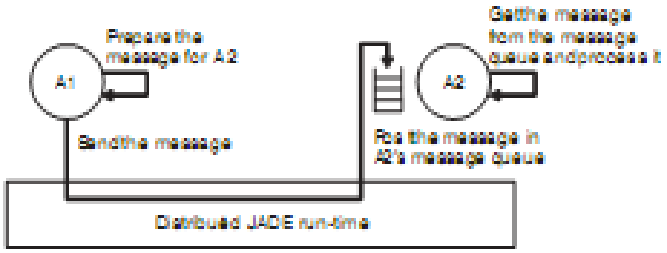
JADE cung cấp hai lớp (trong package jade.core.behaviours) mà có thể cài đặt để tạo các behaviour thực thi khi chọn thời gian cho nó.

(1) WakerBehaviour có các phương thức action() và done() được cài đặt trước để thực thi phương thức abstract onWake() sau 1 thời gian xác định kết thúc (đặc tả trong cấu trúc). Sau khi thực thi phương thức onWake () thì behaviour kết thúc.

(2) TikerBehaviour có các phương thức action() và done được cài đặt trước để thực thi lặp đi lặp lại phương thức abstract onTick(), chờ đợi một thời gian xác định (đặc tả trong cấu trúc) sau mỗi lần thực thi. Một TickerBehaviour không bao giờ kết thúc trừ phi nó được xóa hoặc phương thức stop() của nó được gọi.

2.5.2. Truyền thông giữa các tác tử

Truyền thông giữa các tác tử có lẽ là tính năng cơ bản nhất của Jade và được thực hiện theo các đặc tả FIPA. Các mô hình truyền thông dựa trên truyền thông điệp bất đồng bộ. Như vậy, mỗi tác tử có một "hộp thư" , nơi Jade tại thời gian chạy gửi thông điệp được gửi đến bởi các tác tử khác. Khi nào một thông điệp được gửi vào trong hàng đợi hộp thư thì tác tử được gửi thông điệp đó sẽ nhận được thông báo.



Hình 2.4. Cơ chế truyền thông điệp không đồng bộ trong JADE

2.5.2.1. Gửi thông điệp

Gửi thông điệp đến tác tử khác đơn giản như việc điền vào các trường của một đối tượng ACLMessage và sau đó gọi phương thức send() của lớp Agent. Các ACL performative được định nghĩa bởi FIPA cũng đã định nghĩa các ngữ nghĩa hình thức có thể được khai thác để làm cho một tác tử tự động đưa ra các quyết định đúng đắn khi nhận được một thông điệp. Để giữ cho mọi thứ điều đơn giản đến mức có thể, chúng ta sẽ đặt tiêu đề của cuốn sách muốn mua vào trong nội dung của thông điệp CFP được gửi bởi các tác tử mua. Tương tự, nội dung của các thông điệp PROPOSAL mang theo những lời chào hàng của các tác tử bán sẽ là giá của cuốn sách. Đây là cách một thông điệp CFP có thể được tạo ra và gửi bởi một tác tử mua.

2.5.2.2. Nhận thông điệp

Như đã đề cập ở trên, tại thời gian chạy Jade sẽ tự động đưa các thông điệp vào hàng đợi thông điệp cá nhân của một người nhận ngay sau khi chúng tới. Một agent có thể lấy các thông điệp từ hàng đợi thông điệp của mình bằng phương thức receive(). Phương thức này sẽ trả về thông điệp đầu tiên trong hàng đợi (do đó gây ra việc nó sẽ được bỏ khỏi hàng đợi), hoặc null nếu hàng đợi rỗng, và ngay lập tức trả lại.

2.5.2.3. Khóa hành vi đợi thông điệp

Lập trình viên thường cần phải thực hiện các hành vi xử lý các thông điệp nhận được từ các tác tử khác. Đây là trường hợp đối với hành vi OfferRequestsServervà PurchaseOrderServer, ở đây chúng ta cần phục vụ các thông điệp từ các tác tử mua. Những hành vi này phải được liên tục thực hiện (cyclic behaviours) và, ở mỗi lần thực hiện phương thức action(), phải kiểm tra xem thông điệp đã được nhận chưa và xử lý nó.

Phương thức createReply() của lớp ACLMessage tự động tạo ra một ACLMessage mới, tự động cài đặt những người nhận và bất kỳ một trường cần thiết nào cho việc kiểm soát cuộc hội thoại (ví dụ như conversation-id, reply-with, in-reply-to).

Tuy nhiên, chúng ta có thể nhận thấy rằng ngay khi chúng tôi thêm các hành vi trên, luồng hoạt động của tác tử bắt đầu một vòng lặp liên tục mà cực kỳ tốn dung lượng CPU. Mặt khác, chúng tôi muốn phương thức action() của hành vi OfferRequestsServer như được thực thi chỉ khi nhận được một thông điệp mới. Để làm được điều này, chúng ta phải sử dụng phương thức block() của lớp Behaviour,

trong đó, mặc dù như những gì tên phương thức gợi ý, nó không phải là một cuộc gọi bị chặn, mà chỉ đánh dấu hành vi như 'bị chặn' để các tác tử không còn lập lịch để thực hiện nó. Khi một thông điệp mới được đưa vào hàng đợi của các tác tử, tất cả các hành vi bị cấm trở nên có hiệu lực thực hiện lại để chúng có cơ hội xử lý thông điệp nhận được.

2.5.2.4. Lựa chọn thông điệp từ hàng đợi

Ta thấy OfferRequestsServer và PurchaseOrderServer đều là các hành vi vòng với một phương thức action() bắt đầu với một lời gọi đến myAgent.receive(). Chúng ta có thể nhận thấy một vấn đề là làm thế nào có thể chắc chắn rằng hành vi OfferRequestsServer chỉ đọc từ hàng đợi những thông điệp CFP và hành vi PurchaseOrderServer chỉ đọc những thông điệp chứa yêu cầu mua hàng? Để giải quyết vấn đề này chúng ta phải sửa đổi mã hiện tại bằng cách xác định các "mẫu" (template) để được sử dụng khi gọi phương thức receive(). Khi một mẫu được xác định, phương thức receive() trả về thông điệp đầu tiên thỏa mãn và bỏ qua tất cả các thông điệp không thỏa mãn. Mẫu này được cài đặt là thể hiện của lớp lớp jade.lang.acl.MessageTemplate cung cấp một số phương thức để tạo ra các mẫu một cách rất đơn giản và linh hoạt.

2.5.2.5. Các cuộc hội thoại phức tạp

Các hội thoại phức tạp thường được thực hiện dựa theo một giao thức được xác định rõ ràng, chẳng hạn như những gì được xác định bởi FIPA. Jade cung cấp hỗ trợ phong phú cho một số các giao thức tương tác được sử dụng phổ biến nhất trong gói jade.proto. Cuộc hội thoại mà chúng ta thực hiện ở trên, ví dụ, theo giao thức 'Contract-net' giao thức mà có thể là rất dễ dàng thực hiện bằng cách khai thác lớp jade.proto.ContractNetInitiator. Điều này sẽ tiếp tục được mô tả trong các phần sau.

2.5.2.6. Nhận thông điệp tại node đang khóa

Bên cạnh phương thức receive(), lớp Agent cũng cung cấp phương thức blockingReceive(), nhưtên cho thấy, là một lời gọi khóa: nó không trả lại cho đến khi có một thông điệp trong hàng đợi thông điệp của tác tử. Một phiên bản quá tải mà dùng MessageTemplate như một tham số (nó không trở lại cho đến khi có một thông điệp phù hợp với mẫu quy định) cũng khả dụng. Điều quan trọng cần nhấn mạnh rằng phương thức blockingReceive () thực sự chặn thread của tác tử. Vì vậy nếu bạn gọi blockingReceive () từ trong một hành vi, điều này ngăn cản tất cả các

hành vi khác cho đến khi thực hiện lời gọi đến blockingReceive () trả về. Hãy cùng xem xét,một việc lập trình tốt là để nhận được các thông điệp sử dụng blockingReceive() trong phương thức setup() và takeDown(); sử dụng receive() trong sự kết hợp với Behaviour.block ().

2.5.3. Tác tử với giao diện đồ họa

Một vấn đề điển hình mà những người phát triển phải làm là cách quản lý các agent Jade tương tác với GUI (giao diện đồ họa người dùng) của họ và ngược lại. Vấn đề ở đây là các mô hình lập trình giao tiếp giữa các luồng thích hợp phải được sử dụng giữa các luồng tác tử. Nó phải được đánh thức bất cứ khi nào nhận được một thông điệp ACL và AWT gửi đi và nó thức dậy bất cứ khi nào các thành phần của AWT (tức là các thành phần GUI) kích hoạt các kiểu sự kiện khác nhau (ví dụ người dùng nhấn vào một nút). Vấn đề tiêu biểu cần tránh được phản ứng với một sự kiện AWT bằng cách chặn các sự kiện gửi đi cho tới khi một thông điệp ACL được nhận, hoặc cập nhật GUI từ luồng bên trong hoặc sửa đổi các biến không đồng bộ từ cả hai luồng. Bên dưới là một vài gợi ý về cách thực hành lập trình tốt để tránh một vài vấn đề này.

2.5.3.1. Thực hành lập trình tốt với bộ lắng nghe sự kiện AWT

Khi một tác tử có một GUI, nó cần phản ứng lại các hành động của người dùng, như là khởi đầu một hội thoại mới khi người dùng nhấn một nút. Khi sự kiện hành động AWT xảy ra, phương thức actionPerformed() được gọi bằng sự kiện luồng gửi đi trên ActionListener đã đăng ký nguồn gốc sự kiện. Bên trong phương thức này, một thực hành lập trình tốt là để chuẩn bị một đối tượng lịch trình JADE Behaviour để thực hiện bằng luồng sự kiện.

Các hành vi được lên lịch để thực hiện chỉ sau khi phương thức setup() của đối tượng tác tử đã được kết thúc. Hành vi luôn được thực hiện bởi các luồng tác tử. Kết quả là không có đồng bộ giữa các hành vi khác nhau được yêu cầu.

Tất nhiên, trong một vài trường hợp, thêm một hành vi là gánh nặng không cần thiết khi phản ứng đến hành động AWT phải được thay đổi một cách đơn giản giá trị của một biến hoặc chuẩn bị và gửi một ACLMessage (nhớ là phân phối thông điêpk là hoàn toàn không đồng bộ). Đây là tất cả các hoạt động hợp lệ cho luồng AWT, nói chung, không gây ra vấn đề. Ngược lại, ngăn chặn các cuộc gọi (ví dụ Agent.blockingReceive()) không bao giờ được thực thi trong luồng AWT.

2.5.3.2. Thực hành lập trình bằng cách sửa đổi giao diện đồ họa trong luồng thực thi của tác tử

Có ý kiến cho rằng tác tử có các luồng thực thi của riêng nó, các chuyên gia lập trình Java sẽ ngay lập tức suy ra rằng cập nhật GUI từ bên trong các luồng này có thể dẫn đến các vấn đề bất ngờ do các vấn đề đồng bộ hóa. AWT, Swing, MIDP (và hầu hết các framework giao diện người dùng khác) cung cấp một phương thức đặc biệt thích hợp xếp một đối tượng Runnable và khiến nó được thực hiện đồng bộ trên luồng sự kiện gửi đi GUI :

- java.awt.EventQueue.invokeLater() cho AWT
- javax.swing.SwingUtilities.invokeLater() cho Swing
- javax.microedition.lcdui.Display.callSerially() cho MIDP

2.6. Những đặc điểm nâng cao của JADE

2.6.1. Hợp các hành vi để xây dựng các tác tử

Như đã mô tả , các công việc trong JADE được thực hiện bằng cách xây dựng các lớp mở rộng của lớp jade.core.behaviours.Behaviour và cài đặt các phương thức action () và done (). Tuy nhiên, khi liên kết với các công việc phức tạp liên quan đến các bước tính toán, có thể pha trộn với các tác tử khác …thì điều này xem ra không thuận lợi. Chúng ta hãy xem xét ví dụ: hành vi BookNegotiator. Mặc dù nó chỉ đơn giản là trao đổi một vài thông điệp và lấy một quyết định, phương thức action() của nó khá là phức tạp. Một phương pháp đơn giản và rõ ràng hơn để thực hiện các nhiệm vụ phức tạp trong Jade là kết hợp các hành vi – tạo nhiệm vụ phức tạp từ các hành vi đơn giản. Cơ sở cho các đặc trưng này được cung cấp bởi lớp CompositeBehaviour trong gói jade.core.behaviours. Lớp này có các phương thức scheduleFirst() và scheduleNext() dùng để lập lịch các hành vi con. Những phương pháp này được khai báo trừu tượng và phải được định nghĩa trong các lớp con của CompositeBehaviour.

Ba kiểu hành vi kép được cung cấp trong JADE là SequentialBehaviour, FSMBehaviour và ParallelBehaviour.

2.6.1.1. Lớp SequentialBehaviour

Lớp SequentialBehaviour cài đặt một hành vi gộp để lập lịch các hành vi con theo một chính sách tuần tự đơn giản. Nó bắt đầu hành vi con đầu tiên, sau khi hoàn thành nó chuyển sang hành vi con tiếp theo và cứ như thế cho đến khi thực hiện hết

các hành vi con. Các hành vi con được add vào bằng phương thức addSubBehaviour(). Thứ tự được đưa vào chính là thứ tự chúng được lập lịch.

2.6.1.2. Lớp FsmBehaviour

Lớp FSMBehaviour cài đặt một hành vi gộp trong đó các hành vi con được lập lịch theo một máy hữu hạn trạng thái (FSM). Lớp FSMBehaviour cung cấp 2 phương thức để đăng ký các hành vi con làm các trạng thái của máy hữu hạn trạng thái và để đăng ký sự di chuyển giữa các trạng thái.

2.6.1.3. Lớp ParallelBehaviour

Lớp này cài đặt một hành vi gộp để lập lịch các hành vi con một cách song song. Thông thường, khi gặp phải các hành vi FSM, việc lập lịch có sự phối hợp và không ngừng. Nghĩa là mỗi khi phương thức action() của hành vi song song được thực thi, nó gọi phương thức action() của hành vi con hiện thời và sau đó chuyển con trỏ tới hành vi con tiếp theo mà không cần quan tâm đến việc nó đã hoàn thành hay chưa. Các hành vi con trong hành vi song song được thêm vao bằng cách gọi phương thức addBehaviour(). Một hành vi song song có thể kết thúc khi tất các hành vi con của nó hoàn thành, hoặc khi hành vi con đầu tiên hoàn thành.

2.6.1.4. Chia sẻ dữ liệu giữa các hành vi con: DATASTORE

Khi gộp các hành vi thành hành vi chuỗi, FSM hay song song thì thông thường hành vi con sẽ cần truy cập một số dữ liệu được tạo ra bởi các hành vi con khác. Tất nhiên, những dữ liệu này không thể được truyền vào như là tham số trong hàm khởi tạo của hành vi con vì tất cả các hành vi con đều thường được khởi tạo trước khi toàn bộ hành vi gộp được thực thi. Thông thường, khi các hành vi cần chia sẻ dữ liệu, cần sử dụng các biến thành viên của tác tử hoặc của hành vi gộp.

2.6.2. Hành vi luồng

Như đã giới thiệu, việc lập lịch hành vi được thực hiện theo một cách không ưu tiên. Đó là, phương thức action() của một hành vi không bao giờ bị ngắt để cho phép một hành vi khác nhảy vào. Chỉ khi phương thức action() của hành vi đang chạy trả về, việc điều khiển được truyền cho hành vi tiếp theo. Bất cứ hành vi Jade (đơn giản hay gộp) có thể được thực thi như một hành vi luồng bằng lớp jade.core.behaviours.ThreadedBehaviourFactory. Lớp này cung cấp phương thức wrap() để bao bọc hành vi jade thông thường vào một hành vi luồng wrapper. Có vài điểm quan trọng cần phải chú ý khi làm việc với hành vi luồng:

- Phương thức removeBehaviour() của lớp Agent không ảnh hưởng tới hành vi luồng. Một hành vi luồng bị xóa bằng việc lấy đối tượng Thread có nó bằng việc gọi phương thức getThread() của lớp ThreadedBehaviourFactory và gọi phương thức interrupt().
- Khi một tác tử chết, di chuyển hoặc tạm dừng, các hành vi luồng đang hoạt động của nó phần bị kill một các rõ ràng bằng việc sử dụng kĩ thuật đã mô tả ở trên.
- Nếu một hành vi con của hành vi song song (parallel) được cấu hình với chính sách kết thúc WHEN_ANY là hành vi luồng, việc kết thức các hành vi con khác không stop nó. Hành vi luồng con phải được kill một cách rõ ràng như mô tả ở trên.
- Khi một hành vi luồng truy xuất một vài tài nguyên tác tử, cái mà có thể được truy xuất bởi các hành vi luồng hoặc không luồng khác, việc quan tâm tính đúng đắn phải trả giá bằng việc đồng bộ.

2.6.3. Các giao thức tương tác

Ở giai đoạn này người đọc khá quen thuộc với ngôn ngữ FIPA-ACL được sử dụng bởi tác tử JADE để giao tiếp. Ngôn ngữ này cung cấp một tập các biểu diễn chuẩn, mỗi một biểu diễn như vậy được định một cách rõ ràng. Một trong những ưu điểm chính của đặc điểm này là khả năng chỉ ra chuỗi các thông điệp được định nghĩa trước có thể được áp dụng trong một vài hoàn cảnh chia sẻ cùng một kiểu giao tiếp mà không quan tâm miền ứng dụng. Một chuỗi các thông điệp này được biết đến như là các giao thức tương tác.

2.6.3.1. Gói jade.proto

Tất cả các lớp cung cấp hỗ trợ việc cài đặt các giao thức chuẩn trong JADE nằm trong gói jade.proto. Khi việc tham gia một phiên trao đổi được điều khiển bởi giao thức tương tác một tác tử có thể đóng vài trò là initiator (bên khởi tạo) hoặc responder (bên đáp ứng). Kết quả là các lớp trong gói jade.proto được chia thành initiator và responder.

2.6.3.2. Sử dụng các lớp giao thức

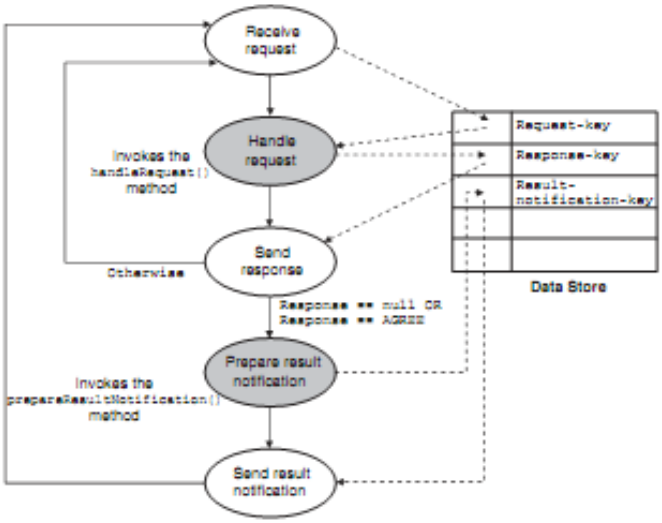
Như đã đề cập ở trên, các lớp giao thức cung cấp một số giao thức gọi lại. Những phương thức này có thể được lập trình viên sử dụng để định nghĩa lại bằng cách tùy biến chúng theo logic của miền ứng dụng. Chúng được khai báo để được

bảo vệ và có một cài đặt mặc định. Theo cách này, lập trình viên có thể chọn (tùy thuộc vào yêu cầu cụ thể) phương thức nào sẽ cài đặt và phương thức nào sẽ bỏ qua. Với cả bên khởi tạo và bên đáp ứng, phần lớn các phương thức gọi lại đều được gọi theo việc nhận thông điệp và có dạng :

protected handle<message-performative>(ACLMessage receivedMessage)

2.6.3.2. Lồng giao thức

Như đã trình bày trong các phần trước, cả lớp initiator và responder đều gọi các phương thức callback khi nhận được các thông điệp. Nếu phải gửi lại một thông điệp phản hồi thì phương thức callback có trách nhiệm tạo thông điệp đó. Tuy nhiên, có những trường hợp để tạo được thông điệp phản hồi, cần phải thực thi một hành vi. Rõ rang là việc này ngăn cản chúng ta sử dụng lớp ContractNetInitiator vì nó không thể thực thi một hành vi trong một phương thức.Để vượt qua giới hạn này, tất cả các lớp giao thức của JADE đều được cài đặt là các lớp con của lớp FSMBehaviour và mỗi phương thức callback được gọi trong một trạng thái của máy hữu hạn trạng thái. Hình 2.5 chỉ ra máy hữu hạn trạng thái của lớp AchieveREResponder.



Hình 2.5. Máy hữu hạn trạng thái của lớp AchieveREResponder

Bên cạnh các trạng thái dùng để gọi các phương thức callback, còn có các trạng thái khác có trách nhiệm gửi, nhận thông điệp và thực hiện các kiểm tra liên quan đến luồng giao thức. Tuy nhiên, chùng được ẩn đi và người lập trình không cần quan tâm đến. Đường nét đứt trong hình 2.5 chỉ ra cách dữ liệu được chia sẻ giữa các trạng thái của giao thức sử dụng DataStore.

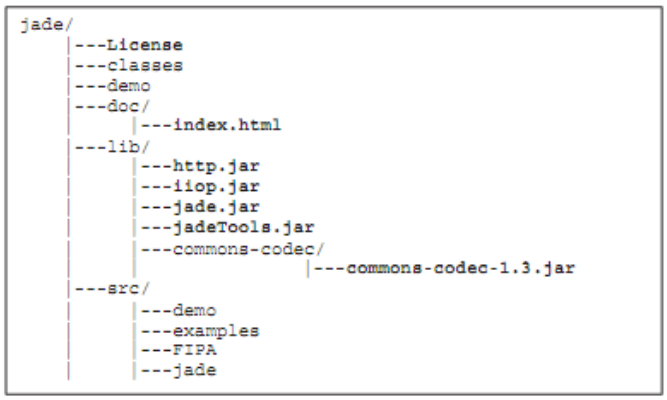
2.7. Biên dịch và chạy chương trình

Các phần mềm liên quan tới JADE được download từ trang web của JADE: http://jade.tilab.com. Phần mềm liên quan tới JADE chia thành hai mục: phân tán chính (main distribution) và tích hợp (add ons). Phần tích hợp gồm các modun tự chứa, cài đặt các đặc trưng mở rộng đặc tả. Sự phân tán chính gồm 5 file archive:

- jadeBin.zip chứa các file archive tiền biên dịch của JADE java đã sẵn sàng trong trạng thái sử dụng được.
- jadeDoc.zip chứa các tài liệu bao gồm các tài liệu hướng dẫn cho lập trình viên hoặc người quản trị. Tài liệu này cũng có trực tuyến trên trang web.
- jadeExamples.zip chứa các mã nguồn của các ví dụ khác nhau .
- jadeSrc.zip chứa tất cả các mã nguồn của JADE .
- jadeAll.zip chứa tất cả 4 file trên.

Các file zip trên được download và giải nén, có file/thư mục quan trọng như:

- Giấy đăng kí (license), giấy đăng kí mã nguồn mở, quy định cách sử dụng của phần mềm.
- File jade/doc/index.html là điểm bắt đầu tốt nhất cho những người bắt đầu làm quen với jade, gồm các liên kết (link) tới các chuyên đề (thematic tutorial), tài liệu hướng dẫn cho lập trình viên và người quản trị, tài liệu javadoc của tất cả các mã nguồn mở, cùng với nhiều tài liệu hỗ trợ khác.
- Thư mục jade/lib chứa tất cả các file .jar chứa trong CLASSPATH của java để có thể thực thi jade. Nó bao gồm thư mục con lib/commons- codec, chứa các mã 64bit có trong CLASSPATH của java.



Hình 2.6. Cấu trúc thư mục JADE

- Thư mục jade/src chức 4 thư mục con. Thư mục Demo chứa mã nguồn của các ví dụ đơn giản. Thư mục Examples chứa mã nguồn của nhiều ví dụ hữu ích theo các phân đoạn khác nhau về tác tử. Thư mục FIPA chứa mã nguồn của một mô đun được định nghĩa bởi FIPA. Thư mục Jade chứa tất cả các mã nguồn của chính nó.

Mã nguồn của jade có thể được biên dịch bằng cách sử dụng công cụ ANT. Các mục tiêu ANT quan trọng nhất là:

- Jade – biên dịch các mã nguồn và tạo các file .class trong thư mục con classes.
- Lib – tạo các file archive của java trong thư mục con lib.
- Doc – tạo các file tài liệu javadoc trong thư mục con doc.
- Example – biên dịch tất cả các ví dụ.

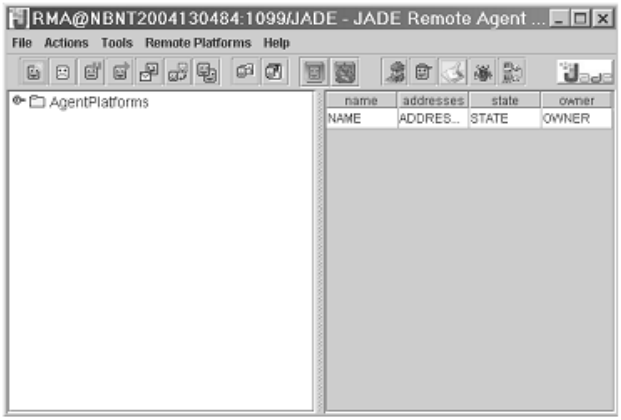
Thư mục lib chứa 5 file archive, trong các file đó có chứa lớp mà JADE cần:

- jade.jar chứa tất cả các gói jade ngoại trừ add ons, MTP và các công cụ đồ họa.
- jadeTool.jar chứa tất cả các công cụ đồ họa.
- http.jar chứa MTP dựa trên HTTP, nó là MTP mặc định khi platform được khởi tạo.
- iiop.jar chứa MTP dựa trên IIOP. MTP này không được sử dụng thường xuyên, nhưng là đối tượng nghiên cứu của các ví dụ sau này và nó cài đặt các đặc tả MTP IIOP FIPA.
- commons-codec/commons-codec-1.3.jar chứa các mã 64 bit được sử dụng bởi JADE.

Thư mục classes chứa các file class của các ví dụ. Sau khi download JADE và giải nén vào ổ C, ta tạo file runjade.bat ở ổ C với nội dung sau:

```
java -classpath
.;C:\jade\lib\jade.jar;C:\jade\lib\jadeTools.jar;C:
\jade\lib\iiop
.jar;C:\jade\lib\http.jar;C:\jade\lib\commons-
codec\commons-codec-1.3.jar jade.Boot -gui
```

Sau khi chạy file runjade.bat ta được kết quả như Hình 2.7

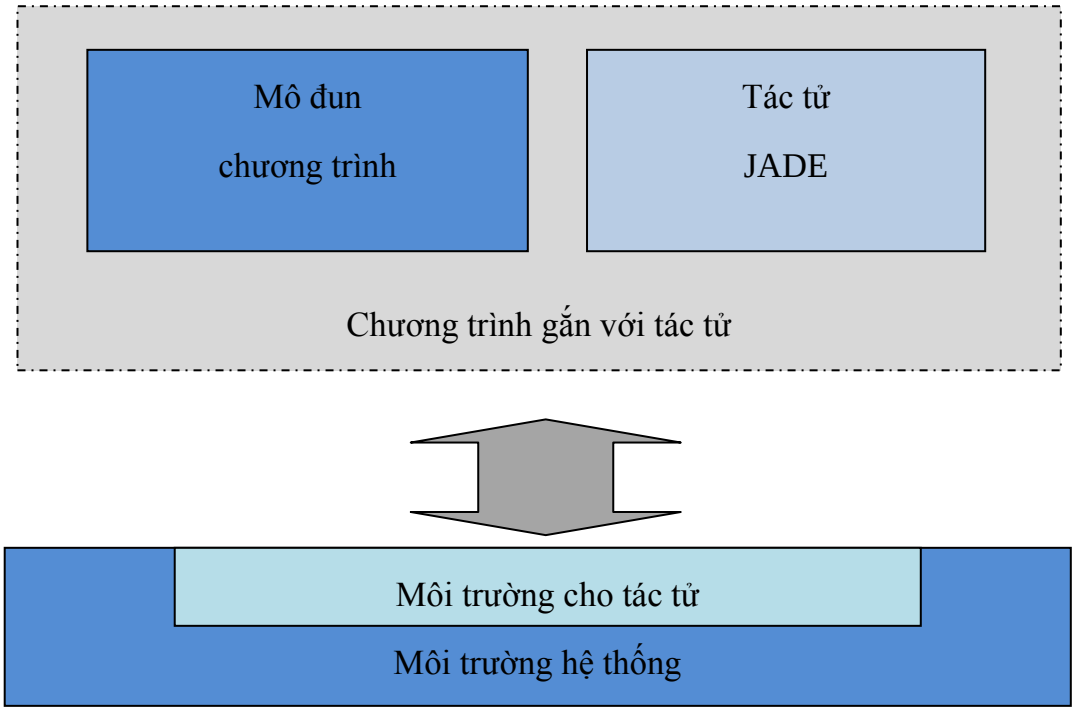


Hình 2.7. Giao diện của JADE RMA

CHƯƠNG 3: KIẾN TRÚC PHẦN MỀM DỰA TRÊN TÁC TỬ VÀ ỨNG DỤNG

Phần mềm máy tính đã trải qua quá trình phát triển dài, với nhiều tiến bộ trong công nghệ cũng như yêu cầu công việc thực tế, phần mềm ngày nay trở nên phức tạp và khó kiểm soát. Tuy nhiên, sự phát triển này chưa giải quyết được các bài toán phát sinh. Hướng tiếp cận xây dựng phần mềm dựa trên tác tử là cách tiếp cận khác để xây dựng các chương trình mang tính đặc thù.

3.1 Kiến trúc phần mềm dựa trên tác tử



Hình 3.1. Mô hình kiến trúc phần mềm dựa trên tác tử

Với cách tiếp cận truyền thống các mô đun chương trình, rộng hơn là các chương trình được thực thi trên môi trường của nó (platform). Do các trường hợp đặc biệt các chương trình này không đủ khả năng thực thi yêu cầu được người dùng đặt ra. Do đó các chương trình này được kết hợp với tác tử để tận dụng khả năng của tác tử đem lại.

Phần mềm dựa trên tác tử là một phần mềm thông minh, nó bao gồm cả các yêu cầu người dùng và kết hợp với tính ưu việt của tác tử. Vì thế phần mềm dựa trên tác tử không chỉ đáp ứng các yêu cầu người dùng mà còn bao gồm các tính năng của tác tử đồng thời khắc phục được các khuyết điểm của các phần mềm thông thường khác. Tuy nhiên không phải chương trình nào cũng có thể gắn với tác tử,

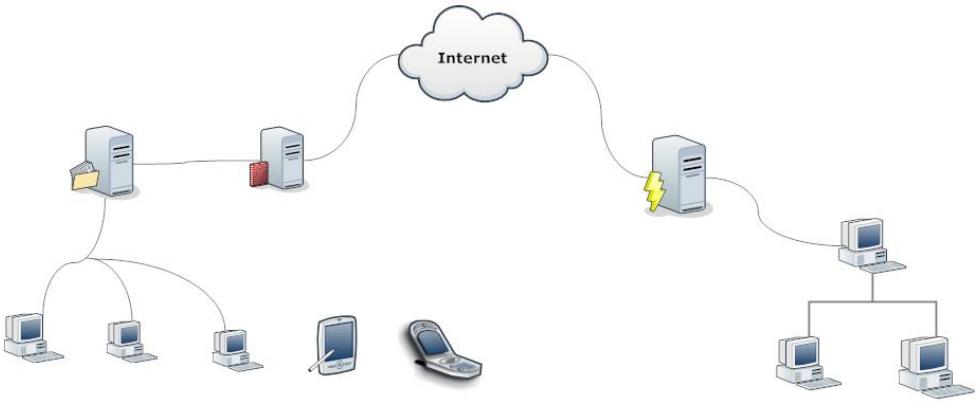
điều này phụ thuộc vào yêu cầu người dùng, hơn nữa, tác tử không phải là công nghệ vạn năng. Do vậy với những yêu cầu của bài toán cụ thể phần mềm sẽ được xây dựng trên nền tảng cụ thể.

3.2 Thực nghiệm

Trong phần này đồ án trình bày thực nghiệm xây dựng chương trình tra cứu điểm sinh viên dựa trên tác tử. Bài toán với quy mô nhỏ nhưng mô tả tổng quát được việc xây dựng phần mềm dựa trên tác tử.

Bài toán

Xây dựng chương trình dựa trên tác tử để tra cứu điểm sinh viên.



Hình 3.2. Mô hình bài toán ứng dụng tác tử

Trong bài toán này các công việc được chi thành các tiến trình như sau:

+ Xây dựng chương trình theo thiết kế

+ Xây dựng các tác tử để gắn với các mô đun trong chương trình.

+ Biên dịch với JADE

+ Thực thi trên môi trường JADE.

Xây dựng các mô đun trong chương trình

Mô đun kết nối cơ sở dữ liệu điểm

```
public class ConnectODBC_DSN {
//các mã chương trình của lớp Connection.
}
```

1. Xây dựng class ConnectODBC_DSN :

```
public class ConnectODBC_DSN {
      private Connection con=null;
      public ConnectODBC_DSN() throws Exception{
      String url="sun.jdbc.odbc.JdbcOdbcDriver";
      Class.forName(url);
      String dbUrl="jdbc:odbc:tracuudiem";
      con=DriverManager.getConnection(dbUrl);
      }
      public  ResultSet  GetResultSet(String  tableName)throws
SQLException {
            ResultSet rs=null;
            Statement stmt=con.createStatement();
            String sql="select * from "+tableName;
            rs=stmt.executeQuery(sql);
            return rs;
            }
      public void Close()throws Exception{
            con.close();
            }
      public static String Masinhvienkt()
      {
            Scanner msv = new Scanner(System.in);
            System.out.println("Nhap ma sinh vien: ");
            String ma = msv.next();
            return ma;
      }
}
```

2. Xây dựng class TracuudiemAgent:

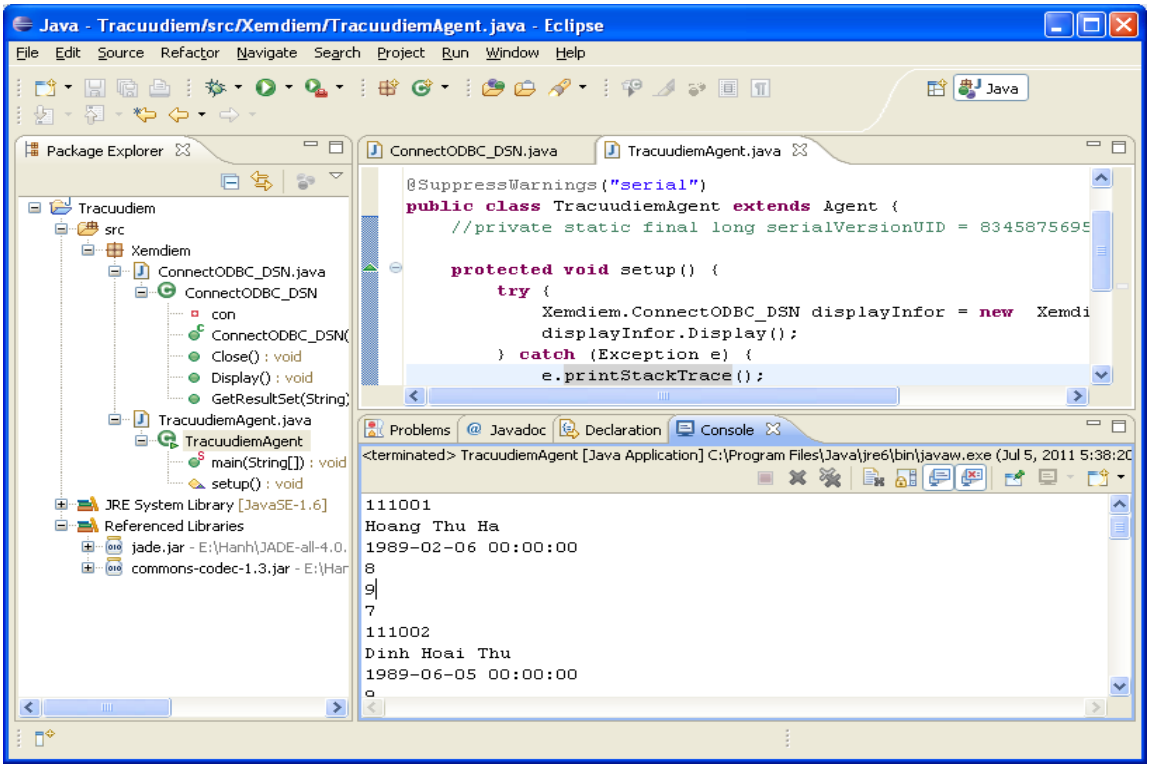
public class TracuudiemAgent extends Agent {

//Các phương thức và hành vi của tác tử.

}

```
public class TracuudiemAgent extends Agent {
 protected void setup() {
       try {
            ConnectODBC_DSN conDSN=new ConnectODBC_DSN();
            ResultSet rs=conDSN.GetResultSet("BANGDIEM");
            while(rs.next())
                 {
System.out.println(rs.getString("MaSinhVien"));
System.out.println(rs.getString("HoTen"))
System.out.println(rs.getString("NgaySinh"));
System.out.println(rs.getString("DiemMon1"));
System.out.println(rs.getString("DiemMon2"));
System.out.println(rs.getString("DiemMon3"));
                 }
conDSN.Close();
} catch (Exception e) {
e.printStackTrace();  }
}
public static void main(String[] args) {
 TracuudiemAgent Tracuu = new TracuudiemAgent();
Tracuu.setup();
 }
```

Hình ảnh chương trình thực nghiệm



Hình 3.3. Hình ảnh chương trình thực nghiệm

3.3. Biên dịch tác tử

Trong thư mục Tracuudiem ta tạo các thư mục và file sau:

- thư mục classes để lưu file sau khi biên dịch tác tử.
- thư mục src chứa nguồn là file .java
- file compilejade.bat với nội dung :

javac -classpath e:\jade\lib\jade.jar;e:\jade\lib\commons-codec\commons-codec-1.3.jar;. %1 %2 %3 %4 %5 %6 %7 %8 %9 -d e:\jade\src\examples\tracuudiem

- file runjade.bat với nội dung :

java-classpathclasses;e:\jade\lib\jade.jar;e:\jade\lib\commons-codec\commons-codec-1.3.jar jade.Boot -gui %1 %2 %3 %4 %5 %6 %7 %8 %9

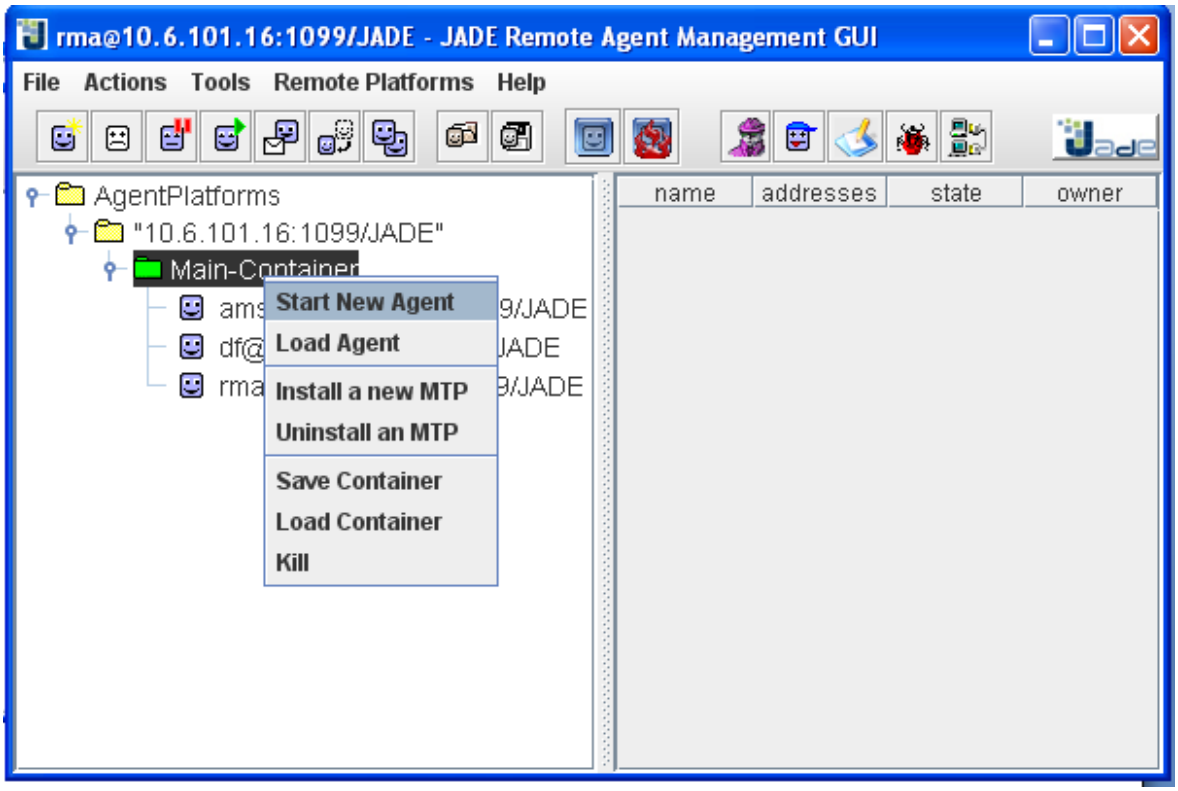
Trong cmd ta gõ dòng lệnh để tìm đường dẫn đến thư mục

1 : e:\>cd jade\src\examples\tracuudiem

2 : e:\>compilejde src*.java

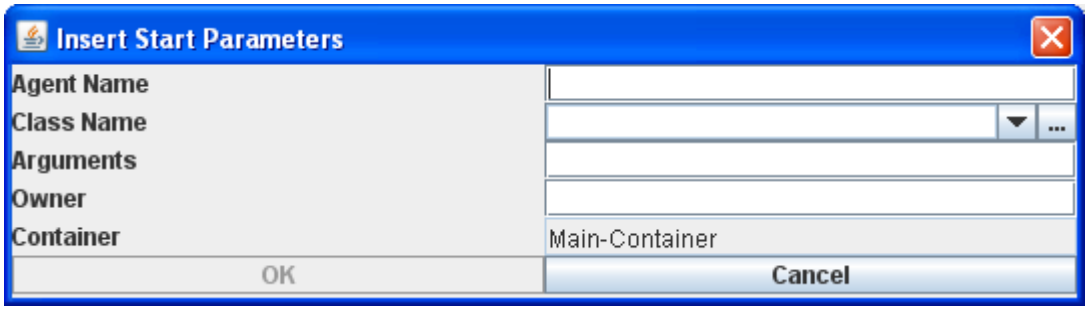
3.4. Gắn tác tử với Jade

- Sau khi biên dịch tác tử ta thực hiện gắn tác tử với JADE bằng câu lệnh trong cmd : runjade.
- Ở mục Main-Container ta kích chuột phải và chọn Start New Agent



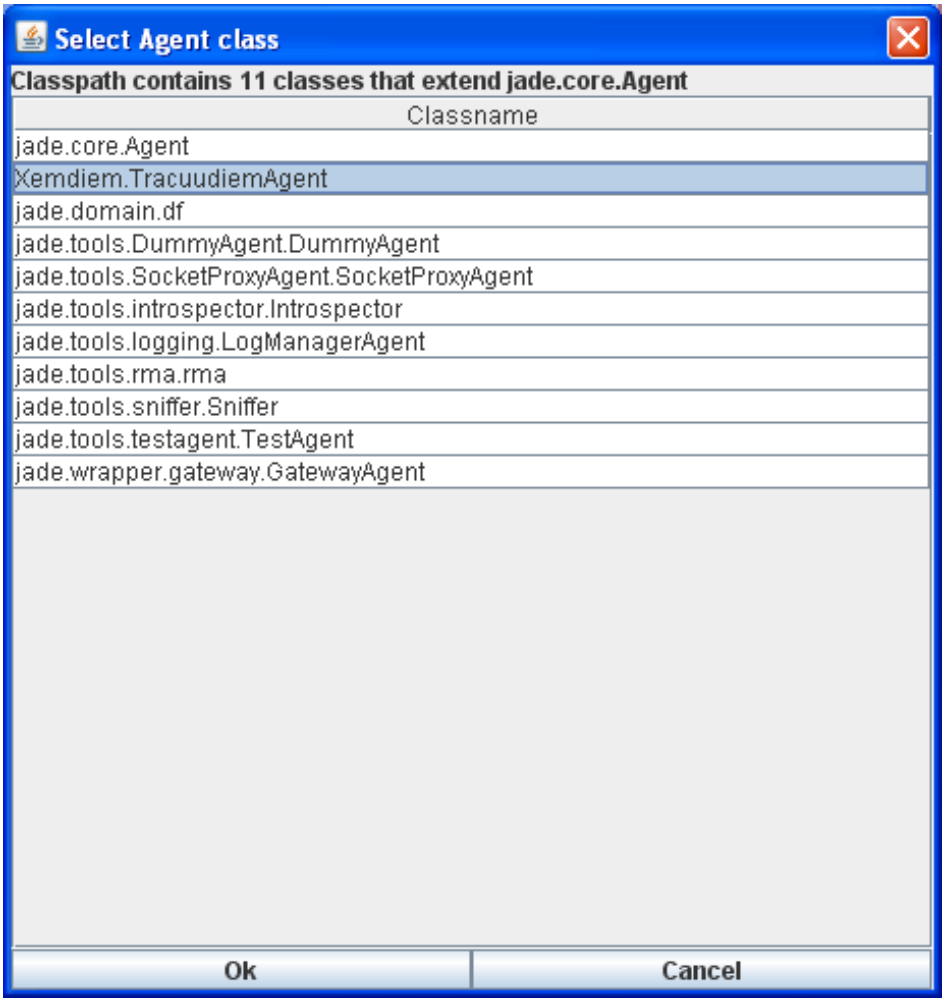
Hình 3.4. Kết quả của thao tác biên dịch tác tử

- Lúc đó hộp hội thoại sau sẽ xuất hiện :



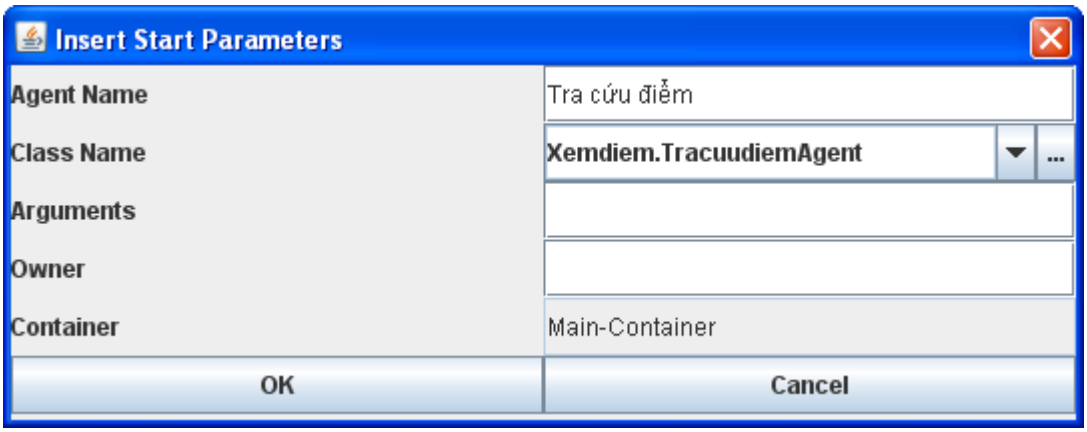
Hình 3.5. Tìm tới tác tử vừa tạo

- Khi đó ta chọn trong Class Name và chọn Xemdiem.Tracuudiemgent :



Hình 3.6. Kết quả của thao tác tạo một tác tử mới

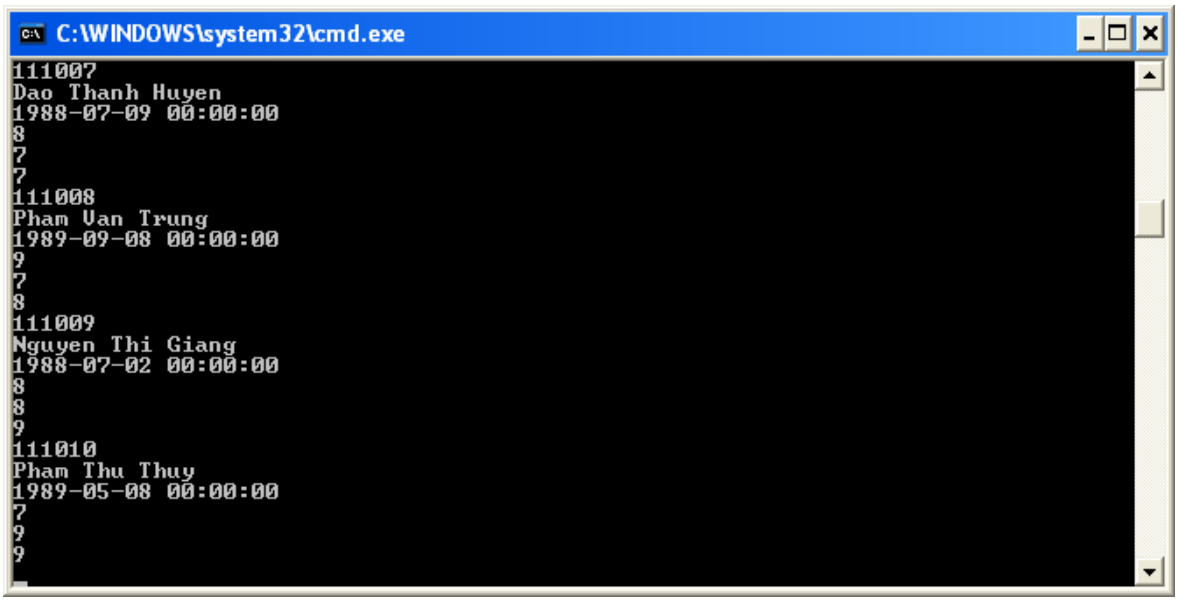
- Ấn nút OK
- Trong mục Agent Name ta điền tên như trong hình :



Hình 3.7. Điền thông tin

- Nhấn OK

- Hình ảnh kết quả trên DOS :



Hình 3.8. Kết quả chạy trên DOS

KẾT LUẬN

Qua thời gian tìm hiểu về tác tử và phần mềm dựa trên tác tử, đặc biệt là trong quá trình thực hiện đồ án tốt nghiệp, em đã nắm rõ được cách phân tích và xây dựng phần mềm dựa trên tác tử và áp dụng vào bài toán thực tế : Xây dựng chương trình dựa trên tác tử để tra cứu điểm sinh viên. Những kết quả chính mà đồ án đã đạt được có thẻ tổng kết như sau:

- Tìm hiểu chung về tác tử, hệ đa tác tử và xây dựng phần mềm dựa trên tác tử, một hướng phát triển kiến trúc phần mềm mới và hữu ích; những ứng dụng của tác tử, tác tử di động với các hệ thống phân tán, mở, phức tạp được phát triển cùng với sự lớn mạnh của Internet ngày nay.
- Phân tích, thiết kế, xây dựng chương trình dựa trên tác tử.
- Áp dụng vào bài toán xây dựng chương trình dựa trên tác tử để tra cứu điểm sinh viên.

Tuy nhiên chương trình có tính chuyên nghiệp chưa cao,chưa giải quyết được trọn vẹn nhữngvấn đề phát sinh,chưa đạt được tính thẩm mỹ cao.

Trong thời gian qua em đã giành nhiều thời gian và công sức cho đồ án tốt nghiệp này, qua quá trình tìm hiểu lý thuyết và xây dựng ứng dụng thực nghiệm em đã học được nhiều kiến thức và kinh nghiệm thực tế quý giá.

Ứng dụng và thực nghiệm đã chứng minh tính khả thi của việc xây dựng chương trình phần mềm dựa trên tác tử và ứng dụng vào thực tế, tạo tiền đề cho việc xây dựng phần mềm lớn và phức tạp hơn khi dựa trên tác tử là hoàn toàn khả thi và hữu ích. Đây sẽ là một hướng phát triển phần mềm đầy triển vọng và mang lại nhiều lợi ích trong tương lai.

TÀI LIỆU THAM KHẢO

[1] Fabio Luigi Bellifemine, Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology).

[2] Gerhard Weiss, MultiAgent Systems, A Modern Approach to Distributed Modern Approach to Artificial Intelligence.

[3] Pattie maes, Communication of the ACM, Agents that Reduce Word and Information Overload.

[4] White, JE. Telescrip Technology: Mobile Agent. In Bradsshaw jeffrey, (ed), Software Agent, AAAI Press/MIT Press, 1996.

[5] Rodney A. Books – Subsumption Architecture.

[6] Michael Georgeff, Barney Pell, Martha Pollack, Milind Tambe, MichaelWooldridge, The Belief – Desries – Intention Model of Agency.

[7] http://fipa.org

[8] http://jade.tilab.com/