BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC DÂN LẬP HẢI PHÒNG
-------o0o-------

MÔI TRƯỜNG PHÁT TRIỂN TÁC TỬ DI ĐỘNG VÀ ỨNG DỤNG

ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC HỆ CHÍNH QUY

Ngành: Công nghệ Thông tin

Sinh viên thực hiện: Vũ Thái Sơn
Giáo viên hướng dẫn: Ths. Nguyễn Trịnh Đông
Mã số sinh viên: 110891

HẢI PHÒNG - 2011

MỤC LỤC

DANH MỤC HÌNH ẢNH

MỞ ĐẦU

Tác tử di động (Mobile Agent) là phương thức giao tiếp tiên tiến đang ngày càng chứng tỏ được ưu thế so với mô hình truyền thông báo – cơ sở của hầu hết các giải thuật phân tán trước đây. Các tiến trình thay vì phải gắn liền với nơi khởi tạo có thể di chuyển đến bất kỳ đâu trong quá trình thực hiện, chúng có thể xích lại gần nhau để tương tác trực tiếp chứ không phải trao đổi từ xa thông qua các thông điệp.

Đồ án này nghiên cứu về tác tử (Agent), tác tử di động (Mobile Agent), hệ đa tác tử (Multi Agent), sự tương tác giữa các tác tử và sự thương lượng cũng như các giao thức truyền thông giữa chúng. Đồ án cũng nghiên cứu về ứng dụng của tác tử di động để xây dựng ứng dụng về trao đổi giữa các tác tử thông qua bài toán "Book Trading".

Đồ án được trình bày như sau**:**

Chương 1: Tác tử, tác tử di động và đa tác tử

Chương này giới thiệu tác tử, tác tử di động và hệ thống đa tác tử, đưa ra những ưu điểm của tác tử so với giải thuật phân tán trước đây và nêu ra các lĩnh vực ứng dụng áp dụng nó.

Chương 2: Sự tương tác của tác tử

Đề cập đến sự tương tác giữa các tác tử với nhau để thực hiện vai trò của mình, nghiên cứu sự tương tác của các tác tử dựa vào vai trò của chúng đồng thời nghiên cứu cơ chế truyền thông giữa các tác tử.

Chương 3: Môi trường, nền tảng phát triển tác tử di động

Chương này nhằm giới thiệu một số môi trường và nền tảng để phát triển tác tử di động, nêu ra các tính năng hỗ trợ của hệ thống tác tử di động.

Chương 4: Nền tảng JADE (Java Agent DEvelopment Framework)

Cung cấp một cái nhìn tổng quan về nền tảng JADE và các thành phần chính tạo thành kiến trúc của nó, nêu ra cơ chế hoạt động và các hoạt động của tác tử trên nền tảng JADE.

Chương 5: Thực nghiệm

Đưa ra mô tả bài toán ứng dụng dựa trên công nghệ tác tử và kết quả.

CHƯƠNG 1: TÁC TỬ, TÁC TỬ DI ĐỘNG VÀ ĐA TÁC TỬ

1.1 Giới thiệu về tác tử

1.1.1 Khái niệm tác tử (Agent)

Từ trước đến nay, khi muốn máy tình làm một công việc gì, thì người ta luôn phải xác định trước mục tiêu, sau đó thiết kế và mã hóa chương trình. Và khi gặp các tình huống không xác định, máy tính thường hay gặp sự cố. Tuy vậy trong những năm gần đây, với sự phát triển của Internet đã dẫn đến việc ứng dụng công nghệ thông tin một cách rộng rãi vào nhiều lĩnh vực khác nhau như: tìm kiếm thông tin, quản lý, giám sát mạng viễn thông, … Sự đa dạng của việc áp dụng khiến cho việc phát triển phần mềm ngày càng trở nên phức tạp hơn, thể hiện ở chỗ:

- Khối lượng xử lý công việc ngày càng lớn.
- Yêu cầu về tính chính xác ngày càng lớn.
- Yêu cầu về tính mở và phân tán: ngày nay hầu hết các hệ thống thông tin đều gắn bó chặt chẽ với môi trường mạng và do đó phần mềm cũng phải đáp ứng ngày càng tốt hơn các nhu cầu của con người.
- Yêu cầu về tính độc lập giữa các thành phần của hệ thống.

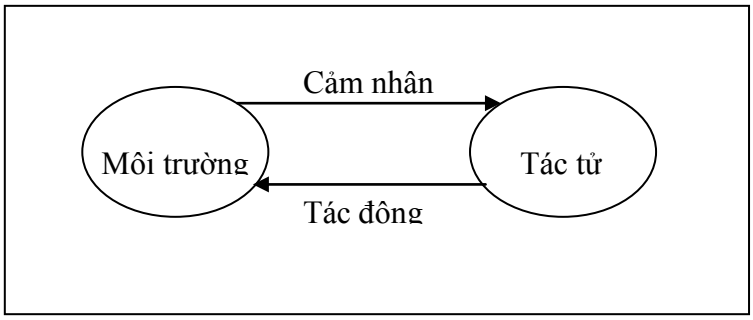
Hướng phát triển và nghiên cứu mạng mẽ công nghệ phần mềm những năm gần đây chuyển từ tiếp cận hướng cấu trúc sang tiếp cận hướng đối tượng và tập ký hiệu chuẩn mà hiện nay đang phát triển và ứng dụng rộng rãi. Tuy nhiên trong những hệ thống thông tin phức tạp thì vẫn còn nhiều hạn chế về tính thụ động của đối tượng, nghĩa là đối tượng chỉ hoạt động khi nhận được thông điệp từ đối tượng khác. Với các hệ thống có yêu cầu về tính phân tán như hệ tìm kiếm thông tin, hệ thương lượng trong thương mại điện tử, hệ quản lý, giám sát mạng viễn thông,… thì tương tác thụ động như vậy đã không còn phù hợp nữa. Các thành phần phần mềm trong hệ thống phải phục vụ các dịch vụ khác nhau, do đó cần phải chủ động theo các mục đích riêng của mình, đồng thời phải tương tác với các thành phần khác để chia sẻ tài nguyên hỗ trợ công việc.

Định nghĩa tác tử:

Có rất nhiều định nghĩa khác nhau, thậm chí khác nhau về tác tử, nhưng có một định nghĩa thường được sử dụng như sau:

Tác tử (Agent) là hệ thống tính toán hoạt động tự chủ trong một môi trường nào đó, có khả năng cảm nhận môi trường và tác động vào môi trường.

Có thể hiểu định nghĩa trên như sau: Hệ thống tính toán có thể là phần cứng, phần mềm, hoặc cả phần cứng lẫn phần mềm. Bất cứ tác tử nào cũng tồn tại và hoạt động trong một môi trường nhất định. tác tử nhận thông tin từ môi trường qua các cơ quan cảm nhận và tác động vào môi trường bằng các cơ quan tác động.



Hình 1.1: tác tử tương tác với môi trường

Đối với các tác tử phần cứng, cơ quan cảm nhận có thể là các cảm biến, camera, cơ quan tác động có thể là các bộ phận cơ học, quang học hoặc âm thanh. Đối với các tác tử là chương trình phần mềm, môi trường hoạt động thông thường là các máy tính hoặc mạng máy tính. Việc cảm nhận môi trường và tác động được thực hiện thông qua các lời gọi hệ thống. Nói chung, tác tử có thể được thiết kế để hoạt động để hoạt động trong nhiều dạng môi trường khác nhau. Một điểm cần chú ý là cảm nhận về môi trường của tác tử có thể không đầy đủ do môi trường quá phức tạp hoặc có chứa các yếu tố không xác định.

Một yêu cầu quan trọng đối với tác tử là tính tự chủ. Cũng như bản thân định nghĩa về tác tử, cũng có nhiều cách hiểu khác nhau về tính tự chủ. Ở đây, tự chủ được hiểu như là khả năng các tác tử hành động không cần đến sự can thiệp trực tiếp của người hay các tác tử khác: tác tử hoàn toàn có khả năng kiểm soát trạng thái cũng như hành vi của mình trong một thời gian tương đối dài. Một số các tác giả định nghĩa tính tự chủ rộng hơn, chẳng hạn yêu cầu tác tử phải có khả năng tự học.

Với đặc điểm tồn tại và hành động tự chủ trong môi trường, tác tử có thể thực hiện các mục tiêu cho trước và do vậy có thay thế chủ của mình (người dùng hoặc các tác tử khác) thực hiện một số các nhiệm vụ nào đó.

1.1.2 Đặc điểm, tính chất của tác tử

Một tác tử thông thường có những đặc điểm, tính chất sau:

- *Tính phản xạ*: tác tử có khả năng phản xạ kịp thời với các thay đổi trong môi trường mà tác tử cảm nhận được.
- *Tính chủ động (hành động có mục đích)*: không chỉ phản xạ, tác tử còn phải biết chủ động tìm kiếm khả năng hành động hướng tới thực hiện mục tiêu được giao.
- *Tính cộng đồng*: tác tử có khả năng tương tác với người dùng hoặc các tác tử khác để thực hiện nhiệm vụ của riêng mình hoặc để giúp đỡ các đối tác.
- *Tính thích nghi*: Thích nghi là khả năng của tác tử tồn tại và hoạt động hiệu quả khi môi trường thay đổi. Mặc dù có nhiều nét liên quan với tính phản xạ, khả năng thích nghi của tác tử khó thực hiện và đòi hỏi nhiều thay đổi trong quá trình suy diễn của tác tử hơn. Tính thích nghi có thể thực hiện nhờ khả năng tự học từ kinh nghiệm của tác tử.
- *Khả năng tự học*: Tự học hoặc học tự động là khả năng của tác tử thu thập các kiến thức mới từ kinh nghiệm thu lượm được, chẳng hạn qua các lần thành công và thất bại. Kết quả tự học phải làm cho các tác tử hành động tốt hơn, hiệu quả hơn.
- *Khả năng di chuyển*: Là khả năng của tác tử (phần mềm) di chuyển giữa các máy tính hoặc các nút khác nhau trong mạng đồng thời giữ nguyên trạng thái và khả năng hoạt động của mình. Các tác tử có đặc điểm này được gọi là tác tử di động. Việc thiết kế và cài đặt tác tử di động đặt ra các yêu cầu đặc biệt về vấn đề an ninh hệ thống.

Có thể so sánh một tác tử có đầy đủ ba đặc điểm trên cùng với một cầu thủ đá bóng. Mục đích của cầu thủ là cùng toàn đội đưa bóng vào lưới đối phương đồng thời ngăn không cho đối phương đưa bóng vào lưới mình. Để đạt được mục đích này, cầu thủ phải tìm mọi cơ hội để đưa bóng về gần lưới đối phương và sút. Đây chính là thể hiện của tính tự chủ hành động có mục đích. Tuy nhiên, tình huống trên sân có khi cầu thủ phải thay đổi mục tiêu tạm thời, cụ thể là chuyền ngang hoặc thậm chí chuyền về. Khi đối phương vào bóng thô bạo thì mục tiêu trước mắt có thể chưa phải là sút bóng mà trước hết là giữ an toàn cho mình. Đây là thể hiện rõ ràng của tính phản xạ. Cuối cùng, cầu thủ trên sân phải có tính cộng đồng, thể hiện với việc phối hợp với đồng đội, tuân theo các chỉ dẫn của huấn luyện viên và trọng tài.

1.1.3 Tác tử tĩnh

Tác tử tĩnh hoạt động như một phần tử độc lập trên một bộ xử lý (client hoặc server), nhưng có thể sinh hàng loạt các tác vụ khác hoạt động trên cùng một bộ vi xử lý hoặc trên các bộ vi xử lý khác. Tác tử tĩnh rất thích hợp với vai trò tìm kiếm và chọn lọc thông tin, điều khiển dòng dữ liệu, điều khiển thiết bị thông minh, lọc thư điện tử và tư vấn cho người sử dụng.

Tác tử tĩnh giữ nguyên vị trí trong quá trình nó hoạt động. Mặc dù có nhiều loại tác tử tĩnh khác nhau song do chúng không thể di chuyển, nên việc thiết kế các tác tử tĩnh trở nên đơn giản hơn tác tử động. Tác tử tĩnh không yêu cầu phải liên lạc với một cơ sở hạ tầng để có thể cho phép các tác tử khác chuyển qua hay hỗ trợ môi trường giao tiếp phức tạp giữa các tác tử. Tác tử tĩnh cũng không yêu cầu quá trình đồng bộ hóa trong quá trình hoạt động của nó.

Các loại tác tử tĩnh điển hình là tác tử giao tiếp, tác tử chức năng, tác tử điều khiển và giám sát, tác tử xử lý thư điện tử, tác tử thu thập thông tin,…

1.1.4. Tác tử di động (Mobile Agent)

1.1.4.1 Khái niệm

Tác tử di động là một một đối tượng di động tự trị, trong đó có mã lệnh, dữ liệu và trạng thái thực hiện, có khả năng di chuyển trên môi trường mạng phân tán, đại diện cho người sử dụng thực hiện một số công việc xác định.

1.1.4.2 Các tính chất của tác tử di động

Tác tử di động có ba tính chất:

Tính tương tác: tác tử có thể cảm nhận và hành động, thực hiện các công việc theo sự thay đổi của môi trường.

Tính tự trị: tác tử hoạt động thực hiện các công việc phức tạp mà không cần có sự can thiệp trực tiếp của người dùng. tác tử có thể thực hiện các hành động độc lập nhờ vào các kiểu trạng thái được cài đặt trước.

Tác tử cũng có khả năng kiểm soát nhất định đối với hành động và trạng thái bên trong của mình.

Tính di động: tác tử có thể linh hoạt tự di chuyển từ nơi này đến nơi khác, theo một hành trình định trước hoặc các tác tử nhận thức về môi trường và hành động dựa theo các tình huống khác nhau.

Cung cấp một hệ thống linh động là ưu điểm chính của tác tử di động. Nó đưa ra một khái niệm cơ sở của các mạng tích cực mà ở đó, mã dịch vụ thường được đặt ở bên ngoài mạng thì nó sẽ được chuyển đến các node chuyển mạch của mạng một cách linh động. Có 2 phương thức để thực hiện việc chuyển động này và chúng đều cung cấp cho mạng một cấu trúc linh hoạt và năng động. Nếu dùng phương pháp tích hợp thì các gói dữ liệu truyền đi chứa các đoạn chương trình được đọc và kích hoạt bởi các node chuyển mạch có khả năng lập trình. Ngược lại, theo phương pháp tiếp cận rời rạc thì mạng sẽ di chuyển mã một cách rời rạc.

Ưu điểm của tác tử di động có thể nói gọn trong cụm từ *Ưu điểm về vị trí* . Khi một tác tử di động di chuyển đến và định cư ở máy trạm đầu xa, chúng tránh được việc trao đổi thông tin với các máy trạm bằng cách định cư ở đó.

Các tác tử di động có thể giúp làm giảm lưu lượng trên mạng thay vì phải truyền một lượng lớn thông tin trên mạng thì chúng ta chỉ cần truyền chức năng của chúng.

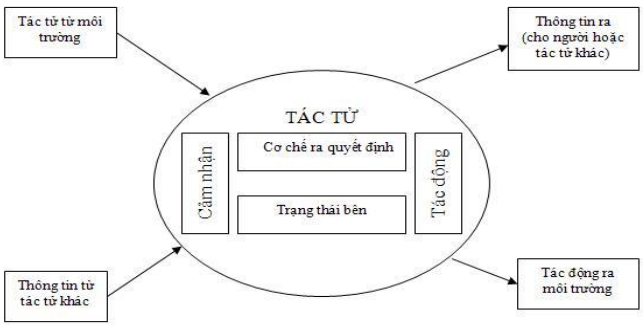
Các tác tử di động cũng có thể xử lý trước dữ liệu và chỉ truyền các kết quả. Còn được gọi là nén ngữ nghĩa.

Ngoài việc giảm lưu lượng trên mạng thì thời gian kích hoạt cũng giảm do sự di chuyển và định cư của tác tử.

1.1.5 Cấu trúc chung của một tác tử

Có thể xem xét tác tử từ hai góc độ: như một thực thể trừu tượng với cơ chế suy diễn quyết định riêng; như một thành viên trong cộng đồng tác tử với các mối quan hệ, tương tác với thành viên khác tức là ta đi xem xét tác tử như một thực thể riêng và hệ đa tác tử.

Ở mức độ tổng quát, tác tử có kiến trúc như hình vẽ sau:



Hình 1.2: Cấu trúc chung của tác tử

Từ hình vẽ ta thấy, tác tử nhận thông tin từ môi trường (bao gồm thông tin từ các tác tử khác) thông qua cơ quan cảm nhận. Nhờ có cơ chế ra quyết định, tác tử lựa chọn hành động cần thực hiện. Quá trình ra quyết định có thể sử dụng thông tin về trạng thái bên trong của tác tử. Trong trường hợp đó, tác tử lưu trữ trạng thái dưới dạng những cấu trúc dữ liệu riêng. Hành động do cơ chế ra quyết định lựa chọn sau đó được tác tử thực hiện thông qua cơ quan tác động.

Cơ chế suy diễn có thể thay đổi cho từng kiểu kiến trúc cụ thể và ảnh hưởng tới những thành phần khác. Chẳng hạn có thể có kiến trúc trong đó quá trình suy diễn không sử dụng tới trạng thái bên trong và do vậy tác tử không cần lưu giữ các thông tin này. Đối với các tác tử có thêm khả năng khác như học tự động, kiến trúc tác tử có thể có thêm thành phần riêng để thực hiện các chức năng này.

1.1.5.1 Cơ chế cảm nhận

Cơ chế cảm nhận cho phép tác tử biết được những gì đang diễn ra xung quanh, từ đó ra quyết định và hành động phù hợp. Đối với người và động vật, quá trình cảm nhận được thực hiện qua những giác quan, còn với tác tử phần cứng như robot cơ quan cảm nhận là cảm biến, camera. Tác tử phần mềm, quá trình cảm nhận có thể diễn ra một cách chủ động bằng cách trao đổi thông điệp với các tác tử khác hoặc thụ động thông qua nhận và xử lý các sự kiện hoặc thông điệp do hệ điều hành gửi tới.

1.1.5.2 Cơ chế tác động

Tác động là quá trình ngược với cảm nhận. tác tử tiến hành tác động vào môi trường khi cơ chế suy diễn và ra quyết định chọn được hành động cần thiết.

1.1.5.3 Cơ chế ra quyết định

a. Mô hình chung

Một cách tổng quát, quá trình ra quyết định của tác tử có thể mô tả như sau. Giả sử thời gian được phân chia thành những thời điểm rời rạc $t_0, t_1, \ldots$. Tại mỗi thời điểm, tác tử phải lựa chọn một hành động từ tập hợp hữu hạn các hành động. Nhờ cơ quan cảm nhận, tác tử thu được những cảm nhận về môi trường. Giả sử tại thời điểm $t_1, t_2, \ldots$ cảm nhận của tác tử về môi trường lần lượt là $p_0, p_1 \ldots$ với $p_i \in p$, p là tập các cảm nhận có thể có của tác tử. Tại mỗi thời điểm t_i, tất cả những gì tác tử cảm nhận về môi trường cho tới thời điểm đó là chuỗi các cảm nhận si=< p_0 , p_1 ,… p_i >.

Giả sử tác tử có thể thực hiện một số hành động nhất định. Gọi tập hợp các hành động mà tác tử có thể thực hiện là A={a,a'….}.

Tại mỗi thời điểm t_i, tác tử có thể lựa chọn hành động $a_i \in A$ để thực hiện. tác tử lựa chọn hành động cụ thể tuỳ thuộc vào chuỗi cảm nhận s_i tại thời điểm đó. Như vậy, mỗi tác tử được đặc trưng bởi một ánh xạ từ chuỗi cảm nhận sang hành động.

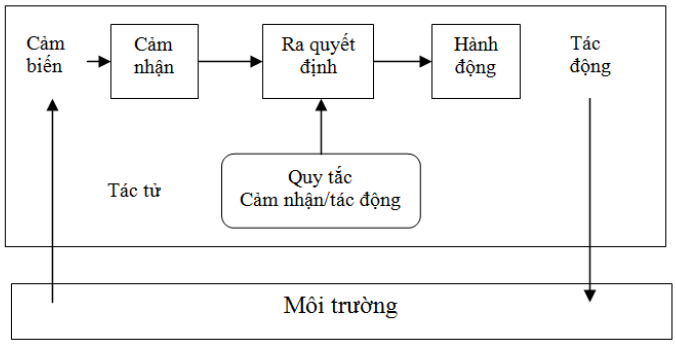
Nói cách khác, bằng cách chỉ rõ hành động mà tác tử thực hiện với chuỗi cảm nhận, ta có thể xác định cơ chế ra quyết định cho một tác tử cụ thể. Ánh xạ S→A có thể cho bằng nhiều cách. Cách đơn giản nhất là xây dựng một bảng liệt kê tất cả những chuỗi cảm nhận có thể cảm nhận và hành động tương ứng. Tuy nhiên, trong đa số trường hợp, kích thước của bảng rất lớn do số lượng chuỗi cảm nhận là vô hạn trừ khi ta hạn chế độ dài của chuỗi đó, chẳng hạn bằng cách giới hạn vòng đời của tác tử. Do vậy, phương pháp liệt kê bảng hành động là không thực tế do đòi hỏi quá nhiều bộ nhớ và thời gian tìm kiếm trong bảng.

b. Tác tử phản xạ

Trong một số trường hợp, tác tử có thể hành động dựa trên cảm nhận hiện thời mà không cần quan tâm đến chuỗi cảm nhận trước đó. Bộ điều nhiệt đơn giảnlà một tác tử như vậy. Dựa trên nhiệt độ đo được tại mỗi thời điểm, trạng thái môi trườn được chia thành "nóng" hoặc "bình thường". Bộ điều nhiệt quyết định bật lò sưởi nếu nhiệt độ là "nóng" và tắt lò sưởi nếu ngược lại. Quyết định này căn cứ trên nhiệt độ hiện thời và không phụ thuộc vào những thời điểm trước đó. Hành động của tác tử là phản ứng đối với trạng thái hiện tại của môi trường.tác tử có hành động chỉ phụ thuộc vào cảm nhận hiện thời mà không phụ thuộc vào cảm nhận trong quá khứ gọi là *tác tử phản xạ*.

Đối với tác tử phản xạ, ánh xạ S→A trở thành ánh xạ từ tập cảm nhận vào tập hành động.

Tác_tử: P→A ánh xạ này có thể xác định bởi các quy tắc có dạng "cảm nhận/hành động", hay thường được viết dưới dạng "**nếu** cảm nhận P **thì** hành động A".



Hình 1.3: Sơ đồ tác tử phản xạ

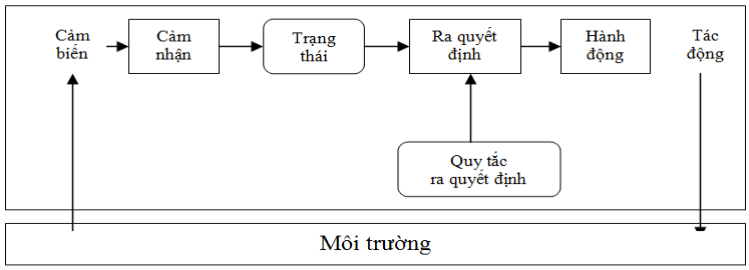
Mặc dù tác tử hoạt động hoàn toàn theo cách phản xạ được coi là tương đối đơn giản, ngay cả các tác tử phức tạp cũng có thể có một phần hành vi xác định bởi cơ chế cảm nhận/hành động như vậy. Ví dụ người và động vật đều có những phản xạ được hình thành do luyện tập hoặc là phản xạ tập trung. Các phản xạ này cho phép hành động nhanh chóng trong một số trương hợp mà không cần mất thời gian để cân nhắc kỹ càng. Khi một người bình thường chạm tay vào một vật nóng, phản xạ tự vệ thông thường là rụt ngay tay lại thì vì suy nghi kỹ càng xem phải làm gì. Phản xạ chính là cách ra quyết định nhanh chóng như vậy.

c. Tác tử có trạng thái

Trong nhiều trường hợp, cảm nhận hiện thời là chưa đủ để tác tử ra quyết định phải hành động như thế nào. Lý do khiến cảm nhận hiện thời chưa đủ quyết định hành động là do tại mỗi thời điểm cơ quan cảm nhận không thể cung cấp đủ toàn bộ thông tin về trạng thái môi trường xung quanh. Để có thể hình dung được toàn thể về môi trường, tác tử phải sử dụng thông tin từ những cảm nhận trước đó. Thông tin này cho phép phân biệt những trạng môi trường khác nhau nhưng lại sinh ra cùng một cảm nhận ở thời điểm hiện tại.

Tác tử ghi lại thông tin về môi trường bằng cách lưu lại chuỗi các cảm nhận cho tới thời điểm hiện tại. Tuy nhiên, nếu chuỗi cảm nhận dài thì việc lưu lại là không hiệu quả. Thay vào đó, tác tử duy trì một cấu trúc thông tin gọi là *trạng thái bên trong*. Việc lựa chọn hành động sẽ dựa trên những thông tin chứa trong trạng thái bên trong của tác tử.

Sơ đồ của tác tử với trạng thái bên trong được minh hoạ như hình vẽ dưới đây:



Hình 1.4: tác tử có trạng thái bên trong

d. Tác tử hành động có mục đích

Thông thường, tác tử được tạo ra để thực hiện một nhiệm vụ nào đó, và để tác tử thực hiện đúng nhiệm vụ của mình thì thông tin về trạng thái môi trường là chưa đủ. tác tử cần có thông tin về nhiệm vụ, mục đích hoạt động. Và để tác tử hoạt động có mục đích là xây dựng sẵn chương trình hành động và yêu cầu tác tử hành động theo các bước ghi trong chương trình đó. Tuy nhiên, cách này có một nhược điểm là cứng nhắc theo những gì đã được sắp đặt trước vì nếu môi trường thay đổi không phù hợp với điều kiện được xây dựng trước thì tác tử sẽ không thể thích nghi với thay đổi đó. Để đảm bảo tính mềm dẻo, ta chỉ thông báo cho tác tử mục đích cần đạt được thay vì cho tác tử biết phải làm thế nào để đạt được mục đích đó.

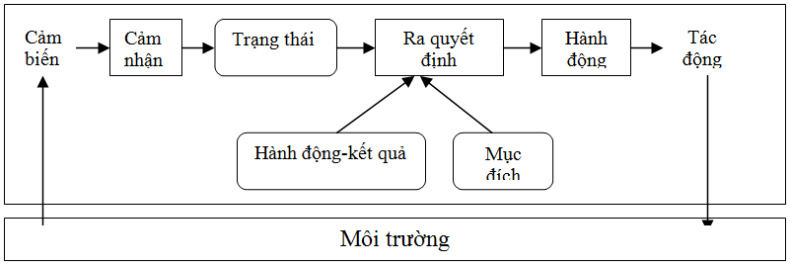
Mục đích thường có hai dạng:

- Đạt được trạng thái nào đó
- Đảm bảo duy trì trạng thái nào đó

Ở dạng thứ nhất, mục đích được cho dưới dạng một số trạng thái cần đạt, tác tử cần hành động sao cho môi trường chuyển sang một trong những trạng thái đó. Ví dụ, đối với tác tử làm nhiệm vụ chơi cờ vua, mục đích cần đạt được là bất cứ thế cờ nào chiếu hết đối phương.

Ở dạng thứ hai, tác tử được yêu cầu tránh một số trạng thái cho trước.

Thông tin mục đích được kết hợp với thông tin về kết quả hành động để lựa chọn hành động cho phép đạt được mục đích. Việc ra quyết định trong trường hợp này khác với ra quyết định theo kiểu phản xạ ở chỗ nó đòi hỏi thông tin về tương lai: hành động nào sẽ dẫn đến kết quả nào. Quá trình ra quyết định dựa trên thông tin về mục đích đòi hỏi khối lượng tính toán lớn hơn so với phản xạ nhưng lại mềm dẻo hơn rất nhiều.



Hình 1.5: tác tử có mục đích

Trong những trường hợp đơn giản, mục đích có thể đạt được sau một hành động duy nhất. Nhưng thường thì việc đạt được mục đích là phức tạp và đòi hỏi một chuỗi hành động.

Mặc dù có tốc độ ra quyết định chậm hơn so với phản xạ thuần tuý do quá trình lập kế hoạch đòi hỏi thời gian, hành động có mục đích cho phép ra quyết định mềm dẻo hơn nhiều.

e. Tác tử với cơ chế suy diễn logic

Nhiều nghiên cứu về trí tuệ nhân tạo cho rằng hành vi thông minh có thể có được bằng cách biến đổi thông tin môi trường và mục đích của tác tử về tính biểu tượng, sau đó thực hiện các biến đổi cần thiết trên những biểu tượng đó. Cách tiếp cận này được gọi là trí tuệ nhân tạo biểu tượng và được sử dụng trong một số nghiên cứu để xây dựng các tác tử thông minh. Hệ thống biểu tượng thông dụng nhất để mô tả môi trường và các biểu thức lôgic. Phần này sẽ trình bày về tác tử với mô hình môi trường được cho dưới dạng biểu thức logic và cơ chế sử dụng phép biến đổi trên mô hình đó dưới dạng suy diễn logic.

Như với mọi hệ thống sử dụng trí tuệ nhân tạo sử dụng biểu tượng khác, để xây dựng tác tử suy diễn cần giải quyết hai vấn đề:

Biến đổi thông tin về dạng biểu tượng: Thông tin về môi trường phải được biến đổi thành mô hình có tính biểu tượng, ở đây là biểu thức logic, cần thiết cho quá trình ra quyết định trong khoảng thời gian đủ ngắn để mô hình đó không bị cũ. Thông thường, một phần mô hình môi trường như các quy luật biến đổi và mục đích của tác tử được xây dựng từ trước, một phần khác được cập nhật trong quá trình tác tử hoạt động và cảm nhận.

Biểu diễn và suy diễn: Thông tin cần được biểu diễn phù hợp bởi biểu thức logic và có cơ chế biến đổi tự động những biểu thức này (suy diễn tự động) để sinh ra hành động. Kết quả suy diễn phải được tạo ra trong thời gian đủ ngắn để không bị lac hậu so với sự thay đổi của môi trường.

Trên cơ sở sử dụng logic để biểu diễn và suy diễn ta có thể xây dựng tác tử với các kiểu kiến trúc và độ phức tạp khác nhau, cụ thể, có thể xây dựng tác tử phản xạ, tác tử có trạng thái và tác tử hành động có mục đích. Cơ chế suy diễn logic không hỗ trợ trực tiếp cho việc xây dựng tác tử hành động theo hàm tiện ích.

Để minh hoạ cho việc sử dụng logic, ta xét một mô hình tác tử suy diễn logic đơn giản sử dụng logic vị từ bậc nhất (first-order predicate logic). Logic vị từ bậc nhất là một trong những hệ thống logic truyền thống được nghiên cứu và sử dụng nhiều trong triết học, toán học và trí tuệ nhân tạo. Hệ thống logic này cho phép biểu diễn thông tin về môi trường dưới dạng các đối tượng, mỗi đối tượng có thuộc tính của mình. Giữa các đối tượng tồn tại quan hệ được biểu diễn bởi các vị từ và các hàm. Việc kết hợp đối tượng và quan hệ giữa đối tượng tạo nên sự kiện và được biểu diễn bởi câu logic.

Trạng thái bên trong của tác tử được lưu giữ dưới dạng cơ sở dữ liệu bao gồm các biểu thức logic vị từ bậc một và theo truyền thống thường được gọi là cơ sở tri thức của tác tử. Trạng thái tác tử gồm hai phần:

- Phần thứ nhất biểu diễn về những sự kiện mà các tác tử nhận được nhờ quan sát và cảm nhận môi trường.
- Phần thứ hai của trạng thái chứa các quy tắc hay các luật suy diễn logic cho phép tác tử tiến hành suy diễn và đưa ra quyết định.

Một số luật cho phép suy diễn trực tiếp về hành động, một số luật khác cho phép cập nhật thông tin hoặc suy diễn về những sự kiện trong môi trường mà tác tử không cảm nhận trực tiếp được.

Hàm Hành_động được định nghĩa dưới dạng thủ tục suy diễn logic tự động. Ở đây, thủ tục suy diễn có dạng đơn giản của kỹ thuật chứng minh định lý tự động, một kỹ thuật được nghiên cứu nhiều trong trí tuệ nhân tạo.

```
1   function Hành_động(KB:I):A
2   begin
3     for a A do
4       if KB /= Thực_hiện(a) then
5           return a
6       end-if
7     end-for
8     for a A do
9       if (KB/= Thực_hiện(a)) then
10          return a
11      end-if
12    end-for
13    return null
14  end function Hành_động
```

Hình 1.6: Hàm hành động của tác tử suy diễn logic

Hàm *Hành_động* nhận tham số là trạng thái hiện thời *KB* và trả về hành động $a \in A$ hoặc null nếu không tìm được hành động thích hợp. Trước tiên, hàm *Hành_động* tìm hành động a sao cho biểu thức *Thực_hiện*(a) có thể suy ra từ sự kiện và luật suy diễn chứa trong *KB*. Đoạn (3) - (7) lần lượt chọn từng hành động *a* trong tập hành động có thể của mình và cố gắng chứng minh *KB*|=*Thực_hiện*(a). Nếu chứng minh được, *a* sẽ được coi là hành động thích hợp nhất cho tình huống hiện tại, hàm ngừng thực hiện và trả về *a*. Rõ ràng, nếu có nhiều hành động như vậy thì hàm trả về hành động bất kỳ trong số đó.

Trong trường hợp không chứng minh được *Thực_hiện*(a), hàm sẽ tìm một hành động nào đó không bị cấm trong tình huống hiện tại. Đoạn (8) - (12) duyệt tập hành động để tìm hành động a sao cho biểu thức *Thực_hiện*(a) không thể chứng minh được từ *KB* hiện thời. Nếu không tìm được, hàm trả về giá trị null có nghĩa không hành động nào được chọn.

1.2 Hệ đa tác tử (Multi Agent)

1.2.1 Khái niệm hệ đa tác tử

Do ứng dụng ngày càng phức tạp, khả năng giải quyết vấn đề của những tác tử riêng lẻ không đáp ứng được yêu cầu đặt ra hoặc tác tử trở nên quá phức tạp. Trong trường hợp đó, hệ đa tác tử là một giải pháp thích hợp.

Hệ đa tác tử là hệ thống bao gồm nhiều tác tử có khả năng phối hợp với nhau để giải quyết được những vấn đề phức tạp mà đơn tác tử không thể giải quyết được.

Một hệ đa tác tử là một tập hợp các tác tử có mục đích riêng, miền tri thức riêng nhưng có thể tương tác với nhau để hoàn thành mục tiêu chung tổng thể của hệ thống.

1.2.2 Đặc điểm của hệ đa tác tử

Một hệ đa tác tử có các đặc điểm sau:

- Mỗi tác tử có những thông tin không đầy đủ để giải quyết một bài toán đặt ra, do khả năng của mỗi tác tử là có hạn và bài toán cần giải quyết là quá lớn với một tác tử đơn.
- Hệ đa tác tử không có bộ điều khiển toàn cục hệ thống, tức là các tác tử hoạt động một cách độc lập không phụ thuộc vào tác tử khác và cả hệ thống tổng thể.

- Dữ liệu được phân tán cho nhiều tác tử khác nhau trong hệ thống. Mỗi tác tử chỉ có thể quản lý một nguồn tài nguyên hạn chế.
- Sự tính toán trong hệ đa tác tử là không đồng bộ.

Như vậy, trong hệ đa tác tử, các tác tử hoạt động một cách tự chủ và độc lập với nhau và có thể tương tác với nhau dưới nhiều hình thức khác nhau: cộng tác, cạnh tranh, thương lượng,… để chia sẻ tài nguyên và khả năng hợp lý.

Môi trường của hệ đa tác tử có tính chất:

- Môi trường đa tác tử cung cấp một cơ sở hạ tầng để xác định các giao thức truyền thông và tương tác.
- Môi trường mở.
- Môi trường chứa những tác tử tự quản, phân tán. Mỗi tác tử có mục tiêu hoạt động riêng.

1.2.3 Phối hợp trong hệ đa tác tử

Định nghĩa phối hợp: Mục đích chính của việc xây dựng và ứng dụng hệ đa tác tử là kết hợp khả năng của nhiều tác tử tự chủ để giải quyết một số nhiệm vụ. Vì vậy, khi thiết kế và xây dựng hệ đa tác tử cần đặc biệt quan tâm đến vấn đề phối hợp.

Phối hợp là tổ chức và quản lý quan hệ phụ thuộc trong hành động của các tác tử sao cho toàn hệ thống hoạt động một cách thống nhất.

Cụ thể, quá trình phải cho phép đáp ứng được những yêu cầu sau:

- Đảm bảo các phần việc của nhiệm vụ chung được sắp xếp trong kế hoạch của ít nhất trong một tác tử (đảm bảo công việc sẽ được ai đó thực hiện).
- tác tử tương tác với nhau sao cho hoạt động của tác tử được kết hợp với nhau để tạo thành một kết quả chung.
- Các yêu cầu đó phải được thực hiện trong một khoảng thời gian hữu hạn với số lượng tài nguyên tính toán hợp lý.

Sự cần thiết phải phối hợp

Sau đây là một số lý do của sự cần thiết phải phối hợp:

- *Hành động của từng tác tử phụ thuộc vào hành động của tác tử khác*. Hành động của tác tử phụ thuộc vào nhau trong hai trường hợp:
    - Quyết định của tác tử này ảnh hưởng đến tác tử khác, chẳng hạn khi đá bóng việc tiền đạo chạy lên phía trước sẽ ảnh hưởng tới quyết định của tiền vệ chuyền bóng lên thay vì chyền ngang.
    - Hành động của tác tử có thể mâu thuẫn với nhau
- *Phối hợp cho phép tránh được tình trạng hỗn loạn*. Trong hệ thống bao gồm nhiều tác tử, mỗi tác tử chỉ có thể hình dung cục bộ về môi trường và hành động của mình, hành động của nhiều tác tử có thể mâu thuẫn với nhau, tình trạng hỗn loạn là rất tự nhiên và không thể tránh khỏi nếu không có cơ chế phối hợp.
- *Phối hợp cho phép đạt được những ràng buộc tổng thể*. Ràng buộc tổng thể là ràng buộc mà nhóm tác tử cần thoả mãn trong quá trình hoạt động. Nếu mỗi tác tử làm việc riêng rẽ và đều cố gắng tối ưu hàm mục tiêu riêng của mình thì các ràng buộc này sẽ bị phá vỡ.
- *Không cá thể nào có khả năng thực hiện công việc một mình do hạn chế về tài nguyên, khả năng hoặc thông tin.* Nhiều công việc không thể hoàn thành bởi những tác tử hoạt động riêng rẽ do không đủ tài nguyên hoặc thông tin. Việc tạo ra một tác tử vạn năng bao gồm những thành phần có khả năng thực hiện có thể không thực tế hoặc kém hiệu quả, và do vậy, phối hợp nhiều tác tử có khả năng giải quyết vấn đề độc lập là giải pháp duy nhất.

Một số đặc điểm của phối hợp trong hệ tác tử

Phối hợp trong hệ đa tác tử xuất hiện trong thời gian hệ thống hoạt động. Do vậy tác tử phải có khả năng phát hiện yêu cầu phối hợp và thực hiện phối hợp như một phần trong hoạt động của mình. Đặc điểm này khác với các hệ thống phân tán truyền thống trong đó quá trình phối hợp giữa các thành phần được dự đoán trước trong quá trình thiết kế.

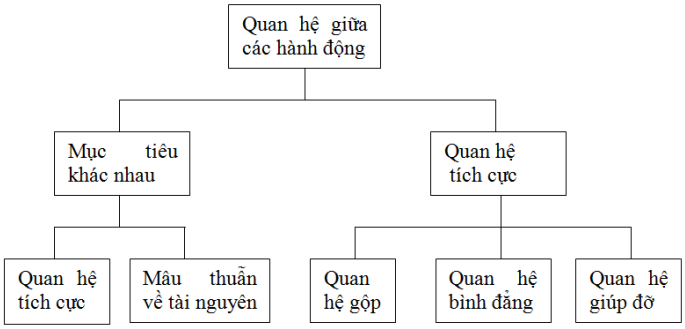
Một điều cần lưu ý là *phối hợp* và *hợp tác* không nhất thiết phải đi cùng với nhau. Nhiều tác tử hợp tác với nhau trong công việc chung không được phối hợp tốt có thể dẫn tới hỗn loạn, thiếu thống nhất. Để có thể hợp tác hiệu quả, tác tử cần lưu trữ mô hình về tác tử khác cũng như hình dung về các tác tử trong tương lai. Trong khi đó, phối hợp có thể thực hiện cho các tác tử không hợp tác với nhau, thậm chí,

điều phối có thể thực hiện đối với hệ thống bao gồm những tác tử cạnh tranh với nhau.

Để thực hiện phối hợp, tác tử có thể *liên lạc* với nhau. Tuy nhiên liên lạc không phải là điều kiện bắt buộc cho phối hợp. Phối hợp không thông qua liên lạc có thể thực hiện nếu mỗi tác tử có được mô hình của tác tử khác.

Quan hệ giữa các hành động

Quan hệ giữa hành động thực hiện bởi các tác tử khác nhau được chia thành hai loại chính: *tiêu cực* và *tích cực*.



Hình 1.7: Các dạng quan hệ giữa các hành động

Hai hoặc nhiều hành động có *quan hệ tiêu cực* hay quan hệ mâu thuẫn với nhau nếu việc thực hiện hành động này gây cản trở cho việc thực hiện đồng thời hành động khác. Nguyên nhân của quan hệ tiêu cực có thể do mâu thuẫn về mục tiêu hành động hay do hạn chế về tài nguyên. Ngược lại, *quan hệ tích cực* là quan hệ cho phép hành động này được lợi từ việc thực hiện hành động khác. Việc kết hợp hành động có quan hệ tích cực cho kết quả tốt hơn so với thực hiện các hành động một cách độc lập. Quan hệ tích cực được phân chia tiếp thành một số dạng sau:

- *Quan hệ bình đẳng* là quan hệ khi một hành động không gắn với một tác tử cụ thể và do đó có thể được thực hiện bởi bất cứ tác tử nào.
- *Quan hệ gộp* là quan hệ khi hành động A của một tác tử X là một phần trong hành động B của tác tử Y, do vậy khi Y thực hiện hành động B của mình, hành động A cũng được thực hiện theo.
- *Quan hệ giúp đỡ* là quan hệ trong đó việc thực hiện hành động này có lợi cho việc thực hiện hành động khác.

1.2.4 Ưu điểm của hệ đa tác tử

Mặc dù bài toán phân tán thì không thể triển khai tập trung được, tuy nhiên ưu điểm của phân tán là dễ hiểu, dễ phát triển, đặc biệt khi bản chất vấn đề cũng phân tán.

Ngày nay thông tin thường phân tán: về địa lý, đa dạng về nội dung, có nhiều thành phần nên thành phần trong hệ thống phân tán có cấu hình động, biến đổi nhanh chóng và ứng dụng riêng lẻ nên rất khó quản lý.

Cách tiếp cận ngày nay là kết hợp phân tán với trí tuệ nhân tạo. Phân tán thường thông qua hệ thống đa tác tử. Mỗi tác tử sẽ hành động như một chương trình thông minh, quản lý các tài nguyên trong phạm vi của mình, kết hợp với các tác tử khác để chia sẻ tài nguyên. Với cách tiếp cận này, hệ thống sẽ dễ dàng phát triển và mở rộng vì tác tử hoạt động và phát triển độc lập.

Trước đây trí tuệ nhân tạo thường tập trung xác định các tác tử có khả năng hoạt động thông minh, với khả năng suy diễn và kiểm soát nội tại theo kiến trúc Von Neumanm. Nhưng hệ thông minh không hoạt động tách biệt – chúng là một phần của môi trường và cũng chứa những hệ thông minh khác.

1.2.5 Các lĩnh vực ứng dụng

Các hệ thống đa tác tử đang ngày càng được sử dụng rộng rãi trong rất nhiều ứng dụng, từ các hệ thống 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 gồm kiểm soát tiến trình (Jenning, 1994), chẩn đoán hệ thống (Albert, 2003), sản xuất (Parunak, 1987), dịch vụ vận tải (Neagu, 2006), và quản lý mạng (Greenwood, 2006).

Một trong những lĩnh vực ứng dụng quan trọng nữa của hệ thống đa tác tử là quản lý thông tin. 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. Trong thực tế, trước sự gia tăng của thương mại điện tử, việc quản lý tiến trình nghiệp vụ đã gần như được điều khiển bởi các tương tác của con người. Giờ đây thương mại điện tử và tiến trình nghiệp vụ được tự động hóa đang ngày càng đóng vai trò then chốt trong nhiều tổ chức bởi vì chúng

cung cấp những cơ hội để cải thiện đáng kể cách thức mà các thực thể khác nhau tham gia vào tương tác của tiến trình nghiệp vụ.

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 (Neagu et al, 2006). Một số lĩnh vực đã được giải quyết như OASIS, OASIS là một hệ thống điều khiển không lưu phức tạp dự trên mô hình tác tử BDI, được triển khai và được sử dụng thành công tại sân bay Sydney ở Australia.

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. 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.

Nhiều hệ thống đa robot cũng sử dụng các kỹ thuật lập kế hoạch phân tán và đa tác tử để phối hợp các robot khác nhau. FIRE phối hợp các hành động của nhiều robot ở nhiều mức trừu tượng hóa. MISUS phối hợp các kỹ thuật từ việc lập kế hoạch và lập lịch với học máy để thực hiện việc thăm dò có tính khoa học một cách tự chủ. Kỹ thuật lập kế hoạch và lập lịch phân tán được sử dụng để tạo ra các kế hoạch cộng tác nhiều lần, việc thực thi kế hoạch giám sát có hiệu quả và thực hiện làm lại kế hoạch khi cần thiết. Hơn nữa, những hệ thống này có thể suy luận về những mục đích phụ thuộc lẫn nhau để thực hiện tối ưu hóa kế hoạch và để tăng giá trị của dữ liệu thu thập được.

Một số ứng dụng đa tác tử đáng quan tâm khác có thể được tìm thấy trong hệ thống chăm sóc sức khỏe. Trong thực tế, các hệ thống đa tác tử đã được đề xuất để giải quyết nhiều loại vấn đề khác nhau trong lĩnh vực chăm sóc sưc khỏe, bao gồm lập kế hoạch và quản lý bệnh nhân, chăm sóc sức khỏe người cao tuổi và cộng đồng, truy cập và quản lý thông tin y tế và hỗ trợ quyết định. Một vài ứng dụng đã cài đặt cho thấy rằng hệ thống đa tác tử có thể là giải pháp đúng đắn cho việc xây dựng các hệ thống hỗ trợ quyết định y học và cải thiện sự phối hợp giữa các chuyên gia khác nhau tham gia vào quá trình chăm sóc sức khỏe.

CHƯƠNG 2: SỰ TƯƠNG TÁC CỦA TÁC TỬ

2.1 Tổng quan về tương tác trong hệ đa tác tử

Hệ đa tác tử bao gồm nhiều tác tử tự chủ có thể hoạt động trên những máy tính khác nhau. Tuy nhiên, các tác tử thường phải trao đổi, tương tác với nhau và chính các tương tác trong hệ đa tác tử quyết định kiến trúc của hệ thống đó. Các dạng tương tác này phức tạp hơn rất nhiều so với các tương tác trong hệ đối tượng. Các tác tử tương tác với nhau bằng cách gửi thông điệp và bản chất của các thông điệp này cũng là những lời gọi hàm như trong hệ các đối tượng nhưng các lời gọi trong tương tác giữa các tác tử có nhiều khác biệt so với tương tác giữa các đối tượng:

Các tham số có thể có kiểu được định nghĩa trong một cấu trúc ngữ nghĩa gọi là ontology.

Các tham số được viết theo một dạng thông điệp truyền thông được định nghĩa bởi một ngôn ngữ truyền thông tác tử (như KQML hoặc FIPA-ACL).

Nội dung của thông điệp trong tương tác đa tác tử có thể rất phức tạp như một chuỗi các hành động hoặc các yêu cầu…

Ngoài sự khác nhau về dạng của các đối số, tương tác trong hệ đa tác tử cũng khác tương tác giữa các đối tượng do bản chất khác nhau giữa đối tượng và tác tử. tác tử là thành phần có tính tự chủ và hành động hướng đích chứ không thụ động như các đối tượng.

Với mỗi hệ tác tử cụ thể được xây dựng thì mục đích chung của hệ thống và mục đích riêng của từng tác tử có thể khác nhau, thậm chí không tương thích nhau. Ví dụ trong hệ thương mại điện tử, nếu tác tử mua có nhiệm vụ mua được hàng với giá càng rẻ càng tốt thì tác tử bán lại có mục đích là bán với giá càng cao càng tốt. Sự thống nhất và mâu thuẫn về mục đích của các tác tử trong hệ thống dẫn đến sự đa dạng của các mô hình tương tác trong hệ đa tác tử. Như vậy, tương tác trong hệ đa tác tử có những đặc trưng riêng khác biệt so với tương tác đa đối tượng. Vai trò của tương tác trong hệ đa tác tử có thể được tổng kết như sau:

Thông qua tương tác, mỗi tác tử sẽ thu thập thông tin và tri thức nhằm đạt được đích riêng của mình và hướng tới đích chung của cả hệ thống.

Tương tác tạo nên tính động cho hệ đa tác tử. Qua tương tác, hệ thống có thể được mở rộng hay thu hẹp một cách dễ dàng, nhất là với các hệ đa tác tử sử dụng tác tử trung gian.

Quá trình tương tác không chỉ diễn ra giữa các tác tử mà còn có thể diễn ra giữa các hệ tác tử khác nhau. Khi đó, khả năng phối hợp giữa các hệ thống để giải quyết các vấn đề phức tạp tăng lên nhiều lần.

Tương tác giữa các tác tử quyết định kiến trúc và hoạt động của hệ đa tác tử đó. Thông qua việc xem xét các tương tác cần có giữa các tác tử, người thiết kế hệ thống có thể xây dựng kiến trúc hệ thống và phân tách nhiệm vụ một cách rõ ràng cho từng tác tử.

Tương tác giữa các tác tử giúp tích hợp các nguồn thông tin trong hệ thống. Trong hệ tích hợp thông tin, mỗi tác tử đại diện cho một nguồn thông tin nhất định. Các nguồn thông tin này thường là không đồng nhất, được biểu diễn theo những cách khác nhau. Thông qua tương tác, thông tin giữa các nguồn đó sẽ được tích hợp để thu được những thông tin cần thiết.

Ba vấn đề sau đây cần quan tâm xem xét khi nghiên cứu về tương tác trong hệ đa tác tử:

Mô hình tương tác: Tuỳ thuộc vào mục đích của hệ thống cụ thể mà người phát triển hệ thống phải lựa chọn một mô hình tương tác phù hợp, mô hình tương tác này sẽ quy định kiến trúc của hệ thống cũng như hành vi của các tác tử trong hệ thống.

Ngôn ngữ truyền thông sử dụng trong các thông điệp: Khi hoạt động trong cùng một hệ thống với nhau các tác tử phải sử dụng chung một ngôn ngữ truyền thông. Ngôn ngữ này không chỉ quy định cấu trúc thông điệp mà còn quy định các dạng thông điệp hỏi và trả lời trong các phiên hội thoại.

Ontology và sử dụng ontology trong tương tác đa tác tử: Mỗi tác tử trong hệ thống là một thành phần phần mềm riêng biệt, do đó, miền tri thức quan tâm của các tác tử trong một hệ thống có thể khác nhau. Để các tác tử có thể hiểu nhau trong quá trình trao đổi thì hệ thống phải sử dụng ontology nhằm biểu diễn các khái niệm mô tả miền và mối quan hệ giữa các khái niệm đó.

2.1.1 Ngôn ngữ truyền thông giữa các tác tử

Các tác tử trao đổi với nhau thông qua các thông điệp. Khác với hệ hướng đối tượng, thông điệp trong hệ đa tác tử không chỉ biểu diễn các lời gọi hàm mà còn phải biểu diễn thông tin và tri thức cần trao đổi giữa các tác tử. Các thông điệp này được biểu diễn theo các ngôn ngữ truyền thông tác tử (ACL: tác tử Comminucation Language) nhằm mục đích:

Định nghĩa khuôn dạng các thông điệp để trao đổi giữa các tác tử trong hệ thống.

Thiết lập một giao thức trao đổi giữa các tác tử, bao gồm: định nghĩa các kiểu thông điệp gửi và nhận, các mô hình trao đổi thông điệp giữa các tác tử.

Các ngôn ngữ truyền thông đều dựa trên lý thuyết hành động - lời nói. Mỗi thông điệp bao giờ cũng phải mô tả đầy đủ người gửi, người nhận, mục đích của lời nói và ngữ nghĩa của lời nói. Một hành động - lời nói đầy đủ không chỉ định nghĩa cấu trúc lời nói mà còn xác định hành động liên quan đến lời nói đó. Có nhiều ngôn ngữ truyền thông đa tác tử đã được đưa ra trong đó hai ngôn ngữ truyền thông được sử dụng rộng rãi nhất nhất là KQML và FIPA-ACL.

KQML (Knowledge Query and Manipulation Language)

Đây là một ngôn ngữ được phát triển theo dự án DARPA trong khoảng thời gian đầu những năm 1990. KQML định nghĩa ngôn ngữ và giao thức cho quá trình chuyển đổi thông tin và tri thức trong hệ đa tác tử.

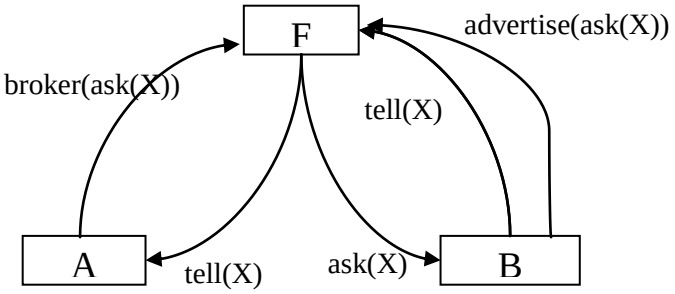
KQML định nghĩa ba mức là mức nội dung, mức thông điệp và mức truyền thông. Mỗi thông điệp KQML định nghĩa một hành động thoại, ngữ nghĩa đi kèm hành động thoại đó, giao thức và một tập các thuộc tính. Cấu trúc chung của một thông điệp KQML như sau:

(performative-name
: sender A
: receiver B
: content X
: language L
: ontology N
: reply-with W
: in-reply-to P)

Mỗi một thông điệp KQML tương ứng với một dạng tương tác trong trường performative-name. Có tới 25 dạng tương tác đã được định nghĩa bao gồm ask-one, advertise, broadcast, insert... Đồng thời, KQML cho phép mở rộng và định nghĩa thêm các dạng tương tác khác khi cần thiết.

Trường content mô tả nội dung của thông điệp. Nội dung này có thể rất phức tạp tuỳ thuộc vào nhu cầu trao đổi thông tin của hai tác tử trong phiên liên lạc đó. tác tử nhận sẽ hiểu được nội dung trong trường content bằng cách tham chiếu vào

trường ontology của thông điệp mà nó nhận được. KQML cũng định nghĩa các giao thức truyền thông bao gồm cả thứ tự các thông điệp, các perormative. Ví dụ một giao thức truyền thông được định nghĩa trong KQML như Hình 2.1:



Hình 2.1: Một giao thức truyền thông trong KQML

FIPA-ACL (Foudation for Intelligent Physical Agent)

FIPA-ACL (Foundation Intelligent Physical Agent) là ngôn ngữ truyền thông tác tử được phát triển năm 1997. FIPA-ACL cũng dựa trên lý thuyết hành động - lời nói và có cấu trúc tương tự như KQML. FIPA-ACL sử dụng XML theo dạng như sau:

```
<fipa-message act = " ">
 <sender>       </sender>
 <receiver>     </receiver>
 <content>      </content>
 <language>     </language>
 <ontology>     </ontology>
 <conversation-id> </conversation-id>
</fipa-message>
```

So với KQML, FIPA-ACL linh động hơn và có thể dễ dàng thêm vào các dạng tương tác mới.

2.1.2 Các mô hình tương tác

Phân loại mô hình tương tác

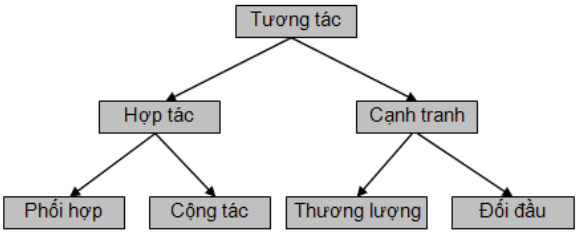
Dựa vào mục đích của các bên tham gia tương tác, có thể chia các hình thức tương tác thành hai loại chính: Hợp tác và cạnh tranh.

Hợp tác: Hai bên cùng thực hiện một công việc chung (cộng tác) hoặc công việc của bên này là bước tiền đề cho bên kia (phối hợp). Hình thức tương tác này

thường xuất hiện khi các tác tử có chung mục đích, nhiệm vụ hoặc cùng thực hiện một tiến trình phức tạp nhất định.

Cạnh tranh: Hai bên cạnh tranh nhau về thông tin, hoặc quyền lợi (thương lượng) hoặc hoàn toàn trái ngược nhau về lợi ích (đối đầu).

Như vậy, có thể có các loại hình tương tác như sau:



Hình 2.2: Các loại hình tương tác

Các giao thức phối hợp

Trong các môi trường phân tán và hạn chế về tài nguyên cho các tác tử thì các tác tử thường phải phối hợp với nhau. Như trình bày trong phần trên, mô hình tương tác được coi là phối hợp khi công việc của tác tử này là tiền đề cho công việc của tác tử kia.

Để các tác tử phối hợp với nhau, các nghiên cứu cho rằng cần xây dựng kỹ thuật phân tán công việc cần thực hiện, bao gồm cả phân tán về điều khiển và phân tán dữ liệu. Phân tán về điều khiển tức là các tác tử có thể tự chủ trong việc sinh ra các hành động mới và quyết định mục đích kế tiếp để hướng tới việc thực hiện công việc chung. Tri thức của hệ thống trong trường hợp này cần được biết bởi tất cả các thành phần trong hệ thống. Dựa trên tri thức này, các tác tử sẽ xác định hành động tiếp theo cần thực hiện trong một chuỗi công việc cần thiết để hoàn thành mục tiêu chung của hệ thống.

Liên quan đến mô hình phối hợp còn nhiều vấn đề khác như sự thoả thuận, các quy ước và việc biểu diễn các thoả thuận hay các quy ước này.

Các giao thức cộng tác

Chiến lược chung của các giao thức cộng tác là phân rã nhiệm vụ cần thực hiện của cả hệ thống và sau đó phân tán các tác vụ cụ thể cho các thành viên. Các tác tử cùng hướng tới đích chung thông qua việc thực hiện các tác vụ mà mình được giao.

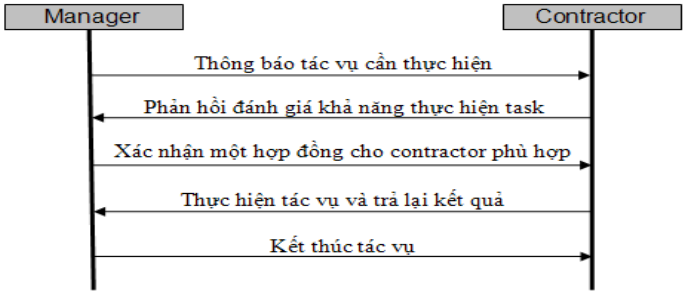
Việc phân rã các tác vụ như thế nào được thực hiện bởi người thiết kế hệ thống và tuân theo các giao thức cụ thể. Các tác vụ được phân rã phải thoả mãn các yêu cầu sau:

- Tránh xung đột tài nguyên.
- Các tác vụ phải phù hợp với khả năng của tác tử.
- Tạo ra một tác tử có nhiệm vụ phân phối tác vụ cho các tác tử khác trong hệ thống.
- Xác định các tác vụ có độ độc lập cao để giảm thiểu việc truyền thông và đồng bộ hoá kết quả.
- Định nghĩa lại các tác vụ nếu cần thiết để hoàn thành một tác vụ "khẩn cấp".

Phần tiếp theo sẽ trình bày hai giao thức cộng tác tiêu biểu là giao thức mạng hợp đồng và giao thức bảng đen.

Giao thức Mạng hợp đồng

Giao thức mạng hợp đồng là một giao thức tương tác kiểu cộng tác. Giao thức này sẽ kết hợp các kết quả từ các tác tử khác nhau thông qua việc mô hình hoá hệ thống theo cơ chế hợp đồng sử dụng trong thương mại để trao đổi sản phẩm và dịch vụ. Mạng hợp đồng sẽ cung cấp giải pháp cho bài toán: tìm một tác tử phù hợp cho một tác vụ cho trước. Giả sử có một tác tử có một tác vụ cần xử lý. tác tử này sẽ được gọi là *manager*, và tác tử có khả năng xử lý tác vụ này gọi là *contractor*. Tương tác giữa *manager* và *contractor* sẽ diễn ra theo biểu đồ tương tác sau:



Hình 2.3: Giao thức mạng hợp đồng

Manager sẽ gửi thông báo về tác vụ cần thực hiện cho tất cả các tác tử khác trong hệ thống. Khi nhận được yêu cầu, các tác tử sẽ gửi trả lại cho *manager* thông báo về khả năng thực hiện tác vụ của mình. *Manager* sẽ đánh giá và chọn ra tác tử phù hợp nhất để thực hiện tác vụ đó và xác nhận tác tử đó thành *contractor*. *Contractor* sẽ có nhiệm vụ thực hiện tác vụ và trả lại kết quả cho *manager*.

Trên lý thuyết, *manager* có thể là bất kỳ tác tử nào trong hệ thống khi có task cần thực hiện. Vì vậy mỗi tác tử có thể nhận được nhiều task, nếu tác tử đó là tác tử có khả năng xử lý cao thì rất nhiều *manager* sẽ chọn tác tử đó làm *contractor*. Khi đó, *contractor* sẽ lựa chọn task "hấp dẫn" nhất và mô hình mạng hợp đồng sẽ trở nên phức tạp hơn nhiều.

Giao thức Bảng đen

Phương pháp giải quyết bài toán dựa trên giao thức bảng đen được mô tả như sau:

Giả sử có một nhóm chuyên gia hoặc tác tử cùng ngồi cạnh một bảng đen lớn. Các chuyên gia sẽ cộng tác với nhau để giải quyết bài toán thông qua việc sử dụng bảng đen để phát triển lời giải. Quá trình giải bài toán bắt đầu khi bài toán và dữ liệu đầu vào được viết lên bảng đen. Các chuyên gia sẽ quan sát bảng đen và cố gắng đưa ra ý kiến để phát triển lời giải của bài toán. Khi tìm ra được một thông tin phù hợp, chuyên gia này sẽ viết thông tin đó lên bảng đen. Các chuyên gia khác sẽ sử dụng thông tin này để tiếp tục tìm ra lời giải. Quá trình cứ tiếp tục như vậy cho đến khi bài toán được giải quyết hoàn toàn.

Áp dụng giao thức bảng đen cho hệ đa tác tử ta sẽ có mô hình tương tác kiểu bảng đen. Khi đó, hệ thống này có các đặc điểm sau:

- Tính độc lập về giải pháp: Các chuyên gia đưa ra các ý kiến độc lập với nhau.
- Tính đa dạng trong kỹ thuật giải bài toán: Thông qua bảng đen và các phương pháp biểu diễn tri thức thì một bài toán có thể có rất nhiều hướng giải quyết khác nhau.
- Cho phép biểu diễn thông tin một cách linh hoạt trên bảng đen.
- Sử dụng ngôn ngữ tương tác chung.

2.1.3 Tương tác với tác tử trung gian

2.1.3.1 Vai trò của tác tử trung gian

Trên quan điểm chú trọng đến các mô hình có sử dụng tác tử trung gian, ta có thể chia các mô hình tương tác trong hệ đa tác tử thành: (i) tương tác với tác tử trung gian và (ii) tương tác không sử dụng tác tử trung gian.

Các mô hình tương tác không sử dụng tác tử trung gian như mô hình bảng đen, mạng hợp đồng… có ưu điểm là đơn giản, dễ xây dựng và phù hợp với những

hệ thống đa tác tử đóng. Các mô hình này yêu cầu các tác tử phải biết được khả năng của các tác tử khác trong hệ thống mà nó muốn tương tác. Do đó, hệ thống với các mô hình này khó mở rộng cho tác tử khác tham gia như trong môi trường Internet.

Khác với các mô hình bảng đen hay hợp đồng, mô hình tương tác với tác tử trung gian sử dụng một tác tử trung gian nhằm quản lý khả năng của các tác tử khác. Trong mô hình này, tác tử *Yêu cầu* sẽ tương tác với tác tử trung gian để biết được khả năng của các tác tử trong hệ thống có thể giải quyết được yêu cầu của mình . Vai trò của tác tử trung gian trong những mô hình cụ thể có thể khác nhau nhưng lớp tác tử này đều có chung những đặc trưng sau:

- Cung cấp các phương tiện dịch vụ cơ bản để quản lý xã hội các tác tử.
- Phối hợp các dịch vụ được cung cấp theo một giao thức xác định nào đó.
- Đảm bảo quản lý các tác tử bên trong xã hội tác tử và quản lý việc thêm hay bớt các tác tử tham gia vào hệ thống.

Lớp mô hình tương tác sử dụng tác tử trung gian được chia ra thành 3 mô hình nhỏ gồm mô hình tương tác kiểu tác tử trung tâm, mô hình tương tác kiểu môi giới và mô hình tương tác kiểu điều phối.

2.1.3.2 Các mô hình tương tác với tác tử trung gian

Mô hình tương tác với tác tử trung tâm (Mediator Agent)

Trong mô hình này, nhiệm vụ của tác tử trung tâm là chủ động liên lạc với các tác tử khác có dữ liệu hay tri thức cần thiết trong hệ thống. Các dịch vụ mà tác tử Trung tâm có thể cung cấp là:

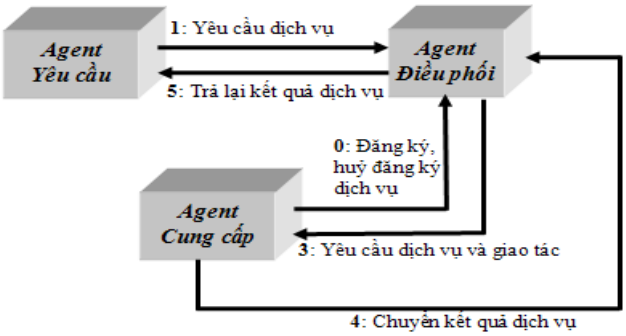
- Tự động xác định các dịch vụ thông tin;
- Xác định vai trò của các tác tử trong hệ thống;
- Tự thu thập và tạo ra thông tin từ các tác tử cung cấp (Provider Agent) sau đó gửi trả lại cho các tác tử yêu cầu.

Để thực hiện nhiệm vụ trên, tác tử trung tâm sử dụng mô hình thông tin toàn cục bằng cách thu thập và tích hợp các thông tin cần thiết để giải quyết các yêu cầu hoặc có thể chuyển yêu cầu cho các tác tử phù hợp trong hệ thống để giải quyết. Như vậy, tác tử trung tâm đóng vai trò vừa là tác tử trực tiếp quản lý các tác tử khác lại vừa tự tìm ra thông tin cần thiết để giải quyết và gửi trả lại kết quả cho các tác tử

yêu cầu. Công việc của tác tử trung tâm là rất nhiều và hiệu quả hoạt động của hệ thống phụ thuộc hoàn toàn vào khả năng của tác tử này.

Mô hình tương tác với tác tử điều phối (Broker Agent)

Trong mô hình này, tác tử trung gian đóng vai trò là tác tử điều phối. Công việc mà tác tử điều phối cần thực hiện là một phần công việc của tác tử trung tâm và được biểu diễn như trong Hình 2.4. Khi có một tác tử mới tham gia vào hệ thống, thì nó phải đăng ký khả năng cung cấp dịch vụ của mình cho tác tử điều phối. Dịch vụ, tên và địa chỉ của tác tử cung cấp này sẽ được cập nhật vào cơ sở tri thức của tác tử điều phối.



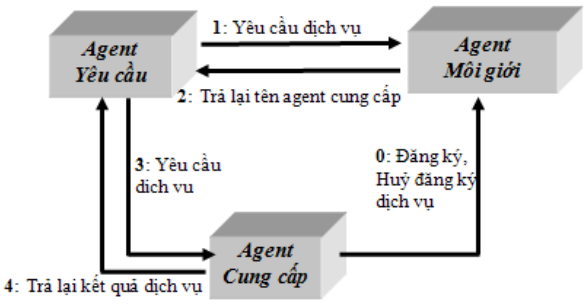
Hình 2.4: Mô hình tương tác với tác tử điều phối

Khi có một tác tử tác tử yêu cầu gửi cho tác tử điều phối một yêu cầu dịch vụ nào đó, tác tử điều phối sẽ tìm kiếm trong cơ sở tri thức của mình xem có tác tử cung cấp nào có thể giải quyết được yêu cầu của tác tử yêu cầu không và sau đó liên lạc trực tiếp với tác tử đó để giải quyết yêu cầu. Sau cùng, tác tử điều phối sẽ gửi lại kết quả cho tác tử yêu cầu và kết thúc quá trình tương tác.

Như vậy, trong mô hình tương tác này, bất cứ một liên lạc nào giữa tác tử yêu cầu và tác tử cung cấp đều phải thông qua tác tử điều phối. Trong một số tài liệu, tác tử điều phối còn được gọi là Facilitator. Ưu điểm của mô hình này là khả năng mở rộng hệ thống. Một tác tử mới muốn tham gia vào hệ thống thì tác tử đó chỉ cần đăng ký dịch vụ với tác tử điều phối. Tuy nhiên, nhược điểm của mô hình này là tác tử yêu cầu phải gửi đi toàn bộ yêu cầu của mình cho tác tử điều phối mà điều này thường không thực tế đặc biệt trong thương mại điện tử.

Mô hình tương tác với tác tử môi giới (MatchMaker Agent)

Trong mô hình này, tác tử trung gian đóng vai trò như một tác tử môi giới mà nhiệm vụ chính của nó là tạo ra cơ chế liên lạc trực tiếp giữa tác tử yêu cầu và tác tử cung cấp như minh hoạ trong Hình 2.5.



Hình 2.5: Mô hình tương tác với tác tử môi giới

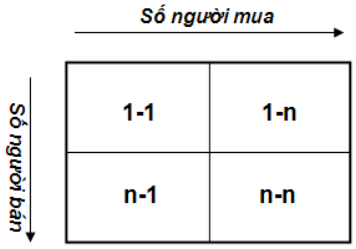
Tương tự như mô hình tác tử điều phối, khi muốn tham gia vào hệ thống, mỗi tác tử phải đăng ký dịch vụ với tác tử môi giới và trở thành nhà cung cấp dịch vụ (tác tử cung cấp). tác tử môi giới sẽ cập nhật vào cơ sở tri thức của nó tên và khả năng dịch vụ của tác tử cung cấp. Trong một tương tác cụ thể, khi có một tác tử yêu cầu yêu cầu một dịch vụ, nó sẽ gửi yêu cầu đó đến tác tử môi giới. Tác tử môi giới sẽ xem xét trong cơ sở tri thức của nó để tìm ra tác tử cung cấp có thể thực hiện yêu cầu và sẽ gửi cho tác tử yêu cầu tên, địa chỉ của tác tử cung cấp đó. Quá trình tương tác sau đó sẽ diễn ra trực tiếp giữa tác tử yêu cầu và tác tử cung cấp.

Như vậy, công việc mà tác tử môi giới phải thực hiện là một phần công việc của tác tử điều phối. Trong mô hình này, tác tử yêu cầu chỉ cần gửi đi yêu cầu nào liên quan đến việc tìm ra tác tử cung cấp phù hợp.

2.2 Thương lượng trong hệ đa tác tử

Khác với các mô hình tương tác kiểu cộng tác như mạng hợp đồng hay bảng đen, thương lượng là một tiến trình tương tác vừa cộng tác vừa cạnh tranh được diễn ra giữa hai hay nhiều bên tham gia, bắt đầu bằng những mục tiêu khác nhau, dần dần đi đến một thoả thuận chung có lợi cho tất cả các bên.

Trong bài toán thương lượng, tuỳ vào số bên tham gia, người ta chia làm bốn nhóm là: thương lượng 1-1, thương lượng 1- n, thương lượng n-1 và thương lượng n-n. Sự phân chia này được minh hoạ như Hình 2.6.



Hình 2.6: Các dạng thương lượng

Thương lượng 1-1: còn gọi là thương lượng song phương, chỉ có một người bán thương lượng với một người mua.

Thương lượng n-1: nhiều người mua một người bán, đây chính là hình thức đấu giá. Đấu giá là một trong những hình thức mua bán phổ biến trong thương mại. Trong mô hình đấu giá (nhiều người mua, một người bán), người mua sẽ trả giá theo một cách thức nào đó, ví dụ như ai trả giá cao nhất sẽ thắng.

Thương lượng 1-n: Một người mua có thể thương lượng đồng thời với nhiều người bán. Hình thức này còn gọi là đấu giá ngược.

Thương lượng n-n: còn gọi là thương lượng đa phương hay chợ. Mỗi người mua có thể thương lượng đồng thời với nhiều người bán và mỗi người bán cũng có thể thương lượng đồng thời với nhiều người mua.

Khi hệ thống có các tác tử đại diện cho người mua và người bán thì các tác tử này sẽ tự động thương lượng với nhau. Trong thương lương song phương người ta chia ra ba cách tiếp cận:

Cách tiếp cận dựa trên lý thuyết trò chơi

Đây là cách tiếp cận sử dụng chiến lược tương tác giữa các tác tử ích kỷ (self-interested agent) theo các luật trò chơi. Trong cách tiếp cận này, các nhà nghiên cứu cố gắng xác định một chiến lược tối ưu bằng cách phân tích mối tương tác giữa các tác tử giống như trong một trò chơi và tìm ra điểm cân bằng của quá trình tương tác giữa các tác tử đó.

Cách tiếp cận dựa trên Heuristic

Trong mô hình này, các hàm quyết định dựa trên heuristic được sử dụng để ước lượng và sinh ra các yêu cầu hay đề nghị mới trong quá trình thương lượng.

Cách tiếp cận dựa trên lập luận (Argumentation-based)

Cách tiếp cận này cho phép các tác tử có thể chuyển các thông tin thêm hoặc lập luận cho các giá trị tinh thần như niềm tin hay ý định trong quá trình thương lượng.

CHƯƠNG 3: MÔI TRƯỜNG, NỀN TẢNG PHÁT TRIỂN TÁC TỬ

Hiện nay có rất nhiều môi trường, hệ thống, nền tảng để phát triển tác tử di động và các ứng dụng về tác tử di động. Chương này giới thiệu không giới thiệu đầy đủ tất cả các môi trường phát triển tác tử di động mà chỉ giới thiệu một cách tổng quan một số môi trưởng và nền tảng để phát triển tác tử di động được sử dụng nhiều hiện nay.

3.1 Aglets

Aglets được xây dựng và phát triển bởi D. B. Lange và IBM Tokyo Research Laboratory, là một hệ thống Java tác tử di động hỗ trợ các khái niệm thi hành tự trị và định tuyến động trên lộ trình của nó. Aglet server là chương trình cung cấp một môi trường thi hành và một máy ảo Java cho aglet hoạt động. Ngoài ra, Aglet server cũng sử dụng một trình quản lý để tiếp nhận và kiểm soát aglet một cách an toàn.

Aglet API là bộ thư viện bao gồm các hàm chuyên biệt dành cho việc phát triển tác tử. Nhờ vào Aglet API, khả năng nổi tiếng của Java là "viết một lần, thi hành bất cứ đâu" được viết lại là "viết một lần, lưu hành bất cứ đâu". Trong mô hình đối tượng aglets, một mobile tác tử là một đối tượng di động có luồng kiểm soát riêng của nó, làm việc theo sự kiện và liên lạc với các tác tử khác bằng cách truyền thông điệp. Aglets có một cơ chế định danh duy nhất và toàn cục dựa trên URL. Aglets hỗ trợ cơ chế di động yếu (weak-mobility). Các aglets giao tiếp với nhau một cách đồng nhất, và độc lập với vị trí lưu trú thông qua đối tượng proxy.

Aglets là một trong những nền tảng được sử dụng nhiều nhất để phát triển các hệ thống mobile tác tử. Một số đề án thực hiện với Aglet có thể kể đến là TabiCan (http://www.tabican.ne.jp) - chợ điện tử chuyên bán vé máy bay và tour du lịch trọn gói -, Cps720 (Artificial Intelligence Topics with Agent) tại đại học Ryerson University, Mỹ , Acme – Hệ thống hỗ trợ Sales Order Processing trong việc mua bán chứng khoán, của Đại học Loughborough, Anh.

3.2 Voyager

Voyager là một môi trường thương mại hỗ trợ phát triển các ứng dụng tác tử được hãng Object Space phát triển từ giữa năm 1996.

Voyager sử dụng ngôn ngữ lập trình Java với cú pháp chuẩn để tạo dựng các đối tượng ở xa một cách rất dễ dàng, cho phép các đối tượng này trao đổi thông điệp với nhau, và di chuyển các đối tượng giữa các máy tính có hỗ trợ môi trường Voyager. Voyager hỗ trợ mạnh về tính di động với khả năng mang toàn bộ mã

chương trình và dữ liệu di chuyển từ máy ảo Java này sang máy ảo Java khác nếu các máy ảo có hỗ trợ Voyager. Trạng thái hoạt động của tác tử cũng sẽ được bảo toàn và tiếp tục thực thi tại nơi tác tử đến.

Một trong những đặc điểm nổi trội khác của Voyager là tính phổ quát. Các chương trình viết trong Voyager có thể trao đổi thông tin hai chiều với các chương trình viết bằng SOAP, CORBA, RMI và DCOM. Các dạng thông tin được trao đổi có thể là các lời gọi hàm từ xa, các dịch vụ đặt tên, dịch vụ thư mục. Voyager có thể được xem là một cửa ngõ, một cầu nối làm cho các chương trình theo chuẩn khác trở nên liên thông với nhau. Hơn nữa, tất cả các chương trình và đối tượng có thể được tổ chức thành một không gian chung, nhờ vậy việc liên lạc sẽ trở thành một–nhiều một cách tự động.

Thế mạnh thật sự của Voyager nằm ở sự đơn giản và dễ dùng. Sự "trong suốt" hay cách mà Voyager che dấu các kỹ thuật lập trình phân tán phức tạp đã làm cho việc xây dựng các ứng dụng tác tử di động trở nên dễ dàng hơn rất nhiều. Việc tích hợp các công nghệ mới và các chuẩn mới vào cùng một sản phẩm tạo cho Voyager sự hấp dẫn rất riêng biệt.

3.3 Mole

Mole là hệ thống tác tử di động được xây dựng với ngôn ngữ Java tại đại học Stuttgart (CHLB Đức

Được xây dựng trên Java, Mole có khả năng thực thi trên tất cả các môi trường có hổ trợ JDK1.1.x (Jdk1.1.7 và Jdk1.1.8), sử dụng giao thức TCP/IP trong quá trình giao tiếp. Mole hổ trợ di chuyển yếu.

Để thực hiện giao tiếp giữa các tác tử Mole sử dụng các cơ chế truyền thông điệp, gọi hàm từ xa RPCs, và cơ chế đặc trưng của Mole là session, badge. Ngôn ngữ giao tiếp giữa các tác tử được Mole hỗ trợ là KQML. Việc trao đổi dữ liệu giữa các tác tử được thực hiện theo nghi thức TCP/IP. Mole cho phép đa tiểu trình/tác tử và quản lí tài nguyên và lập lịch các tiểu trình trong hệ thống thông qua bộ lập lịch trung tâm MCP. Khả năng bảo mật của Mole được đánh giá khá tốt trong các hệ thống tác tử. Tác tử trong hệ thống Mole được chia làm hai loại: tác tử người dùng và tác tử hệ thống. tác tử người dùng là những tác tử di động được kích hoạt bởi người dùng và không thể truy cập trực tiếp tài nguyên hệ thống. Ngược lại, tác tử hệ thống - được khởi động bởi người quản trị - không có tính di động và được phép truy cập tài nguyên hệ thống.

Môi trường Mole phù hợp cho phát triển những ứng dụng trong các lĩnh vực: Truyền thông, ứng dụng thuộc lĩnh vực hệ thống thông tin điện tử. Một số ứng dụng dược phát triển trên môi trường Mole: AIDA - Infrastructure for Mobile Agents, ASAP, ATOMAS, FESTIVAL (Mole office, Mole shopping), HAWK.

3.4 Zeus

Zeus là môi trường do British TeleCommunication phát triển để hỗ trợ xây dựng các hệ thống đa tác tử. Ngoài các tính năng thông thường trong việc tạo lập và quản lý các tác tử, Zeus đặc biệt chú trọng việc hỗ trợ một phương pháp luận và một bộ công cụ mạnh để phát triển ứng dụng đa tác tử trên môi trường phân tán.

Zeus định nghĩa một phương pháp luận để phân tích, thiết kế, triển khai hệ thống và còn kèm theo các công cụ cho phép người phát triển có thể bắt lỗi hệ thống cũng như phân tích sự thực hiện của mình. Hai đặc tính quan trọng của các tác tử Zeus là tính tự trị và cộng tác. Khả năng thương lượng và cộng tác giữa các tác tử cũng được Zeus tích hợp vào trong toolkit thông qua một thư viện các giao thức, cùng các chiến lược thương lượng và cộng tác. Do có mã nguồn mở, người dùng có thể thêm vào thư viện này các chiến lược riêng phù hợp với ứng dụng của mình.

Các tác tử Zeus truyền thông theo point-to-point socket TCP/IP với mỗi message là một chuỗi các kí tự mã ASCII. Ngôn ngữ truyền thông Zeus sử dụng là FIPA ACL (http://www.fipa.org). Nhằm cung cấp khả năng "hiểu" lẫn nhau cho các tác tử, Zeus cung cấp các công cụ cho việc định nghĩa các ontology - cơ sở khái niệm chung cho cộng đồng tác tử. Visualiser của Zeus cung cấp các công cụ để kiểm tra các quan hệ giao tiếp giữa các tác tử, trạng thái tác vụ những tác tử đang thực hiện và trạng thái bên trong của tác tử. Đồng thời, Zeus Statistic Tool cho phép người dùng so sánh các thống kê khác nhau về cộng đồng tác tử. Cũng nhằm quản lí tác tử, Zeus cung cấp những tác tử tiện ích như tác tử Name Server hoạt động như một Yellow Page, Facilitator như một White Page, Visualiser và Database Proxy.

Một hạn chế của Zeus là tuy được liệt kê vào một trong những môi trường tác tử di động nhưng hiện hướng nghiên cứu về tính di động của Zeus chỉ mới ở bước đầu, chưa được cài đặt. Do đó mà tính bảo mật của Zeus cho các tác tử hầu như không có.

Zeus đã và đang được triển khai trong một số ứng dụng như tác tử Based Work-flow Management, PTA: Personal Travel Assistance, Personal Computer Manufacture, tác tử-based Electronic Commerce, Network Management (VPNP), Home Shopping.

3.5 JADE (Java Agent DEvelopment Framework)

JADE là một phần mềm framework được xây dựng hoàn toàn bằng ngôn ngữ Java. Nó đơn giản hóa việc triển khai các hệ thống đa tác tử (multi - Agent systems) thông qua một middleware mà nó đòi hỏi phải tuân theo các chi tiết kỹ thuật FIPA và thông qua một bộ công cụ hỗ trợ trong gỡ lỗi và trong triển khai các giai đoạn. Nền tảng tác tử (Agent nền tảng) có thể được phân phối trên nhiều máy (không nhất thiết cần phải cùng một hệ điều hành) và cấu hình có thể được điều khiển thông qua một giao diện từ xa (a remote GUI). Cấu hình có thể được thay đổi ngay cả ở thời gian chạy (run-time) bằng cách tạo ra các tác tử mới và di chuyển các tác tử từ một máy đến một máy khác khi cần thiết. Các kiến trúc truyền thông cung cấp truyền tin linh hoạt và hiệu quả, JADE tạo ra và quản lý một hằng đợi các tin nhắn ACL đến từng tác tử. Các tác tử có thể truy cập vào hàng đợi thông qua sự kết hợp của nhiều phương thức: blocking, polling, time-out và mô hình kết hợp cơ bản.

Mục tiêu của JADE là để đơn giản hóa sự phát triển trong khi vẫn đảm bảo tuân thủ tiêu chuẩn thông qua một hệ thống dịch vụ và tác tử. Để đạt được mục tiêu trên, JADE cung cấp danh sách các tính năng cho việc lập trình tác tử sau đây:

FIPA-compliant Agent Nền tảng, trong đó bao gồm các AMS (Hệ thống quản lý tác tử - Agent Managentment System), các DF (Directory Facilitator), và ACC (kênh truyền thông tác tử - Agent communication Channel). Tất cả ba tác tử trên sẽ được tự động kích hoạt khi tác tử nền tảng được kích hoạt.

Phương thức trao đổi đơn giản của ACL message bên trong các tác tử Nền tảng giống nhau như thông điệp được chuyển mã hóa là đối tượng Java hơn là chuỗi, để tránh thủ tục marshalling và unmarshaling. Khi người gửi hoặc nhận không thuộc cùng một nền tảng, thông điệp sẽ được tự động chuyển sang các định dạng chuỗi FIPA.

Thư viện các giao thức tương tác FIPA sẵn sàng để được sủ dụng.

Đăng ký tự động của tác tử với AMS.

Tên dịch vụ tương thích FIPA: lúc khởi động các tác tử tồn tại luôn GUID Globally Unique Identifier) của chúng từ nền tảng.

Đồ họa giao diện người dùng để quản lý một số tác tử và tác tử nền tảng từ cùng một tác tử. Từng hoạt động của nền tảng có thể được theo dõi và khóa.

Giao diện đồ họa cho người dùng để quản lý nhiều tác tử và các nền tảng tác tử cùng một tác tử.

3.6 Các tính năng hỗ trợ của các hệ thống tác tử di động

Để đánh giá các tính chất của một môi trường phát triển ứng dụng tác tử cần chú ý nhiều đến các đặc tính của tác tử: tính di động, tính tự trị và khả năng bảo mật mà môi trường hỗ trợ.

Tính di động (mobility): xét khả năng môi trường hỗ trợ di động mạnh hay yếu cho tác tử.

Tính tự trị: xét vai trò của môi trường về việc cung cấp các cơ chế hỗ trợ cho tác tử linh động thực thi, qua đó thể hiện được tính tự trị.

Tính an toàn: cần chú ý tới 2 vấn đề: bảo vệ các host tránh các cuộc tấn công từ các tác tử nguy hiểm và bảo vệ các tác tử chống lại các môi trường nguy hiểm khi tác tử di trú

Tính thích ứng: là khả năng hỗ trợ tính thích ứng từ phía môi trường khi tiếp nhận một tác tử du nhập.

Khả năng cộng tác: để xem xét khả năng phối hợp hoạt động của các tác tử, cần lưu ý về ngôn ngữ liên lạc, giao thức liên lạc, mô hình cộng tác mà môi trường hỗ trợ.

Phương pháp luận và công cụ phát triển tác tử: mô hình tác tử tương đối mới mẻ, nên việc hỗ trợ các phương pháp luận giúp cho việc phân tích, thiết kế và triển khai ứng dụng cũng là một yêu cầu cần thiết

Khả năng mở rộng: các môi trường có mã nguồn mở sẽ rất có triển vọng trong việc phát triển. Việc đầu tư nghiên cứu sẽ được tiến hành song song tại nhiều điểm và do đó môi trường sẽ có được những tính năng rất đa dạng và phù hợp với nhiều đối tượng sử dụng.

CHƯƠNG 4: NỀN TẢNG JADE

Chương này cung cấp một cái nhìn tổng quan về nền tảng JADE và các thành phần chính tạo thành kiến trúc của nó.

4.1 Tóm tắt lịch sử của JADE

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. 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.

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 phẩm 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.

4.2 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 phần mềm. Một đặc điểm đầy ý nghĩa của JADE là nó thực thi sự trừu tượng này trên ngôn ngữ hướng đối tượng, Java, cung cấp một API đơn giản và thân thiện.

Một tác tử có tính tự chủ và hướng đích: một tác tử không thể cung cấp các call - back 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). Việc này không phụ thuộc vào thời gian giữa người gửi và người nhận: một người nhận có thể không có mặt khi người gửi gửi thông điệp đến. Cũng không cần phải lấy tham chiếu đối tượng của tác tử nhận mà cần có các định danh tên để hệ vận chuyển thông điệp có thể dựa vào đó để chuyển thông điệp đến đúng địa chỉ. Thậm chí bên gửi có thể không cần biết về định danh của bên gửi, nó có thể định nghĩa một danh sách bên nhận sử dụng intentional grouping (nhóm người nhận dự kiến) hoặc sử dụng một proxy tác tử trung gian.

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. Cách truyền thông này còn cho phép bên gửi có thể điều khiển luồng thực thi của nó và như vậy không bị khóa cho đến khi bên nhận xử lý thông điệp.

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 sau đây:

- Một hệ thống hoàn toàn phân tán mà các tác tử cư trú trên đó, mỗi tác tử hoạt động như là một luồng riêng biệt, và có khả năng giao tiếp một cách trong suốt với tác tử khác.
- Tuân thủ đầy đủ các đặc tả của FIPA. Nền tảng tham gia thành công vào tất cả các sự kiện phối hợp hoạt động của FIPA và được sử dụng như là tầng giữa của nhiều nền tảng trong mạng lưới tác tửcities. Điều này đã tạo nên sự đóng góp lớn lao của đội JADE vào quá trình chuẩn hóa của FIPA.
- Phương tiện vận chuyển hiệu quả của các thông điệp không đồng bộ thông qua một API trong suốt về vị trí. Nền tảng lựa chọn các phương tiện sẵn có tốt nhất của truyền thông và khi có thể, tránh sự sắp xếp theo thứ tự hoặc không theo thứ tự các đối tượng Java. Khi đi qua ranh giới nền tảng, các thông điệp tự động được biến đổi từ cách biểu diễn bằng Java bên trong của JADE sang các cú pháp tuân theo FIPA, cách giải mã và các giao thức vận chuyển.
- Thực thi cả 2 dịch vụ white-page và yellow-page. Hệ thống có thể được cài đặt để biểu diễn các miền và các miền con như một đồ thị các thư mục.
- Quản lý vòng đời tác tử đơn giản nhưng hiệu quả. Khi các tác tử đã được tự động gán một định danh toàn cục duy nhất và một địa chỉ vận chuyển được sử dụng để đăng ký với dịch vụ white-page của nền tảng. Các API đơn giản và các công cụ đồ họa cũng được cung cấp để quản lý vòng đời tác tử vừa từ xa và vừa cục bộ, như tạo, đình chỉ, phục hồi, đóng băng, tan băng, di chuyển, lặp lại và xóa.
- Cung cấp tính di động của tác tử. Cả mã và trạng thái của tác tử đều có thể di chuyển giữa các tiến trình và các máy. Sự di chuyển của được tạo ra để các tác tử giao tiếp một cách trong suốt mà có thể tiếp tục tương tác thậm chí là trong suốt quá trình di chuyển.
- Một cơ chế đặt trước cho mỗi tác tử và thậm chí là cả các ứng dụng bên ngoài muốn đăng ký với một nền tảng để được thông báo về tất cả các sự kiện của nền tảng
- Một tập các công cụ đồ họa để hỗ trợ người lập trình khi *debug* và *monitor*. Chúng đặc biệt quan trọng và phức tạp trong các hệ thống đa luồng, nhiều tiến trình, nhiều máy ví dụ như một ứng dụng JADE điển hình.

- Hỗ trợ các Ontology và các ngôn ngữ nội dung. Việc kiểm tra ontology và việc mã hóa nội dung được thực hiện tự động bởi nền tảng, các nhà lập trình có thể lựa chọn các ngôn ngữ nội dung và ontologies yêu thích.
- Một thư viện của các giao thức tương tác: mô hình các mẫu đặc trưng của truyền thông nhằm đạt được một hoặc nhiều mục đích. Các skeleton độc lập với ứng dụng là một tập các lớp Java có sẵn và có thể tùy chọn.
- Sự tích hợp với các công nghệ khác nhau dựa trên Web bao gồm các công nghệ JSP, Servlet, applet và Web Service. Nền tảng cũng có thể được cấu hình một cách dễ dàng để xuyên qua tường lửa.
- Hỗ trợ nền tảng J2ME và môi trường không dây. JADE run-time có thể dùng cho các nền tảng J2ME – CDC và J2ME-LCDC thông qua một tập không đổi của các API che phủ cả 2 môi trường J2ME và J2SE.
- Một giao diện tiến trình bên trong cho việc khởi chạy và việc điều khiển một nền tảng và các thành phần phân tán của nó từ một ứng dụng bên ngoài.
- Một nhân có thể mở rộng được thiết kế để cho phép những người lập trình mở rộng các chức năng của nền tảng thông qua việc bổ sung các dịch vụ phân tán mức nhân.

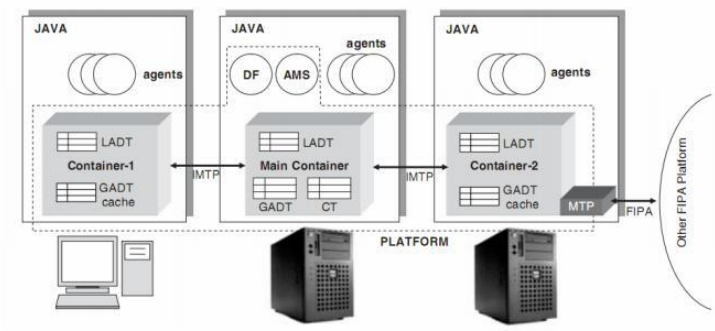
4.3 Kiến trúc JADE

Hình 4.1 chỉ ra các thành phần kiến trúc chính của nền tảng JADE. Một nền tảng JADE bao gồm các khung chứa tác tử, 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 nền tảng: 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 4.2 miêu tả quan hệ giữa các thành phần kiến trúc chính trong JADE.

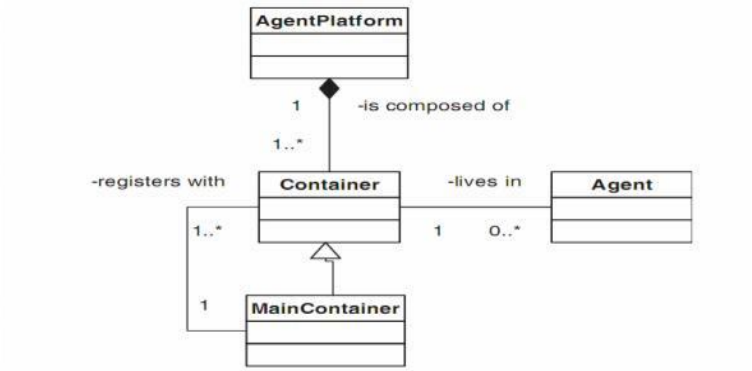
Người lập trình sử dụng tên logic để xác định các khung chứa; mặc định, khung chứa chính được gọi là “Main Container” trong khi các khung chứa khác có tên lần lượt là “Container-1”, “Container-2”,…Các dòng lệnh khác nhau đã sẵn có để thay đổi các tên mặc định đó. Main container có những nhiệm vụ đặc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

- Quản lý bảng khung chứa (container table), nơi đăng kí các tham chiếu của đối tượng và các địa chỉ giao dịch của tất các khung chứa có trong nền tảng.

- 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 nền tảng, 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 (tác tử management) và dịch vụ trang trắng (white page service), và dịch vụ trang vàng măc định của nền tảng (default yellow page service).



Hình 4.1: Các thành phần kiến trúc chính của Nền tảng JADE



Hình 4.2: Quan hệ giữa các thành phần kiến trúc chính trong JADE

Một câu hỏi thường gặp là liệu khung chứa chính có phải là nút thắt cổ chai của hệ thống không. Câu trả lời là không, vì JADE cung cấp một bộ nhớ cho GADT để mỗi container quản lý cục bộ. Thông thường, các hoạt động của nền tảng không liên quan tới khung chứa chính mà chỉ liên quan tới bộ nhớ cục bộ và hai khung chứa lưu trữ các tác tử - là chủ thể và đối tượng của hoạt động (ví dụ, người gửi và người nhận thông điệp). Khi một khung chứa muốn tìm ra nơi mà tác tử nhận thông điệp cư ngụ, đầu tiên nó tìm kiếm trong LADT của nó (Local Agent descriptor table), nếu việc tìm kiếm không đưa lại kết quả, thì khung chứa chính được liên hệ để lấy tham chiếu từ xa phù hợp, sau đó tham chiếu này được lưu vào LADT để sử

dụng sau này. Vì hệ thống là động (các tác tử có thể cư ngụ, chấm dứt, hay tác tử mới có thể xuất hiện), nên đôi khi chúng có thể sử dụng một giá trị ánh xạ có được từ một địa chỉ vô giá trị. Trong trường hợp này khung chứa nhận một ngoại lệ và buộc phải làm mới lại bộ nhớ để chống lại khung chứa chính. Chính sách thay thế của bộ nhớ là LRU (least recently use), được thiết kế để tối ưu các cuộc đàm thoại dài, khách quan hơn là cho các cuộc đàm thoại trao đổi thông điệp đơn, rời rạc trong các ứng dụng đa tác tử.

Tuy nhiên, mặc dù khung chứa chính không phải là nút thắt cổ chai, nhưng nó có một điểm gây ra lỗi trong nền tảng. Để quản lý điều này, JADE cung cấp dịch vụ Main Replication Service để đảm bảo nền tảng JADE vẫn hoạt động một cách đầy đủ ngay cả trong trường hợp main container thất bại. Với dịch vụ này, bộ phận quản trị có thể điều khiển mức độ chịu đựng lỗi của nền tảng, mức độ co giãn của nền tảng và mức phân tán của nền tảng. Trong trường hợp cực đoan, mỗi khung chứa có thể được tạo ra để gia nhập Main Replication Server và hoạt động như một phần của tầng điều khiển.

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 với tên của nền tảng. Địa chỉ của tác tử là địa chỉ giao dịch được kế thừa từ nền tảng, mỗi địa chỉ nền tảng 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 nền tảng. Nó là điểm kết nối cho tất cả các tác tử muốn tương tác để truy cập trang trắng của nền tảng 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.

Tất cả các phần mềm liên quan tới JADE có thể được download từ trang web của JADE: http://jade.tilab.com.

4.4 Các gói của JADE

Các gói chính là:

- *jade.core* cài đặt lõi của JADE, môi trường thời gian chạy phân tán hỗ trợ toàn bộ nền tảng và các công cụ của nó. Nó chứa lớp gốc *jade.core.agent* cũng như các lớp run-time cơ bản cần để thực thi các container tác tử. Nó còn chứa 1 tập các gói con, mỗi gói thực thi 1 sức năng đặc biệt ở mức lõi. Đó là:
  - *jade.core.event* cài đặt dịch vụ thông báo các sự kiện phân tán. Nó giúp cho người dùng thấy được các sự kiện phát sinh bởi các thành phần phân tán khác nhau trong 1 nền tảng;
  - *jade.core.management* cài đặt dịch vụ quản lý vòng đời tác tử phân tán;
  - *jade.core.messaging* cài đặt dịch vụ phân tán thông điệp;
  - *jade.core.mobility* cài đặt dịch vụ nhân bản và di chuyển tác tử, bao gồm việc truyền cả tạng thái và mã nguồn của 1 tác tử;
  - *jade.core.nodeMonitoring* cho phép các container quản lý lẫn nhau và phát hiện các container không quản lý được hoặc đã chết;
  - *jade.core.replication* cho phép tái tạo 1 main container mới nếu có lỗi nghiêm trọng trong main container ban đầu.
  - *jade.core.behaviors* là 1 gói con của *jade.core* chứa 1 hệ thống các hành vi lỗi độc lập với ứng dụng. Một hành vi JADE biểu diễn 1 tác vụ mà 1 tác tử thực hiện.
- *jade.content* và các gói con của nó chứa 1 tập các lớp hỗ trợ lập trình viên tạo và thao tác với các biểu thức nội dung phức tạp theo 1 ngôn ngữ nội dung cho trước và ontology. Nó chứa tất cả các cấu trúc được mã hóa để tự động chuyển đổi giữa cách biểu diễn nội trong JADE và định dạng truyền nội dung thông điệp theo FIPA.

- *jade.domain* chứa phần cài đặt của các tác tử AMS và DF, theo như chuẩn FIPA, cùng với các mở rộng đặc tả JADE của chúng sẽ được nói đến sau.
- *jade.gui* chứa 1 vài thành phần Java chung và các icon dùng để xây dựng các giao diện GUI dựa trên Swing dùng cho các tác tử JADE.
- jade.imtp chứa các cài đặt JADE IMTP (Giao thức truyền thông điệp bên trong). Về nguyên tắc, gói con jade.imtp.rmi là IMTP mặc định của JADE dựa trên Java RMI.
- *jade.lang.acl* chứa chỗ trợ cho FIPA ACL chứa lớp ACLMessage, chương trình phân tích, mã hóa, và 1 lớp hỗ trợ các mẫu biểu diễn của các thông điệp ACL.
- *jade.mtp* chứa 1 tập các giao diện Java nên được cài đặt bởi JADE MTP. Nó còn chứa 2 gói con với 1 gói cài đặt dựa trên giao thức HTTP (là cài đặt mặc định) và 1 gói dựa trên giao thức IIOP.
- *jade.proto* chứa các cài đặt của 1 vài giao thức tương tác chung, trong đó 1 số được đặc tả bởi FIPA.
- *jade.tools* chứa cài đặt của tất cả các công cụ đồ họa JADE.
- *jade.util* chứa 1 số lớp hữu dụng khác.
- *jade.wrapper* cùng với các lớp *jade.core.Profile* và *jade.core.Runtime* cung cấp hỗ trợ giao diện đang chạy JADE cho phép các ứng dụng Java bên ngoài sử dụng JADE như 1 thư viện.

4.5 Dịch vụ vận chuyển thông điệp

Theo các đặc tả FIPA, một Dịch vụ vận chuyển thông điệp (MTS) là 1 trong 3 dịch vụ quan trọng mà mọi nền tảng tác tử phải cung cấp (2 dịch vụ khác là Dịch vụ quản lý MS và Xúc tiến thư mục DS). Một MTS quản lý tất cả các thông điệp trao đổi bên trong và giữa các nền tảng.

4.5.1 Các giao thức truyền thông điệp

Để hỗ trợ tương tác giữa các nền tảng khác nhau (ví dụ, với nền tảng không phải của JADE), JADE cài đặt tất cả các chuẩn Giao thức vận chuyển thông điệp (MTP) định nghĩa bởi FIPA, nơi mỗi MTP bao gồm 1 định nghĩa giao thức vận chuyển và 1 chuẩn mã hóa của phong bì chứa thông điệp.

Mặc định, JADE luôn luôn khởi động bằng 1 MTP dựa trên HTTP được khởi tạo trong main container, không MTP nào được hoạt động trong các container thường. Nó tạo ra một server socket trên host main container và lắng nghe các kết nối với qua HTTP tại URL đặc tả trong dòng lệnh thứ 2 phía trên. Khi 1 kết nối tới được xác định và thông điệp hợp lệ được nhận từ kết nối, MTP sẽ gửi thông điệp đến đích cuối cùng, thường là 1 tác tử nằm trong nền tảng phân tán. Phía trong, nền tảng sử dụng 1 giao thức vận chuyển tên là IMTP (Giao thức truyền thông điệp bên trong) sẽ được mô tả trong phần tiếp theo. JADE thực hiện việc truyền thông điệp cho cả các thông điệp đến và đi sử dụng 1 bảng đơn bước yêu cầu IP trực tiếp giữa các container.

Sử dụng các lệnh tùy chọn, vô số MTP có thể hoạt động trong mỗi JADE container, bao gồm cả các MTP cài đặt các giao thức truyền khác nhau. MTP còn thể được 'nhúng' và khởi tạo tại thời gian chạy nhờ sử dụng RMA GUI. Khi 1 MTP hoạt động trong 1 nền tảng, Nền tảng JADE sẽ nhận được 1 địa chỉ truyền mới, 1 đầu mút nơi các thông điệp có thể nhận. Địa chỉ này còn được thêm vào trong cấu trúc dữ liệu sau:

- Thông tin nền tảng, có thể đọc từ AMS nhờ lệnh get-description.
- Toàn bộ các đối tượng ams-tác tử-description chứa trong kho chứa AMS, có thể đọc nhờ 1 lệnh tìm kiếm.
- Định danh tác tử (AID) cục bộ của bất kỳ tác tử trong bất kỳ container nào có thể đọc nhờ phương thức getAID( ) của lớp tác tử.

Giao diện MTP mô hình 1 kênh song hướng có thể vừa gửi và nhận các thông điệp ACL bằng cách kế thừa giao diện jade.mtp.OutChannel và jade.mtp.InChannel là các kênh 1 hướng. Giao diện jade.mtp.TrasnportAddress chỉ đơn giản biểu diễn 1 URL cung cấp truy nhập đến các trường như giao thức, host, cổng và tệp. Khi các MTP được liệt kê trong Bảng 4.3 có trong public domain, mỗi MTP được triển khai dưới dạng các tệp jar riêng lẻ.

Giao thức truyền	Mã hóa thông điệp	Nhà phát triển
HTTP và HTTPS	XML	Đại học Autonoma ở Barcelona, Tây Ban Nha (UAB)
IIOP (bản cài đặt của Sun)	CORBA IDL	Nhóm JADE
IIOP (bản cài đặt của ORBacus)	CORBA IDL	Giovanni Rimassa, Đại học Parma, Italia
JMS	Cấu trúc dữ liệu Java	Edward Curry, đại học Galway
Jabber XMPP	Cấu trúc dữ liệu Java	Đại học Politecnica ở Valencia, Tây Ban Nha

Hình 4.3: Các giao thức truyền thông trong JADE hiện nay

Trong khi HTTP và IIPO MTP được đính kèm trong bản phân phối chính JADE, các giao thức còn lại đều phải download dưới dạng bản add-on tại trang chủ JADE. HTTP là MTP mặc định để chạy với main container. HTTP được chọn làm MTP mặc định vì bản cài đặt cung cấp bởi UAB có những ưu điểm sau đây:

- Số cổng cục bộ của các kết nối đến và đi có thể được chọn cho cấu hình firewall sử dụng các biến jade_mtp_http_port và jade_mtp_http_outPort.
- Proxy có thể được cấu hình thông qua các kết nối bất biến: thay vì thực hiện bắt tay TCP với mỗi thông điệp, các kết nối có thể được lưu lại và sử dụng lại khi các thông điệp được trao đổi thường xuyên giữa 2 nền tảng khác nhau.
- HTTPS có thể được sử dụng để thiết lập bảo mật và các kênh chứng thực giữa các nền tảng. Để sử dụng HTTPS, 1 địa chỉ truyền đi phải đơn giản bắt đầu bằng https. Tất nhiên phải lưu ý rằng, mặc dù HTTPS tăng cường bảo mật, nó vẫn gặp phải 1 số khuyết điểm khi thực hiện; ước tính sơ lược cho thấy rằng HTTPS MTP chậm hơn 15% so với HTTP MTP chuẩn.

JADE RMA cho phép quản lý MTP linh hoạt bằng việc cho phép kích hoạt hoặc tắt chúng khi nền tảng đang chạy. Nhấn chuột phải vào 1 nút cây trong bảng bên trái của RMA GUI sẽ hiện ra 1 menu trong đó có 2 mục là Install a new MTP và Unistall an MTP. Lựa chọn đầu sẽ tạo ra 1 cửa sổ để người dùng chọn MTP mới để cài đặt, tên đầy đủ của lớp cài đặt giao thức, và địa chỉ truyền lắng nghe được ưu tiên. Nếu chọn Unistall an MTP, 1 cửa sổ sẽ xuất hiện để người dùng có thể chọn MTP trong danh sách đang hoạt động để gỡ nó ra khỏi nền tảng.

Một vài ứng dụng có thể không cần tới các giao tiếp bên ngoài nền tảng cục bộ. Trong trường hợp đó, lệnh tùy chọn –nompt sẽ tạo ra 1 HTTP MTP mặc định trong main container. Tất nhiên nó sẽ cách ly nền tảng khỏi mọi giao tiếp với các nền tảng từ xa. Lưu ý rằng 1 container từ xa chỉ là 1 container mà không nằm chung host với main container, nhưng vẫn nằm chung nền tảng; nói cách khác, 1 container từ xa không được là 1 phần của 1 nền tảng từ xa. Các container trong cùng nền tảng luôn giao tiếp bằng JADE IMTP.

4.5.2 Giao thức truyền thông điệp nội bộ (IMTP)

JADE IMTP chuyên dùng để trao đổi thông điệp giữa các tác tử sống trong các container khác nhau trong cùng 1 nền tảng. Nó tương đối khác với các MTP ngoài nền tảng, như HTML. Thứ nhất, vì nó chỉ được dùng cho việc giao tiếp bên trong nền tảng, nên không cần phải tương thích với các chuẩn FIPA. Thực tế JADE

IMTP không chỉ được dùng để truyền thông điệp mà còn truyền các lệnh bên trong cần thiết để quản lý nền tảng phân tán, cũng như giám sát trạng thái của các container từ xa.

JADE được thiết kế để cho phép lựa chọn IMTP trong thời gian nền tảng chạy. Hiện tại, đã có 2 các cài đặt ITMP chính. Một cách dựa trên Java RMI và là tùy chọn mặc định. Cách thứ hai dựa trên 1 giao thức sử dụng TCP socket giúp loại bỏ đi sự thiếu sót hỗ trợ Java RMI trong môi trường J2ME; nó được khởi động mặc định khi chạy nền tảng JADE LEAP và sẽ được mô tả trong Chương 8. Cả 2 cách cài đặt này đều cung cấp các lựa chọn cấu hình cho phép điều chỉnh IMTP theo mạng và các thiết bị nhất định.

Giao thức truyền thông điệp nội bộ theo chuẩn RMI (RMI-IMTP)

RMI-IMTP được cái đặt bởi gói jade.imtp.rmi. Khi main container khởi động, nó sẽ tìm 1 đăng ký RMI trong host cục bộ và gọi đến các đối tượng tham chiếu; nếu không tìm thấy, nó sẽ tạo ra 1 đăng ký mới. Khi 1 container thường khởi động, nó sẽ xác định đăng ký RMI trên host đặc tả main container và tìm đối tượng tham chiếu của main container. Sau đó nó sẽ gọi phương thức từ xa addNode ( ) của main container để tham gia nền tảng và đăng ký tham chiếu của nó với main container.

Các thông điệp tác tử và thông tin điều khiển hệ thống được trao đổi giữa các container được cài đặt thông qua 1 mẫu lệnh khi nút yêu cầu (ví dụ, 1 container) tạo ra 1 đối tượng Command và truyền đi đối tượng này, với 1 yêu cầu thực thi, đến nút thực thi. Hai biến dòng lệnh sau có sẵn trong RMI-IMTP:

-host <hostName>

sẽ đặc tả host đang chờ main container để đăng ký với nó; giá trị mặc định là localhost. Lựa chọn này cũng được sử dụng khi chạy main container để override giá trị của localhost, ví dụ để đọc toàn bộ tên miền của host với –host anduril.cselt.it khi localhost chỉ trả về là 'anduril'.

- port <portNumber>

sẽ đặc tả số cổng mà đăng ký RMI được tạo ra bởi main container để nhận các yêu cầu tìm kiếm. Giá trị mặc định là 1099.

4.6 Cửa sổ quản trị JADE

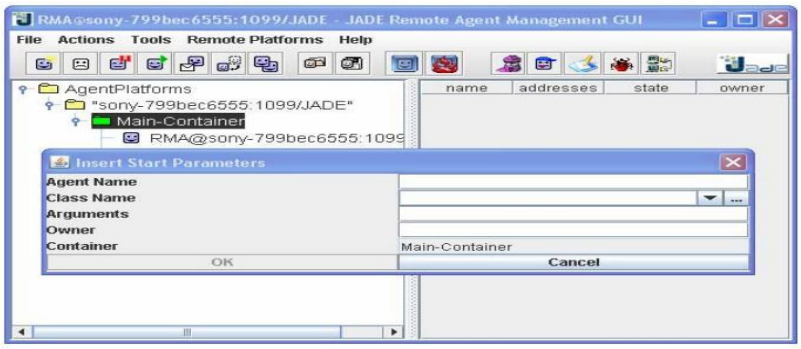
JADE RMA (Remote Monitoring Agent) là một công cụ hệ thống thực thi một giao diện quản lý nên tảng đồ họa. Công cụ được thực thi bởi lớp *jade.tools.rma.rma* nhưng nó thường được bắt đầu trực tiếp từ dòng lệnh sử dụng tuỳ chọn –gui. Nó cung cấp một giao diện đồ hoạ để giám sát và quản lý nền tảng JADE phân tán được tạo thành từ một hoặc một số host và các nút container. Một số RMA có thể được khởi hoạt trong cùng một nền tảng nếu một tên tác tử khác được đăng kí cho mỗi thể hiện.

Tại lúc khởi động RMA tác tử đăng kí với AMS để được thông báo tất cả các sự kiện cấp nền tảng, Hình 4.4 hiện thị giao diện sử dụng đồ hoạ của nó. Panel trái cung cấp cái nhìn của mô hình nền tảng được biểu diễn như một cây của các container các lá là các tác tử. Panel này được thực thi bởi lớp jade.gui.AgentTree và được sử dụng lại bởi tất cả các công cụ khác. Nói cụ thể, có 3 kiểu của nút: tác tử nền tảng, container và tác tử.

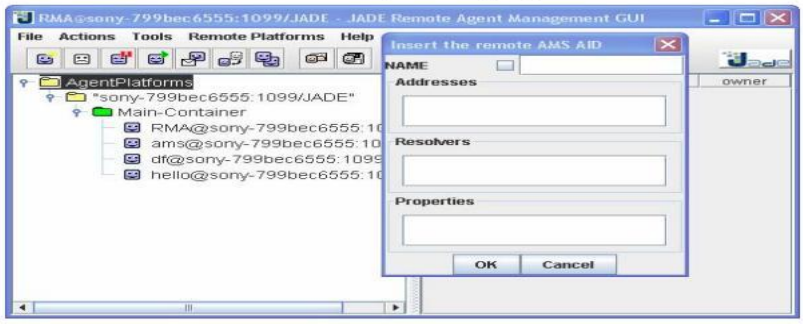
Nếu một tác tử được chọn, menu sổ xuống cho phép tác tử được treo (suspend), hồi phục lại (resume), giết (kill), tạo bản sao (clone), lưu (saved), đóng băng (frozen) hoặc di chú đến một container khác. Nó cũng cho phép cấu hình và gửi một thông điệp tuỳ chỉnh, đặc biệt.

Nếu một container được chọn, menu sổ xuống cho phép tạo một tác tử mới, tải một tác tử đang tồn tại, cài đặt hoặc xoá bỏ một MTP, lưu/tải container bao gồm tất cả các tác tử của nó và kết thúc container.

Chú ý rằng gốc của cây được gọi là “Agent Platform”. Nó biểu thị sự thật rằng RMA có thể được sử dụng để điều khiển một tập nền tảng được cung cấp chúng là tất cả FIPA – compliant. Tất nhiên, mức độ của điều khiển được giới hạn khi việc tương tác với một nền tảng ở xa khi đó chỉ việc quản lý các thông điệp và action được định nghĩa trong FIPA có thể được sử dụng, thay vì thông qua JADE IMTP trong bất kì Nền tảng JADE nào. Để giao tiếp với một nền tảng ở xa, nhận dạng của AMS của nó phải được cung cấp (ví dụ: AMS AID), nó phải bao gồm tên và ít nhất 1 địa chỉ truyền (transport address) hợp lệ. Điều này hiện thị trong Hình 4.5



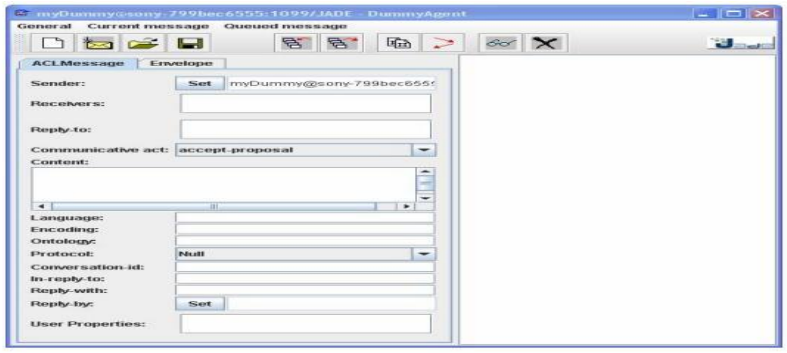
Hình 4.4: Giao diện tác tử mới



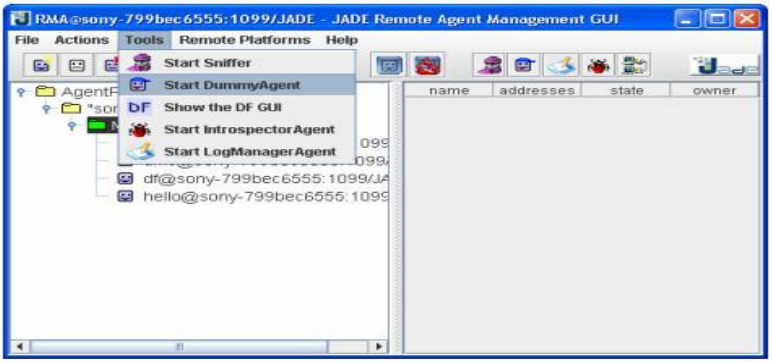
Hình 4.5: Giao tiếp với Nền tảng từ xa

4.6.1 Dummy Agent

Dummy Agent là một công cụ rất đơn giản hữu dụng cho việc gửi các tác nhân kích thích theo dạng các ACL thông điệp tuỳ chỉnh để kiểm tra hành vi của các tác tử khác. Khả năng của nó là gửi và nhận các thông điệp tuỳ chỉnh có thể được tạo ra sử dụng một GUI đơn giản và được tải từ một file. Khi một ứng dụng tác tử được khởi hoạt, một *Dummy Agent* có thể được sử dụng để giả vờ nó bằng việc gửi các thông điệp được người dùng chỉ ra và việc phân tích các phản ứng của nó trong thời hạn các thông điệp được nhận. Hình 4.6 hiển thị *Dummy Agent* GUI với panel bên phải dành để hiện thị danh sách các thông điệp gửi và nhận. Panel bên trái sử dụng để tạo ra các thông điệp tuỳ chỉnh.



Hình 4.6: Kết quả chạy DummyAgent từ dòng lệnh

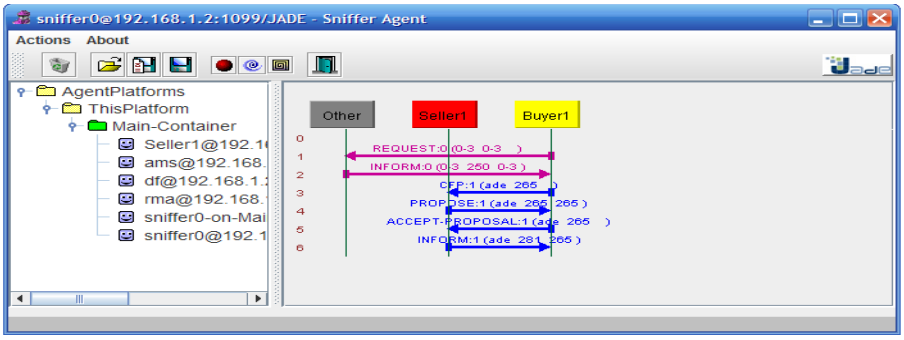


Hình 4.7: Kết quả chạy DummyAgent từ giao diện nền tảng

4.6.2 Sniffer Agent

Trong khi tất cả các công cụ khác phần lớn được sử dụng cho việc gỡ lỗi một tác tử đơn, công cụ này được sử dụng rộng rãi cho việc gỡ lỗi, hoặc đơn giản là viết các cuộc nói chuyện giữa các tác tử. "sniffer" đăng kí với nền tảng AMS để được thông báo tất cả các sự kiện của nền tảng và tất cả các sự trao đổi thông điệp giữa một tập các tác tử xác định. Hình 4.8 hiện thị GUI của *Sniffer Agent*. Panel trái là trình duyệt tương tự như RMA, nhưng được sử dụng cho việc duyệt tác tử nền tảng và việc chọn các tác tử được sniff. Phần bên phải cung cấp biểu diễn đồ họ của các thông điệp được trao đổi giữa các tác tử được sniff, nơi mỗi mũi tên biểu diễn một thông điệp và mỗi màu xác định một cuộc nói chuyện.

Khi người sử dụng quyết định sniff một tác tử hoặc một nhóm các tác tử, mỗi thông điệp gửi đi hoặc đến, tác tử được lưu vết và được hiện thị trong sniffer GUI. Người sử dụng có thể chọn và xem chi tiết của mỗi thông điệp, lưu thông điệp vào đĩa như một file văn bản hoặc *serialize* một cuộc nói chuyện như một file nhị phân.



Hình 4.8: Kết quả chạy SnifferAgent

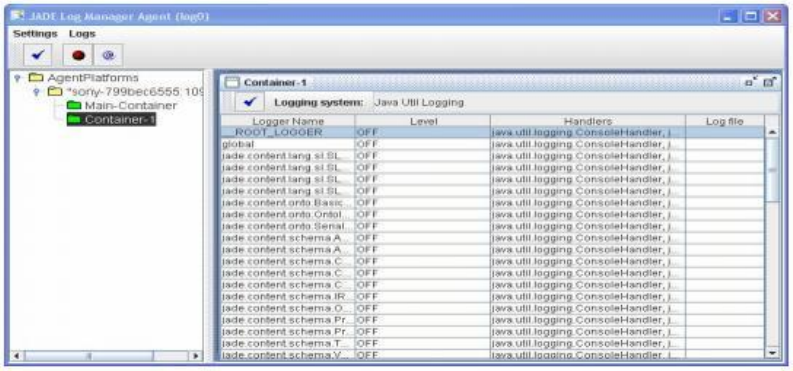
4.6.3 Introspector Agent

Trong khi Sniffer Agent có ích trong việc đánh hơi, giám sát và *debug* các cuộc hội thoại giữa các tác tử, thì Introspector Agent được dùng để *debug* hành vi

của một tác tử. Công cụ này cho phép giám sát việc thực thi của tác tử, cụ thể là những hành vi nào được thực thi, những hành vi nào được đưa vào hàng đợi, và cho phép giám sát những phản ứng của chúng đối với kích thích bên trong. Hình 4.9 biểu diễn giao diện của Introspector Agent khi đang giám sát tác tử DF.

Hình 4.9: Giao diện của Introspector Agent khi đang giám sát tác tử DF

Sau khi container mới được tạo ra trên nền tảng, khởi động Log Manager Agent trên container đó, ta thấy mức độ logging của container này đều được thiết lập là OFF và các handler đều được thiết lập là java.util.logging.ConsoleHandler:



Hình 4.10: Giao diện Log Manager Agent của Container-1

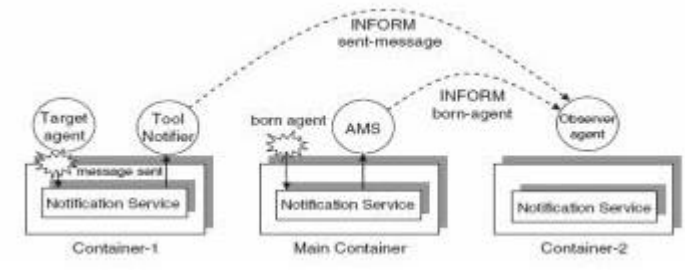
4.6.4 Dịch vụ thông báo sự kiện và mô hình công cụ JADE

Dịch vụ thông báo sự kiện (Event Notification Service - ENS) là một dịch vụ ở mức nền tảng quản lý các thông báo phân tán của tất cả các sự kiện được sinh ra bởi mỗi node của nền tảng. Mỗi khi có một sự kiện được sinh ra bởi một container, nó sẽ bị chặn bởi ENS và được định tuyến tới mọi tác tử đã đặt trước để được thông báo về các kiểu sự kiện. Nếu không có tác tử nào đặt trước thì ENS có hiệu năng không đáng kể. Thực tế, những node có hiệu năng thấp là container nơi cư trú của các tác tử đã đặt trước và là container sinh ra sự kiện được thông báo. Vì tất cả các tác tử công cụ có thể hoạt động khi cần, thậm chí tại thời điểm chạy trong quá trình

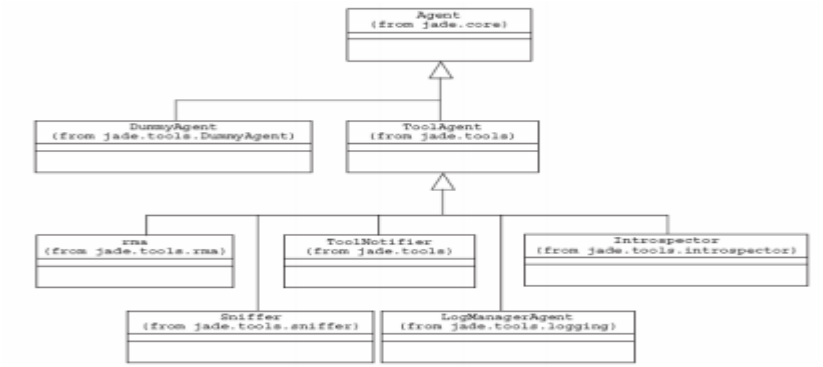
vận hành nền tảng, việc cải thiện hiệu năng có thể đạt được bằng cách chỉ bắt đầu chúng khi cần thiết. Có 4 loại sự kiện chính:

- Sự kiện liên quan đến vòng đời, còn được gọi là sự kiên kiểu nền tảng (nền tảng-type) vì chúng luôn liên quan đến container chính. Những sự kiện này liên quan đến những thay đổi trong vòng đời tác tử (ví dụ: born, dead, moved, suspended, resumed, frozen, thawed) và liên quan đến những thay đổi trong vòng đời container (ví dụ: added, removed).
- Sự kiện kiểu MTP-type được sinh ra bởi nền tảng khi một MTP được kích hoạt (kết thúc) và khi một thông điệp được gửi/nhận bởi/từ một MTP, cụ thể là khi có một số phiên truyền thông liên nền tảng (inter-nền tảng).
- Sự kiện kiểu *message-passing-type* được sinh ra khi một thông điệp ACL được gửi, nhận, định tuyến hoặc được đưa vào hàng đợi thông điệp. Chúng là những sự kiện mà Sniffer thường sử dụng để giám sát.
- Sự kiện kiểu *Agent-internal-type* liên quan đến những thay đổi trong trạng thái và hành vi của tác tử. Chúng là những sự kiện mà Introspector thường sử dụng để giám sát.

Các tác tử tương tác với ENS bằng cách trao đổi các thông điệp ACL với AMS. Kiểu sự kiện *message-passing-type* và *Agent-internal-type* nói cách khác chỉ được nằm trong container chính nơi tác tử tạo ra chúng cư trú. Việc chuyển chúng vào container chính thực tế sẽ làm giảm đáng kể hiệu năng của nền tảng. Điều này được thực hiện bằng các phương tiện của hành động SnifferOn và hành động DebugOn của JADEManagementOntology. Kết quả của hành động SniffOn là một tác tử phụ ToolNotifier được tạo ra trong container mà tác tử đích cư trú ở đó, tác tử này lắng nghe sự kiện *message-passing* cục bộ và chuyển tiếp chúng tới tác tử quan sát. Hệ thống thông báo sự kiện của JADE được minh họa trong Hình 4.11:



Hình 4.11: Hệ thống thông báo sự kiện của JADE



Hình 4.12: Biểu đồ các lớp công cụ của JADE

Mọi công cụ của JADE, ngoại trừ DummyAgent, đều kế thừa từ lớp jade.tools.ToolAgent – lớp cung cấp khả năng nhận các thông báo theo một cách thống nhất. Hình 4.12 minh họa biểu đồ lớp UML về các công cụ của JADE. Điều này cho phép một số tính năng và sự đơn giản hóa quan trọng:

- Vòng đời của một công cụ JADE có thể được quản lý như các tác tử khác của nền tảng.
- Khả năng truyền thông điệp của tác tử cơ sở có thể được sử dụng để cho phép sự tương tác giữa công cụ và AMS, cụ thể là việc đặt trước các thông báo sự kiện của nền tảng.
- Một số thể hiện của cùng một công cụ có thể cùng tồn tại trên cùng một nền tảng và thậm chí trong cùng container.

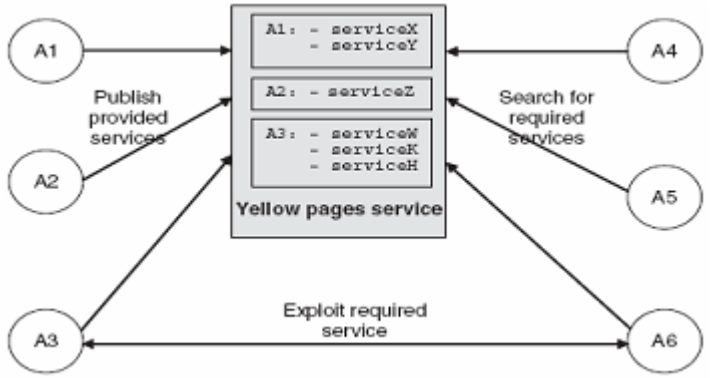
Lớp jade.tools.ToolNotifier cài đặt các tác tử phụ trợ được dùng để chuyển tiếp các sự kiện *message - passing* và *Agent - internal* tới các tác tử quan tâm chính là một ToolAgent. Cách này cho phép phát hiện xem tác tử đích hoặc tác tử quan sát đã kết thúc hay chưa.

4.7 Khám phá tác tử – Dịch vụ trang vàng (Yellow Pages)

Khi viết code chúng ta giả định rằng có một tập cố định các tác tử người bán (thông qua mỗi tác tử người mua như là các đối số ban đầu). Trong chương này chúng ta loại bỏ giả định này bằng cách khai thác dịch vụ các trang vàng được cung cấp bởi Nền tảng JADE cho phép các tác tử người mua tự động phát hiện các tác tử người bán sẵn có tại một điểm được đưa ra trong thời gian.

4.7.1 DF Agent

Một dịch vụ "yellow pages" cho phép các tác tử đưa ra các mô tả của một hoặc nhiều các dịch vụ mà chúng cung cấp theo thứ tự mà các tác tử khác có thể dễ dàng phát hiện và khai thác chúng. Bất kỳ tác tử nào cũng có thể đăng ký (xuất bản) các dịch vụvà tìm kiếm cho các dịch vụ (khai thác). Đăng ký, xóa, sửa đổi và tìm kiếm có thể được thực hiện bất cứ lúc nào trong thời gian sống của tác tử.



Hình 4.13: Dịch vụ trang vàng

Dịch vụ các trang vàng trong Jade, phù hợp với các đặc tả quản lý tác tử của FIPA (FIPA Agent Management), được cung cấp bởi một tác tử đặc biệt gọi là DF (Directory Facilitator). Mỗi FIPA-compliant nền tảng nên đăng cai một tác tử DF mặc định (tên địa phương là 'df@<nền tảng-name>'). Các tác tử DF khác có thể được triển khai nếu được yêu cầu và một vài tác tử DF (bao gồm cả mặc định) có thể được tổ chức thành liên đoàn để cung cấp một danh mục liệt kê duy nhất các trang vàng phân tán.

4.7.2 Tương tác với DF Agent

DF là một tác tử, nó có thể tương tác với nó như với bất kỳ tác tử khác bằng cách trao đổi các thông điệp ACL sử dụng một ngôn ngữ có nội dung thích hợp (ví dụ ngôn ngữ SL0) và một ontology thích hợp (ví dụ FIPA – Agent - management ontology) như được định nghĩa trong các đặc tả FIPA. Để đơn giản hóa các tương tác này, JADE cung cấp lớp jade.domain.DFService có thể xuất bản và tìm kiếm các dịch vụ qua nhiều lời gọi phương thức.

4.7.2.1 Công bố dịch vụ

Một tác tử muốn đưa ra 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 cung cấp tùy chọn danh sách các ngôn ngữ và ontology để các tác tử khác sử dụng để tương tác với nó. Mỗi mô tả dịch vụ được công bố phải bao gồm loại dịch vụ, tên dịch vụ, các ngôn

ngữ và các ontology được yêu cầu để sử dụng dịch vụ và tập các thuộc tính đặc trưng của dịch vụ dưới dạng cặp giá trị khóa.

4.7.2.2 Tìm kiếm dịch vụ

Một tác tử muốn tìm kiếm các dịch vụ phải cung cấp DF với một mô tả mẫu. Kết quả tìm kiếm là một danh sách tất cả các mô tả mà phù hợp với mẫu đã cung cấp. Theo các đặc tả FIPA, một mô tả phù hợp với mẫu nếu tất cả các lĩnh vực được đặc tả trong mẫu được diễn tả trong mô tả có giá trị như nhau.

Jade DF cũng cung cấp một kỹ thuật cho phép các tác tử được thông báo ngay khi các tác tử khác đăng ký hoặc xóa khỏi các dịch vụ. Khai thác kỹ thuật này (điều đó có thể sẽ thích hợp hơn trong trường hợp của chúng ta) yêu cầu khởi tạo một giao thức FIPA-Subcribe với DF.

CHƯƠNG 5: THỰC NGHIỆM

Chương này giới thiệu về một bài toán đơn giản ứng dụng công nghệ tác tử trên nền tảng JADE, đó là bài toán mua bán sách giữa hai tác tử.

5.1 Mô tả bài toán

Đây là một bài toán đơn giản để mô tả quá trình mua bán sách giữa 2 tác tử. Ta sẽ xây dựng một BookSellerAgent có chức năng bán sách, một giao diện đơn giản được xây dựng để BookSellerAgent đưa ra tên và giá của mặt hàng. Các mặt hàng sách được BookSellerAgent cung cấp sẽ được đưa vào một Catalogue có sẵn để lưu trữ.

Sau đó ta sẽ xây dựng một BookBuyerAgent để thực hiện mua sách của BookSellerAgent. Trong ứng dụng này sẽ không xây dựng giao diện của BookBuyertác tử mà bản thân các BookBuyerAgent sẽ thêm các mặt hàng cần mua bằng cách truyền tham biến trực tiếp.

Khi BookBuyerAgent cần mua một mặt hàng sách nào đó, nó sẽ đưa ra tên sách, sau đó thực hiện hỏi tất cả các BookSellerAgent, biết nếu chúng sẵn sàng bán, và như vậy sẽ cung cấp một giao dịch. BookSellerAgent nào có sách bán cho BookBuyerAgent thì nó sẽ đưa ra phản hồi.

Các BookBuyerAgent sau khi thực hiện mua thành công cuốn sách, sẽ tự động ngắt. Tiến trình được tiếp tục khi ta tạo các tác tử khác để thực hiện ứng dụng.

5.2 Minh họa bài toán

5.2.1 Xây dựng giao diện cho tác tử Seller

Ta sẽ xây dựng Lớp giao diện BookSellerGui cho phép nhập vào tên sách và giá bán. Nút “Add” được sử dụng để thêm tên sách và giá sách vào Catalogue:

```
addButton.addActionListener( new ActionListener() {
     public void actionPerformed(ActionEvent ev) {
          try {
          String title = titleField.getText().trim();
          String price = priceField.getText().trim();
myAgent.updateCatalogue(title,Integer.parseInt(price));
               titleField.setText("");
               priceField.setText("");
          }
          catch (Exception e) {
```

```
      JOptionPane.showMessageDialog(BookSellerGui.this,    "Invalid
values. "+e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
                        }
                  }} );
```

Phương thức WindowClosing sẽ ngắt Agent Seller khi cửa sổ giao diện đóng:

```
            addWindowListener(new WindowAdapter() {
                  public void windowClosing(WindowEvent e) {
                        myAgent.doDelete();
                  }
```

5.2.2 Xây dựng tác tử Seller

5.2.2.1 Khởi tạo tác tử Seller

Khởi tạo tác tử Seller, sử dụng phương thức Setup(). Trong đó Catalogue được định nghĩa là một Hashtable như sau:

```
catalogue = new Hashtable();
```

Sau đó giao diện BookSellertác tử sẽ được gọi ra từ lớp BookSellerGui:

```
myGui = new BookSellerGui(this);
myGui.showGui();
```

Tiếp đó ta phải đăng ký dịch vụ trang vàng (yellow pages) cho việc bán sách. Dịch vụ này sử dụng tác tử DF để giúp cho các tác tử đăng ký, đặc biệt giúp cho tác tử có sẵn sẽ dễ dàng tìm ra nhau:

```
DFAgentDescription dfd = new DFAgentDescription();
            dfd.setName(getAID());
            ServiceDescription sd = new ServiceDescription();
            sd.setType("book-selling");
            sd.setName("JADE-book-trading");
            dfd.addServices(sd);
            try {
                  DFService.register(this, dfd);
            }
            catch (FIPAException fe) {
                  fe.printStackTrace();}
```

BookSellerAgent khi khởi động có 2 hành vi:

Đây là hành vi đáp ứng câu hỏi từ tác tử Buyer khi có tác tử Buyer tìm kiếm tác tử Seller:

```
addBehaviour(new OfferRequestsServer());
```

Đây là hành vi đáp ứng yêu cầu mua hàng của tác tử Buyer khi có yêu cầu mua:

```
addBehaviour(new PurchaseOrdersServer());
```

5.2.2.2 Kết thúc tác tử Seller

Kết thúc một tác tử Seller, sử dụng phương thức takeDown() như sau:

```
protected void takeDown() {
            // Deregister from the yellow pages
            try {
                  DFService.deregister(this);
            }
            catch (FIPAException fe) {
                  fe.printStackTrace();
            }
            // Close the GUI
            myGui.dispose();
            // Printout a dismissal message
            System.out.println("Seller-Agent  "+getAID().getName()+"
terminating.");
      }
```

Sau khi gọi phương thức takeDown(), dịch vụ trang vàng sẽ bị ngắt và đóng giao diện của tác tử Seller lại, sau đó sẽ đưa ra thông báo tác tử Seller nào kết thúc.

5.2.2.3 Các hành vi của tác tử Seller

a. Thêm mặt hàng mới vào Catalogue

Các mặt hàng sau khi được tác tử Seller đưa ra sẽ được đưa vào Catalogue để lưu trữ, sử dụng phương thức UpdateCatalogue():

```
public void updateCatalogue(final String title, final int price) {
            addBehaviour(new OneShotBehaviour() {
                  public void action() {
                        catalogue.put(title, new Integer(price));
                        System.out.println(title+"   inserted   into
catalogue. Price = "+price);
                  }} ); }
```

b. Trả lời yêu cầu của tác tử Buyer

Đây là hành vi của tác tử Seller chấp nhận yêu cầu được gửi đến từ tác tử Buyer. Nếu cuốn sách được yêu cầu có trong Catalogue thì tác tử Seller sẽ gửi tin nhắn yêu cầu xác nhận giá. Nếu không thì tin nhắn từ chối sẽ được gửi lại.

```
private class OfferRequestsServer extends CyclicBehaviour {
          public void action() {
MessageTemplate mt =
MessageTemplate.MatchPerformative(ACLMessage.CFP);
ACLMessage msg = myAgent.receive(mt);
                if (msg != null) {
                     // CFP Message received. Process it
                String title = msg.getContent();
                ACLMessage reply = msg.createReply();
                Integer price = (Integer) catalogue.get(title);
                     if (price != null) {
// The requested book is available for sale. Reply with the price
          reply.setPerformative(ACLMessage.PROPOSE);
          reply.setContent(String.valueOf(price.intValue()));
                     }
                     else {
                // The requested book is NOT available for sale.
                     reply.setPerformative(ACLMessage.REFUSE);
                     reply.setContent("not-available");
                     }
                     myAgent.send(reply);
                }
                else {
                     block();}}}
```

c. Chấp nhận giao dịch

Đây là hành vi của tác tử Seller chấp nhận đơn đặt hàng từ các tác tử Buyer. tác tử Seller sẽ bỏ cuốn sách khỏi Catalogue và gửi một thông báo "INFORM" tới tác tử Buyer để thông báo là giao dịch được thực hiện thành công.

```
private class PurchaseOrdersServer extends CyclicBehaviour {
          public void action() {
                MessageTemplate mt =
MessageTemplate.MatchPerformative(ACLMessage.ACCEPT_PROPOSAL);
                ACLMessage msg = myAgent.receive(mt);
                if (msg != null) {
                // ACCEPT_PROPOSAL Message received. Process it
                String title = msg.getContent();
```

```
                    ACLMessage reply = msg.createReply();
                    Integer price = (Integer) catalogue.remove(title);
                         if (price != null) {
                         reply.setPerformative(ACLMessage.INFORM);
                         System.out.println(title+" sold to Agent
                         "+msg.getSender().getName());}
                         else {
// The requested book has been sold to another buyer in the
meanwhile .
                         reply.setPerformative(ACLMessage.FAILURE);
                         reply.setContent("not-available");}
                         myAgent.send(reply);}
                    else {
                         block();   }}}}
```

5.2.3 Xây dựng tác tử Buyer

Xây dựng lớp BookBuyerAgent với các phương thức khởi tạo, kết thúc và các hành vi của tác tử Buyer.

Đầu tiên ta sẽ đưa ra tên sách để mua:

```
private String targetBookTitle;
```

Tiếp đó ta sẽ đưa ra danh sách các tác tử Seller được biết đến:

```
private AID[] sellerAgents;
```

5.2.3.1 Khởi tạo tác tử Buyer

Để khởi tạo BookBuyerAgent, dùng phương thức Setup(), trong đó truyền tham biến là tên sách vào:

```
Object[] args = getArguments();
           if (args != null && args.length > 0) {
           targetBookTitle = (String) args[0];
           System.out.println("Target book is "+targetBookTitle);
```

Trong Setup() xây dựng một hành vi của tác tử Buyer, đó là đưa ra danh mục các yêu cầu mua hàng gửi tới tác tử Seller mỗi phút một lần:

```
addBehaviour(new TickerBehaviour(this, 60000) {
                    protected void onTick() {
           System.out.println("Trying to buy"+targetBookTitle);
           // 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(myAgeny,
template);
      System.out.println("Found the following seller Agents:");
      sellerAgents = new AID[result.length];
                        for (int i = 0; i < result.length; ++i)
                {
                      sellerAgents[i] = result[i].getName();
                System.out.println(sellerAgents[i].getName());
                              }
                        }
                        catch (FIPAException fe) {
                              fe.printStackTrace();
                        }

                // Perform the request
                myAgent.addBehaviour(new RequestPerformer());
                      }
                } );}
```

5.2.3.2 Kết thúc tác tử Buyer

Kết thúc tác tử Buyer sử dụng phương thức takeDown():

```
protected void takeDown() {
// Printout a dismissal message
System.out.println("Buyer-Agent "+getAID().getName()+"
terminating."); }
```

5.2.3.3 Các hành vi của tác tử Buyer

Xây dựng lớp RequestPerformer kế thừa lớp Behaviour, trong đó sử dụng phương thức Action() mô tả các hoạt động tìm phản hồi từ các tác tử Seller, tìm kiếm mặt hàng mua, thỏa thuận giao dịch với tác tử Seller:

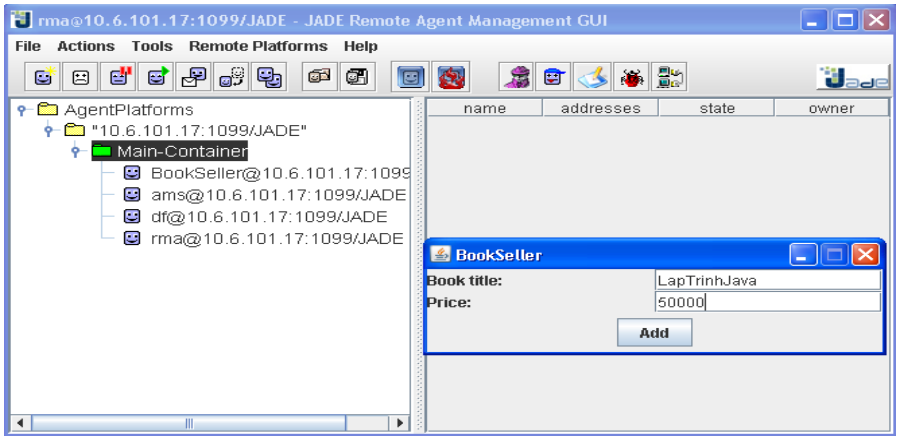
```
private class RequestPerformer extends Behaviour {
private AID bestSeller; // The Agent who provides the best offer
private int bestPrice;  // The best offered price
private int repliesCnt = 0; // The counter of replies from seller
Agents
private MessageTemplate mt; // The template to receive replies
      private int step = 0;
            public void action() {
                  switch (step) {
                  case 0:
                        // Send the cfp to all sellers
                  ACLMessage cfp = new ACLMessage(ACLMessage.CFP);
                  for (int i = 0; i < sellerAgents.length; ++i) {
                        cfp.addReceiver(sellerAgents[i]);
                        }
                        cfp.setContent(targetBookTitle);
                        cfp.setConversationId("book-trade");
      cfp.setReplyWith("cfp"+System.currentTimeMillis());
                        myAgent.send(cfp);
                        // Prepare the template to get proposals
```

```
                        mt =
MessageTemplate.and(MessageTemplate.MatchConversationId("book-
trade"),
            MessageTemplate.MatchInReplyTo(cfp.getReplyWith()));
                        step = 1;
                        break;
                  case 1:
            // Receive all proposals/refusals from seller Agents
                        ACLMessage reply = myAgent.receive(mt);
                        if (reply != null) {
                              // Reply received
                  if (reply.getPerformative() == ACLMessage.PROPOSE)
{
                              // This is an offer
                  int price = Integer.parseInt(reply.getContent());
                              if (bestSeller == null || price < bestPrice)
{
                                    // This is the best offer at present
                                    bestPrice = price;
                                    bestSeller = reply.getSender();
                                          }
                                    }
                                    repliesCnt++;
                                    if (repliesCnt >= sellerAgents.length)
{
                                    // We received all replies
                                    step = 2;
                                    }
                              }
                              else {
                                    block();
                              }
                              break;
                  case 2:
// Send the purchase order to the seller that provided the best
offer
ACLMessage order = new ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
                        order.addReceiver(bestSeller);
                        order.setContent(targetBookTitle);
                        order.setConversationId("book-trade");
            order.setReplyWith("order"+System.currentTimeMillis());
                        myAgent.send(order);
// Prepare the template to get the purchase order reply
                        mt =
MessageTemplate.and(MessageTemplate.MatchConversationId("book-
trade"),
      MessageTemplate.MatchInReplyTo(order.getReplyWith()));
                        step = 3;
                        break;
                  case 3:
                        // Receive the purchase order reply
                        reply = myAgent.receive(mt);
                  if (reply != null) {
                        // Purchase order reply received
                  if (reply.getPerformative() == ACLMessage.INFORM)
{
```

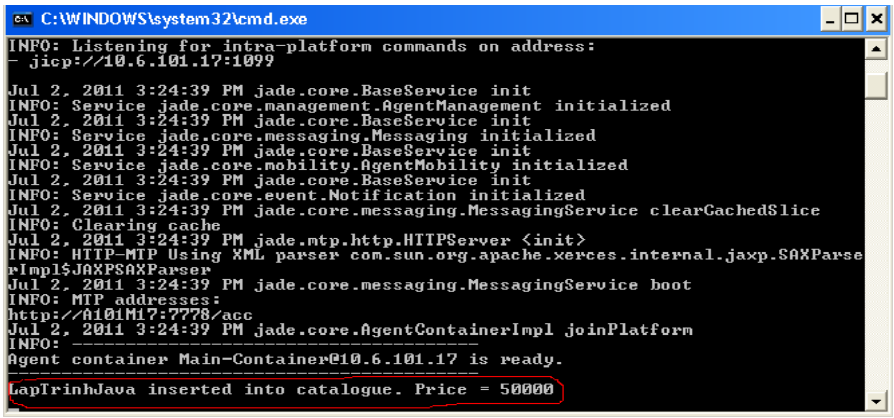
```
                        // Purchase successful. We can terminate
                        System.out.println(targetBookTitle+"
successfully purchased from Agent "+reply.getSender().getName());
                        System.out.println("Price = "+bestPrice);
                              myAgent.doDelete();
                              }
                              else {
      System.out.println("Attempt  failed:  requested  book  already
sold.");
                              }
                              step = 4;
                        }
                        else {
                              block();
                        }
                        break; }}
```

5.3 Kết quả bài toán

Ứng dụng được bắt đầu bằng lệnh runjade để truy nhập vào giao diện Nền tảng JADE. Ở Main-contrainer ta khởi tạo một tác tử Seller tên là "BookSeller", giao diện BookSellerGui hiện ra, sách nhập vào tên là "LapTrinhJava" giá "50000", kết quả như sau:

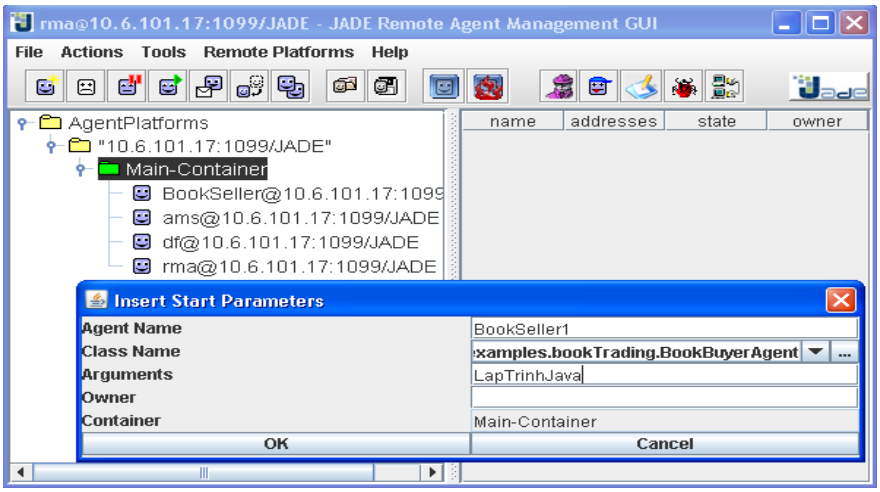


Hình 5.1: Khởi tạo tác tử "BookSeller"

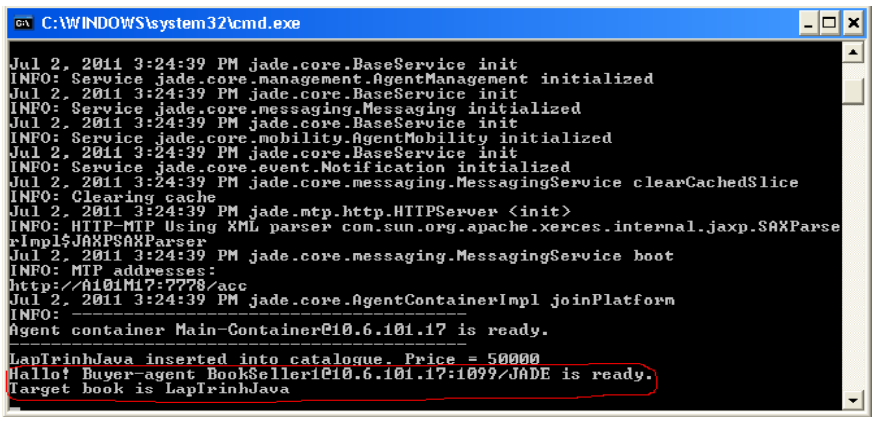


Hình 5.2: Kết quả chạy tác tử "BookSeller"

Vẫn ở Main-container, tạo tác tử Buyer tên là "BookBuyer", sách cần mua là "LapTrinhJava", kết quả như sau:

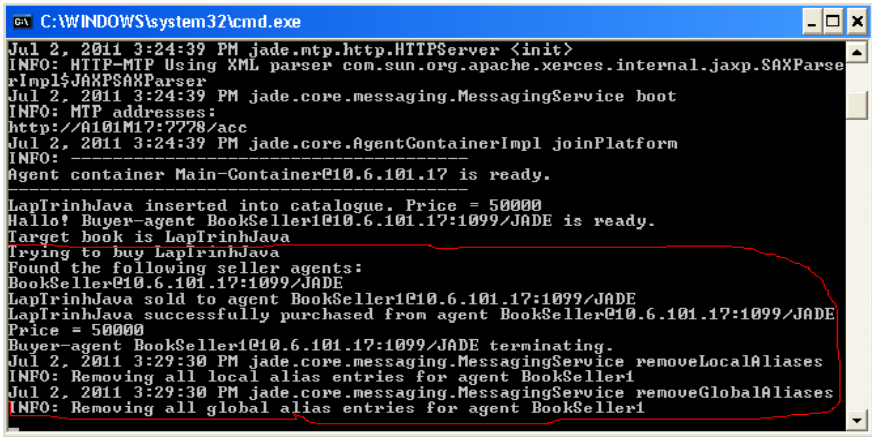


Hình 5.3: Khởi tạo tác tử "BookBuyer"



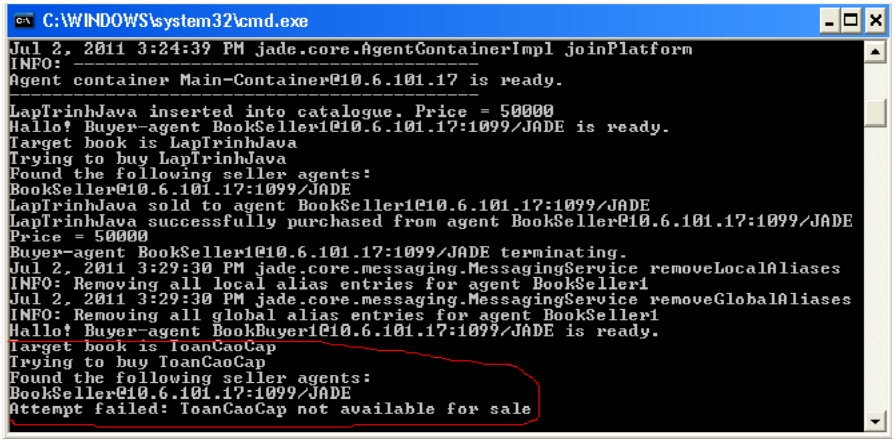
Hình 5.4: Kết quả chạy tác tử "BookBuyer"

Sau khi đưa ra tên sách cần mua là "LapTrinhJava", tác tử "BookBuyer" sẽ tìm tác tử Seller trong Nền tảng, sau đó yêu cầu mua và thực hiện giao dịch:



Hình 5.5: Kết quả giao dịch 2 tác tử Seller và Buyer

Nếu tác tử Buyer mua một mặt hàng sách không có trong Catalogue, nó vẫn tìm và hỏi các tác tử Seller, sau khi không tìm thấy sách, tác tử Seller sẽ thông báo không tìm thấy và gửi cho tác tử Buyer:



Hình 5.6: Kết quả giao dịch không thành công

KẾT LUẬN VÀ PHƯƠNG HƯỚNG TIẾP THEO

Đề tài đã hoàn thành được các mục tiêu đề ra:

- Đề tài đã định nghĩa và đưa ra được ưu – nhược điểm của công nghệ tác tử và tác tử di động. Nghiên cứu về hệ đa tác tử, ưu – nhược điểm so với hệ xử lý phân tán cũ và các lĩnh vực ứng dụng áp dụng công nghệ này.
- Đề tài cũng đưa ra một số giao thức tương tác giữa các tác tử, sự thương lượng giữa các tác tử, nghiên cứu giao thức truyền thông giữa các tác tử.
- Đề tài cũng đưa ra và giới thiệu một số môi trường, nền tảng đề phát triển tác tử, từ đó đi sâu vào nghiên cứu nền tảng JADE.
- Đề tài nghiên cứu một ứng dụng minh họa dựa trên công nghệ tác tử di động và nền tảng JADE, thao tác trên nền tảng JADE.

Phương hướng tiếp theo:

Trên cơ sở bài toán ứng dụng, hướng nghiên cứu tiếp theo là xây dựng và cài đặt các mô hình tương tác và thương lượng phức tạp giữa các tác tử, cụ thể là tác tử "BookSeller" và tác tử "BookBuyer".

Nghiên cứu sâu hơn về các kiến thức nâng cao của nền tảng JADE và thao tác trên JADE.

Em còn rất nhiều ý tưởng để phát triển đề tài có ứng dụng sử dụng công nghệ tác tử trong thực tế, nhưng do kiến thức, khả năng và thời gian còn hạn chế nên trong khuôn khổ đồ án này em chưa thể thực hiện được. Trong tương lai em sẽ tiếp tục nghiên cứu và phát triển đề tài này.

TÀI LIỆU THAM KHẢO:

[1]. Gerhard Weiss, The MIT Press, Cambridge, Massachusetts, London, England - *MultiAgent Systems, A Modern Approach to Distributed Modern Approach to Artificial Intelligence*

[2]. John Wiley Sons, 2007 Gate,Chichester,West Sussex PO19 8SQ, England - *Developing Multi-Agent Systems with JADE*

[3]. Lin Padgham & Michael Winikoff, RMIT University, Melbourne, Australia - *Developing Intelligent Agent Systems*

[4]. MichaelWooldridge - *Intelligent Agents*

[5]. Ning Zhong Maebashi Institute of Technology Japan, Jiming Liu Hong Kong Baptist University, Setsuo Ohsuga Waseda University Japan, Jeffrey Bradshaw University of West Florida USA - *Intelligent Agent Technolog Research and Development*

[6]. Rafael H. Bordini · Mehdi Dastani ·J¨ urgen Dix · Amal El Fallah Seghrouchni - *Multi-Agent Programming*

[7]. W. Branner - *Foundations and Application, Springer, 1998*

DANH MỤC WEBSITE THAM KHẢO

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

[2]. http://www.javaworld.com