Tạp chí Tin học và Điều khiển học, T. 17, S. 4 (2001), 37–44

CÔNG NGHỆ TÁC TỬ DI ĐỘNG VÀ ỨNG DỤNG THỬ NGHIỆM TRONG THƯƠNG MẠI ĐIỆN TỬ

VƯƠNG QUANG KHẢI, NGUYỄN THÚC HẢI

Abstract. This article discusses about the basic concepts, advantages and applications of mobile agent technology. It's also introduce Aglet, an IBM's framework for develop mobile agent applications and demonstrate how to build an e-commerce solution with Aglet.

Tóm tắt. Bài báo giới thiệu các khái niệm cơ bản, ưu điểm và khả năng ứng dụng của công nghệ *tác tử di động* (Mobile Agent), từ đó dẫn đến các kết quả thiết kế và cài đặt thử một hệ thống thương mại điện tử (được đặt tên là *Cybermart*) dựa trên mô hình tác tử di động *Aglet* của hãng IBM.

1. MỞ ĐẦU

Mạng máy tính đang ngày càng phát triển với nhịp độ chóng mặt và theo nhiều hướng khác nhau. Kích thước của mạng tăng nhanh, không chỉ Internet mà cả những mạng nội bộ (Intranet) của các tổ chức đều phát triển mạnh mẽ, nhờ chi phí phần cứng ngày càng rẻ đi rất nhiều và nhu cầu cần có những kênh truyền tin đồng nhất, mở, hiệu quả trong nội bộ cũng như giữa những tổ chức đó. Một hiệu ứng của sự phát triển nói trên là sự gia tăng mật độ lưu thông trên mạng, điều này lại thúc đẩy những cố gắng để nâng cao hiệu năng của hệ thống cơ sở hạ tầng truyền thông. Khả năng kết nối các máy tính ngày càng được nâng cao, và kỹ thuật phát triển dẫn đến việc tăng năng lực tính toán ở các nút mạng trung tâm cũng như những nút đầu cuối. Cùng lúc đó, sự xuất hiện của những ứng dụng phân tán cho tất cả mọi người như *E-mail, Web*... đã thúc đẩy sự quan tâm của người dùng và của thị trường, thúc đẩy việc phát triển những loại ứng dụng mới cho mạng máy tính. Chúng thay đổi cách thức mà mạng nói chung và Internet nói riêng được khai thác ngày nay. Vai trò của mạng chuyển từ một phương tiện truyền thông đơn thuần thành một hệ thống phân tán rộng hỗ trợ những dịch vụ hoàn toàn mới mẻ.

Tuy nhiên, số lượng ứng dụng có thể khai thác triệt để được những thế mạnh của mạng máy tính hiện nay còn ít. Đó là do thiếu những cơ chế cho phép lập trình viên ứng dụng được những khả năng tiềm tàng của hệ thống, hay nói cách khác, các mô hình tính toán hiện nay vẫn chưa theo kịp sự phát triển của cơ sở hạ tầng truyền thông. Có khá nhiều vấn đề cần phải giải quyết. Sự tăng trưởng nhanh chóng của kích thước mạng gây nên nhiều khó khăn cho vấn đề mở rộng chương trình. Nhiều mô hình và công nghệ hoạt động rất hiệu quả với những mạng nhỏ lại rất khó có thể mở rộng để áp dụng ở những mạng toàn cầu như Internet. Những máy nối mạng sử dụng kết nối không dây thậm chí còn vướng phải nhiều vấn đề phức tạp hơn nữa: các nút mạng có thể di chuyển và kết nối / ngắt kết nối liên tục, dẫn đến kết cấu của mạng không còn là tĩnh. Hậu quả là một số lý thuyết và công nghệ về mạng máy tính hoàn toàn không còn sử dụng được nữa, hoặc yêu cầu phải sửa đổi rất nhiều để thích ứng với tình hình mới. Mặt khác, sự phát triển về kích thước và hiệu năng mạng yêu cầu các hệ thống phân tán phải tăng tính mềm dẻo và có khả năng dễ dàng mở rộng để cung cấp những dịch vụ hoàn toàn mới cho người dùng.

Hiện nay, đang có nhiều nỗ lực tìm kiếm những mô hình lập trình hiệu quả cho môi trường mạng mới. Hầu hết các cách tiếp cận này đều cố gắng sửa đổi những mô hình và công nghệ chuẩn cũ để thích ứng với môi trường mới, và thường cải tiến từ kiến trúc *client/server* truyền thống. Tuy nhiên, đó chỉ là những giải pháp nửa vời vì chúng chỉ giải quyết được những vấn đề về kết nối và khả năng mở rộng và không cung cấp những cơ chế và phương tiện cho phép lập trình viên xây

dựng chương trình theo những mô hình mới, mềm dẻo, dễ tùy biến và dễ mở rộng để có thể tận dụng được toàn bộ khả năng tiềm tàng của cơ sở hạ tầng mạng hiện nay.

Một cách tiếp cận đang được chú ý là *tác tử di động* (*Mobile Agent* - MA), một trong những mở rộng của mô hình *mã di động* (*Mobile Code* - MC). Trong mô hình này, MA là những *đối tượng* (*mã lệnh + dữ liệu*) thông minh có thể di chuyển tự do trong mạng máy tính để thực hiện nhiệm vụ được giao thay cho người dùng. Cách tiếp cận mềm dẻo này mở ra những lĩnh vực rất hấp dẫn để phát triển. Tuy nhiên, dù đã có sự quan tâm rộng rãi về lý thuyết cũng như ứng dụng của MA, lĩnh vực này còn rất non trẻ, vẫn đang thiếu những phương pháp luận rõ ràng và những mô hình phát triển ứng dụng hoàn thiện.

2. MÔ HÌNH MA

2.1. Các ưu điểm của mô hình MA

So sánh với các mô hình tính toán phân tán truyền thống, *mô hình MA* thể hiện một số ưu điểm cơ bản sau đây [2, 3, 4, 6, 9].

- *Giảm tải cho mạng:* Các hệ thống phân tán thường sử dụng những giao thức truyền thông định nghĩa trước để tiến hành tương tác giữa các máy nhằm thực hiện nhiệm vụ được giao. Phương pháp này tốn rất nhiều băng thông mạng. MA cho phép chúng ta gói những yêu cầu và gửi chúng tới máy xa, nơi việc tương tác có thể diễn ra một cách cục bộ. MA cũng giúp cho việc giảm lượng dữ liệu thô phải truyền qua mạng: khi một lượng lớn dữ liệu được lưu trữ ở xa, những dữ liệu đó nên được xử lý cục bộ rồi truyền kết quả về hơn truyền tất cả dữ liệu thô về mạng để xử lý tại chỗ.

- *Trừu tượng hóa các giao thức:* Khi dữ liệu được chuyển trong các hệ thống phân tán, mỗi máy đều phải chứa mã lệnh cài đặt giao thức để gửi dữ liệu đi và xử lý dữ liệu tới. Tuy nhiên, khi các giao thức được cải tiến để nâng cao hiệu suất hoặc tăng độ an toàn, việc nâng cấp mã của giao thức trên tất cả các máy trong toàn hệ thống rất khó khăn và đôi lúc là không thể thực hiện được. Kết quả là giao thức trở thành một gánh nặng phải thừa kế rất khó chịu. Trong khi đó, với mô hình MA, chúng ta có thể gửi mã lệnh tới máy ở xa để tạo ra một kênh truyền tin theo đúng giao thức cần thiết một cách linh hoạt.

- *Trung lập với môi trường:* Môi trường mạng thường rất khác nhau, cả về khía cạnh phần cứng và phần mềm. Vì MA thường không phụ thuộc vào tầng phần cứng và tầng truyền tin, mà chỉ phụ thuộc vào các tiến trình ứng dụng nên chúng cho phép tích hợp hệ thống một cách chặt chẽ.

- *Khắc phục độ trễ của mạng:* Các hệ thống quan trọng như *người máy* trong các qui trình sản xuất cần phải phản ứng ngay lập tức với những thay đổi của môi trường. Điều khiển những hệ thống như vậy trong một mạng lớn của một nhà máy sẽ tồn tại một độ trễ đáng kể. Độ trễ này là không thể chấp nhận được đối với các hệ thống yêu cầu thời gian thực. MA sẽ rất có ích ở đây vì chúng có thể được gửi từ hệ thống điều khiển trung tâm đến người máy và được thực hiện một cách cục bộ để điều khiển *người máy* này.

- *Hoạt động tự động và không đồng bộ:* Các thiết bị di động thường phải sử dụng kết nối mạng chậm và đắt tiền. Những công việc yêu cầu kết nối liên tục giữa một thiết bị di động với một mạng cố định hoàn toàn không kinh tế và rất khó khăn về mặt kỹ thuật. Những công việc đó có thể được cài đặt nhúng vào các MA rồi gửi qua mạng. Khi tới nơi, MA có thể hoạt động tự động, độc lập với tiến trình đã tạo ra nó. Thiết bị di động có thể kết nối lại sau này để nhận MA về và lấy kết quả.

- *Chắc chắn, linh hoạt và có khả năng chịu lỗi cao:* Khả năng phản ứng linh hoạt với những thay đổi của môi trường giúp cho việc xây dựng những hệ phân tán chắc chắn và có tính chịu lỗi cao được dễ dàng hơn. Chẳng hạn như khi một máy chuẩn bị tắt, mọi MA thực hiện trên máy đó sẽ được báo động để tiến hành di trú rồi tiếp tục công việc của chúng tại một máy khác trong mạng.

2.2. Các ứng dụng của MA

Có thể xem xét việc sử dụng mô hình MA để phát triển một ứng dụng nếu nó có một trong

những tính chất sau [6, 7, 8, 10].

- *Làm việc trong một môi trường mạng với độ tin cậy thấp và không kết nối thường xuyên:* trong những môi trường như vậy, MA có thể di chuyển tới đích và tính toán cục bộ ở đó, hơn là phải cố gắng kết nối liên tục qua đường mạng chất lượng kém, chi phí cao.
- *Cần phản ứng thời gian thực:* Hệ thống cần phản ứng ngay lập tức đối với những thay đổi của môi trường bên ngoài.
- *Tính toán nhiều chặng, nhiều tiến trình:* Các vấn đề tính toán số học có thể được chia thành những phần rời rạc, mỗi phần được cài đặt vào một *tác tử* (*agent*). Sau khi hoàn thành việc tính toán, các *agent* sẽ trở về nhà với kết quả thu được, nơi hệ thống sẽ tổng hợp lại thành kết quả cuối cùng.
- *Phải cộng tác với những thực thể không được tin cậy:* Các MA liên lạc với nhau thông qua những giao diện được định nghĩa rõ ràng từ trước và được thiết kế nhằm chống lại việc tấn công của các MA khác. Các MA cũng khó có thể gây nên mối đe dọa nào cho tiến trình ứng dụng chứa nó vì chúng bị giới hạn bởi những cơ chế quản lý tài nguyên chặt chẽ.

Cần lưu ý rằng đa số các ứng dụng tiềm tàng của MA vẫn có thể được xây dựng dựa trên những mô hình lập trình truyền thống, tuy nhiên sử dụng MA trong những trường hợp đó có thể sẽ hiệu quả hơn hoặc có chi phí thấp hơn.

Chỉ có một loại ứng dụng duy nhất bắt buộc phải ứng dụng công nghệ MC hoặc MA: đó là việc tiến hành *điều khiển từ xa theo thời gian thực*, chẳng hạn việc *điều khiển các người máy vận hành trên sao Hỏa.* Khi đó, do độ trễ của việc truyền tín hiệu quá lớn, việc phản ứng lại những thay đổi của môi trường sử dụng công nghệ điều khiển truyền thống sẽ có xác suất thành công thấp hơn nhiều so với việc gửi các đoạn mã điều khiển lên để hệ thống có thể thực hiện chúng một cách cục bộ, phản hồi tức thời khi có bất cứ thay đổi nào xảy ra.

3. MÔ HÌNH AGLET

3.1. Giới thiệu

Hiện tại có khá nhiều công trình nghiên cứu nhằm xây dựng các mô hình MA khác nhau. Có thể kể ra một số sản phẩm đáng chú ý như:

- *Aglet* của hãng IBM
- *Aglet Tcl* của Đại học Darthmout
- *Agent for Remote Access* (ARA) của Đại học Kaiserslautern
- *Concordia* của Phòng thí nghiệm Horizon Systems, Công ty Mitsubishi
- *Mole* của Viện nghiên cứu các Hệ thống Phân tán và Song song, Hoa Kỳ
- *Odyssey* của hãng General Magic
- *TACOMA* của Đại học Cornell
- *Voyager* của hãng Object Space
- *Secure and High Performance Mobile Agent Infrastructure* (SHIP-MAI) của Phòng thí nghiệm Multimedia và Mobile Agent, Đại học Ottawa

Trong các mô hình MA nói trên, Aglet của hãng IBM được sử dụng rộng rãi nhất, đặc biệt đối với các ứng dụng thương mại điện tử, vì những lý do cơ cản sau đây:

- *Aglet* sử dụng ngôn ngữ lập trình Java - một ngôn ngữ hiện đại, phổ biến, cho khả năng di động và tính khả chuyển cao.
- Mô hình đối tượng để lập trình *Aglet* được thiết kế rất đầy đủ, chặt chẽ và dễ sử dụng. Chương trình viết bằng *Aglet* dễ đọc và có tính trong sáng cao.
- *Aglet* là sản phẩm của một hãng lớn (IBM), được sử hỗ trợ vững chắc của hãng này (hãng tiên phong trong lĩnh vực MA thương mại - General Magic - đã từ bỏ những nỗ lực của mình và chuyển sang nghiên cứu những công nghệ xử lý âm thanh).

- *Aglet* là phần mềm mã nguồn mở, có thể tự do sửa đổi hoặc thêm những gì cần thiết.

3.2. Mô hình đối tượng Aglet

Aglet được xây dựng gần giống như mô hình *Applet* trong Java. Nó là một tập hợp các đối tượng trừu tượng cho phép lập trình viên có thể khai báo đè (override) những phương thức (method) đã định nghĩa trước để thêm các chức năng mới cho chương trình. Mô hình đối tượng *Aglet* gồm một số đối tượng quan trọng sau [5]:

- *Aglet:* Đây là một đối tượng Java di động có thể di chuyển tới những máy khác trong mạng. Nó chạy trong luồng thực hiện riêng mỗi khi tới một máy và có thể phản ứng lại những thông điệp được gửi tới.
- *Proxy:* Đây là thực thể đại diện cho các *Aglet*. Nó có vai trò như một lá chắn để bảo vệ *Aglet* khỏi bị trục xuất trực tiếp đến những phương thức công cộng. Proxy cũng cung cấp khả năng trong suốt với vị trí cho các *Aglet*: nó giúp *Aglet* không cần quan tâm tới vị trí thực của các *Aglet* khác.
- *Context:* Đây là nơi hoạt động của các *Aglet*. Nó là một đối tượng tĩnh cung cấp những phương thức để điều hành và quản lý các *Aglet* đang chạy trong môi trường thực hiện đồng nhất. Một nút trong mạng máy tính có thể chạy nhiều server và một server có thể chứa nhiều *context*. *Context* được đặt tên và có thể xác định bằng cách kết hợp địa chỉ của server và tên của nó.
- *Message:* Đây là một đối tượng dùng để trao đổi giữa các *Aglet*. Nó cho phép truyền những thông điệp đồng bộ cũng như dị bộ. Các *Aglet* có thể trao đổi các message để cộng tác và truyền thông tin.
- *Future reply:* Đây là một đối tượng được sử dụng trong quá trình truyền thông điệp không đồng bộ như một thẻ bài (handler) để sau này nhận lại kết quả.
- *Identifier:* Mỗi *identifier* được gắn với một *Aglet*. Nó là định danh duy nhất và không thể thay đổi được trong suốt quá trình sống của *Aglet*.

Mặc dù chúng ta có thể phát triển các ứng dụng *Aglet* với bộ công cụ JDK chuẩn của hãng Sun, việc này gặp khá nhiều khó khăn và rõ ràng là không hiệu quả. Đó là vì JDK chỉ cung cấp công cụ để biên dịch mã nguồn Java mà không cung cấp trình soạn thảo đi kèm. Lập trình viên phải sử dụng một trình soạn thảo văn bản riêng biệt để soạn thảo mã nguồn Java, rồi gọi trình dịch của JDK để biên dịch thành file thực thi. Và hệ thống gỡ rối, tìm lỗi trong JDK cũng hết sức nguyên thủy, chỉ cung cấp những chức năng cơ bản nhất với giao diện khó dùng. Do đó, một lập trình viên bắt đầu nghiên cứu phát triển các ứng dụng *Aglet* sẽ gặp nhiều khó khăn thi thực hiện công việc với JDK.

Hãng Borland đã xây dựng một môi trường tích hợp phát triển ứng dụng Java rất mạnh có tên *JBuilder* [1]. Môi trường lập trình này có đầy đủ những chức năng cơ bản như soạn thảo, thiết kế biên dịch, tìm lỗi các ứng dụng Java... Ngoài ra *JBuilder* cũng hỗ trợ những tính năng cao cấp như *code-complete* (tự động sinh mã khi chỉ cần gõ phần đầu của mã lệnh), *code-browsing* (tự động hiển thị cấu trúc các đối tượng khi đang soạn thảo), *trình thiết kế giao diện trực quan, các đồ thuật* (wizard) để sinh mã tự động, cung cấp khả năng đặt các điểm kiểm tra có điều kiện (conditional break point) để gỡ rối, có chức năng tìm lỗi trên máy ở xa... Nếu sử dụng *JBuilder* để phát triển các ứng dụng *Aglet*, chúng ta có thể tận dụng tất cả các khả năng ưu việt của môi trường này, giúp cho việc lập trình được dễ dàng và đạt hiệu quả cao hơn. Tuy nhiên, bản thân *JBuilder* không hỗ trợ sẵn cho việc phát triển các ứng dụng *Aglet*. Tác giả đã xây dựng một *phân hệ mở rộng* để có thể sinh mã tự động và tiến hành thực thi, tìm lỗi các ứng dụng *Aglet* bên trong *JBuilder*. Tuy nhiên, do khuôn khổ bài báo có hạn, tác giả không thể nêu chi tiết về phân hệ này, độc giả quan tâm có thể lấy thêm thông tin tại địa chỉ Web: *http://codecentral.borland.com/codecentral/ccweb.exe/listing?id=15730.*

4. PHÁT TRIỂN ỨNG DỤNG THƯƠNG MẠI ĐIỆN TỬ DỰA TRÊN MA

4.1. Đặt vấn đề

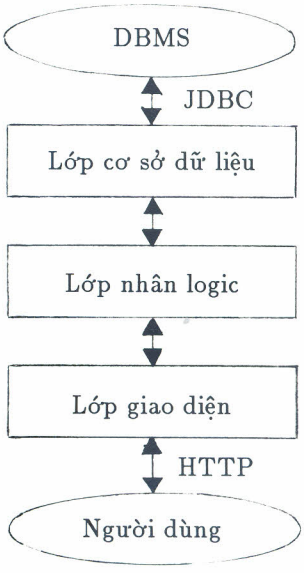
Trong vòng 5 năm trở lại đây, mạng toàn cầu Internet đã dần dần thay đổi, chuyển mình từ một

mạng dành riêng cho giới nghiên cứu, giáo dục và các chuyên gia máy tính trở thành mạng thương mại cho tất cả mọi người. Thị trường thương mại điện tử toàn cầu phát triển rất mạnh và được dự báo sẽ đạt doanh số 7 nghìn tỉ USD vào năm 2005.

Tuy nhiên, các hệ thống thương mại điện tử hiện nay *mới chỉ rút ngắn không gian* cho người sử dụng (khách hàng chỉ cần ngồi ở nhà, lên mạng, nhấn chuột để lựa chọn và mua hàng) mà *chưa tiết kiệm cho họ về mặt thời gian.* Người sử dụng vẫn phải vào từng *Web site* thương mại điện tử, tìm mặt hàng mình cần, rồi tiến hành so sánh giá cả giữa các *site* để tìm *site* bán rẻ nhất. Việc tìm kiếm và so sánh này rất nhàm chán, và đôi khi cũng mất thời giờ không kém kiểu mua bán truyền thống, vì trong hoàn cảnh của Internet, người sử dụng có quá nhiều lựa chọn và họ không có đủ thời gian cũng như sự kiên nhẫn để duyệt qua tất cả các những lựa chọn đó.

Để thử nghiệm ứng dụng công nghệ MA trong thực tế, chúng tôi đã tiến hành xây dựng thử nghiệm một hệ thống thương mại điện tử thông minh. Hệ thống này, được đặt tên là **Cybermart**, cho phép người sử dụng dùng các MA *"cố vấn"* để thay mặt họ tiến hành mua, bán loại hàng hóa bất kỳ qua mạng. Với cách tiếp cận này, người sử dụng sẽ rút ngắn được cả không gian và thời gian: họ chỉ cần nêu ra yêu cầu, các cố vấn điện tử sẽ tự động tìm kiếm đối tác và thương thuyết với nhau để chọn được giao dịch hợp lý nhất. Người sử dụng sẽ nhận được thông báo kết quả khi giao dịch hoàn tất.

Do khuôn khổ bài báo, ở đây chúng tôi chỉ giới thiệu những ý tưởng chủ đạo trong thiết kế hệ thống *Cybermart*, không trình bày thiết kế chi tiết của các lớp.



Hình 1. Mô hình phân lớp hệ thống **Cybermart**

4.2. Thiết kế hệ thống

Hệ thống Cybermart được thiết kế theo *mô hình 3 lớp* (xem Hình 3):

- *lớp cơ sở dữ liệu* có nhiệm vụ lưu trữ và truy xuất thông tin dựa trên một *Hệ quản trị cơ sở dữ liệu* (DBMS) cụ thể,
- *lớp nhân logic* chứa logic chương trình, có nhiệm vụ xử lý thông tin,
- *lớp giao diện* có nhiệm vụ trao đổi thông tin với người sử dụng.

Ba lớp này có thể được thiết kế và phát triển hoàn toàn độc lập, chỉ cần tuân theo giao diện đã được thống nhất từ trước để ghép lại sau này. Việc phân lớp cho phép tách rời các công việc để có thể phát triển cũng như bảo trì, nâng cấp hệ thống dễ dàng, hiệu quả hơn. Khi phát triển, có thể phân công lao động một cách rõ ràng, hợp lý: chia công việc xây dựng lớp cơ sở dữ liệu cho quản trị viên cơ sở dữ liệu, việc xây dựng lớp xử lý logic giao cho lập trình viên, việc xây dựng giao diện giao cho các họa sĩ chuyên nghiệp. Sau này có thể thay đổi, nâng cấp một trong các lớp mà không lo ảnh hưởng đến các lớp còn lại.

Hệ thống *Cybermart* kết nối với cơ sở dữ liệu theo tiêu chuẩn JDBC (Java Data Base Connectivity), nên trên nguyên tắc có thể sử dụng với hầu hết các hệ quản trị cơ sở dữ liệu phổ biến như *Oracle, SQL Server, mySQL...* Hệ thống xây dựng một lớp đối tượng riêng (*Porter*) để thực hiện các yêu cầu về cơ sở dữ liệu, vì thế thậm chí có thể thay đổi cả cấu trúc các bảng, các trường của cơ sở dữ liệu mà không sợ ảnh hưởng đến các lớp khác.

Để có thể mô tả một cách mềm dẻo mọi loại hàng hóa, các *cố vấn* trong hệ thống **Cybermart** sử dụng ngôn ngữ định dạng XML để mô tả các tính chất của hàng hóa (Hình 2). Việc sử dụng định dạng XML cũng có ưu điểm là có thể sử dụng các *trình phân giải* (*Parser*) XML có sẵn trên thị trường để phân tích dữ liệu, và có thể dễ dàng mở rộng hệ thống để tiến hành giao dịch với những hệ thống thương mại điện tử khác sau này.

```
<!ELEMENT Commodity (Name, Category, Producer,
                     Style *, Quantity, Price)>
   <!ELEMENT   Name      (#PCDATA)>
   <!ELEMENT   Category  (#PCDATA)>
   <!ELEMENT   Producer  (#PCDATA)>
   <!ELEMENT   Style     (#PCDATA)>
   <!ATTLIST   Style
            Name  CDATA                              #REQUIRED
            Type  "=" | "!" | ">" | "<" | ">=" | "<=" )  "="
   >
   <!ELEMENT   Quantity  (#PCDATA)>
   <!ELEMENT   Price     (#PCDATA)>
```

Ví dụ:

```
<Commodity>
   <Name>Viva</Name>
   <Category>Automobile</Category>
   <Producer>Suzuki</Producer>
   <Style Name="Color">Blue</Style>
   <Style Name="Model" Type=">=">1996</Style>
   <Quantity>1</Quantity>
   <Price>1400</Price>
</Commodity>
```

Hình 2. Document Type Definition (định nghĩa kiểu tài liệu) của dữ liệu XML về hàng hóa trao đổi trong hệ thống

Để đạt được độ linh hoạt tối đa, *lớp xử lý logic* được thiết kế mở với tính năng nạp thêm các *cố vấn* mua bán mới. Nhờ đó hệ thống có thể được phát triển, nâng cấp một cách dễ dàng với những *cố vấn* ngày càng thông minh hơn. Phần lõi hệ thống chỉ có hai lớp đối tượng là *Receptionist* (có nhiệm vụ hiển thị chào mừng và tiến hành kiểm tra mật khẩu) và *Manager* (phục vụ quản trị hệ thống, dùng để quản lý danh sách người sử dụng, danh sách các máy chủ **Cybermart** và danh sách các *cố vấn*). Người quản trị hệ thống có thể nạp thêm một số lượng tùy ý các *cố vấn* mới và khách hàng có thể lựa chọn *cố vấn* nào thích hợp nhất đối với họ để sử dụng. Các *cố vấn* đều được thừa kế từ một lớp trừu tượng là *Consultant*, lớp này khai báo những phương thức tối thiểu mà một *cố vấn* phải hỗ trợ.

Để phục vụ mục đích minh họa, tác giả cài đặt hai *cố vấn mẫu* là *BuyConsultant*, phục vụ việc mua hàng và *SellConsultant*, phục vụ việc bán hàng. *Cố vấn SellConsultant* thu nhận yêu cầu của người dùng rồi sử dụng một *Aglet* thuộc lớp đối tượng *Seller* để đi chào hàng. *Cố vấn BuyConsultant* cũng thu nhận yêu cầu của người dùng rồi sử dụng một *Aglet* thuộc lớp đối tượng *Buyer* để đi thu thập giá cả. *Buyer* sẽ di chuyển trong các máy chủ **Cybermart** kinh doanh chủng loại hàng hóa cần thiết để tìm đối tác bán hàng. Khi *Buyer* báo cáo tìm được một *Seller* tiềm năng, *BuyConsultant* sẽ thương thuyết với *Seller* đó để thực hiện giao dịch rồi báo cáo về cho người mua. Khi thực hiện xong giao dịch, *Seller* cũng sẽ thông báo về cho *SellConsultant* để báo cho người bán.

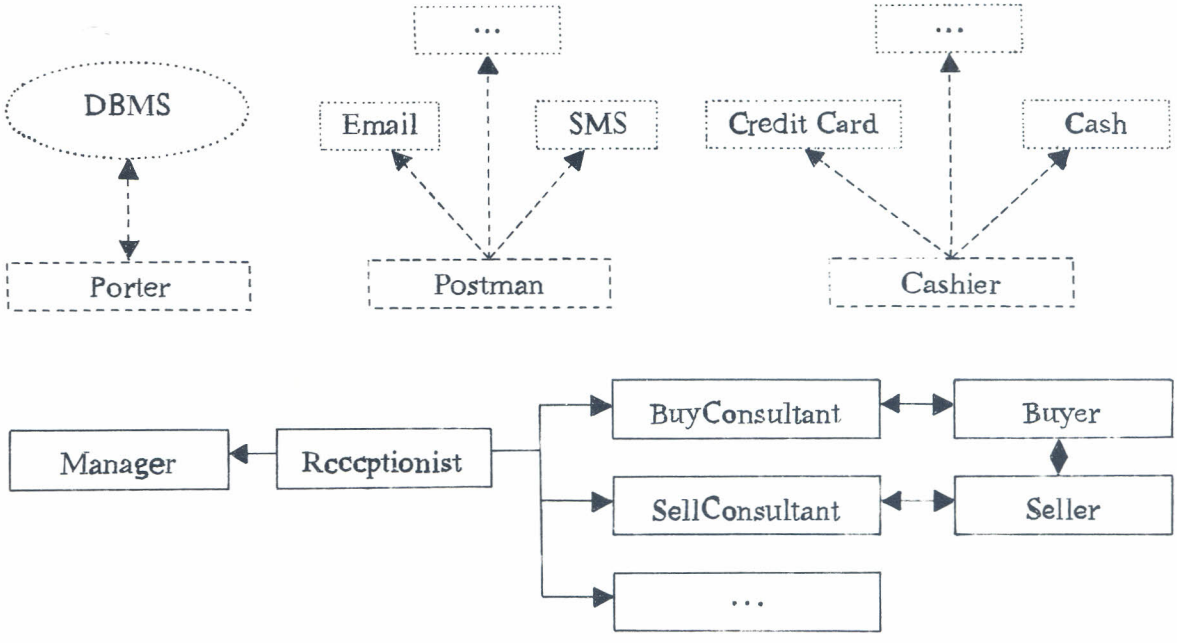
Một trong những yêu cầu quan trọng nhất đối với mọi hệ thống thương mại điện tử là khả năng hỗ trợ việc thanh toán. Hệ thống xây dựng một lớp đối tượng riêng (*Cashier*) để phục vụ cho mục đích này. Đối tượng *Cashier* không tiến hành trực tiếp việc thanh toán mà lại sử dụng giao thức HTTP kết nối đến những dịch vụ *Web* trên Internet để thực hiện việc đó. Cách tiếp cận này có ưu điểm là cho phép tách việc cài đặt chức năng thanh toán ra khỏi hệ thống, sử dụng những hệ thống thanh toán hoàn chỉnh đã được thiết kế sẵn bởi những nhà cung cấp dịch vụ khác trên Internet. Đồng thời nó cũng cho phép mở rộng hệ thống một cách linh hoạt với những hình thức thanh toán

mới sẽ xuất hiện sau này (một số công ty muốn phát triển thương mại điện tử trong điều kiện Việt Nam hiện nay đang dự định sử dụng những loại *thẻ trả tiền trước* đặc biệt để tiến hành thanh toán qua mạng, vì người dân vẫn chưa có thói quen gửi tiền ở ngân hàng và dùng thẻ tín dụng).

Hầu hết các ứng dụng *Aglet* trong các nghiên cứu trước đây trên thế giới đều yêu cầu người sử dụng phải cài đặt *Tahiti* (trình điều khiển giao diện đồ họa của các *Aglet*) cùng mã lệnh của *Aglet* trên máy cục bộ. Hay nói cách khác máy cục bộ sẽ phải đóng vai trò một máy trạm thành phần trong hệ thống MA. Người sử dụng sẽ tương tác trực tiếp với *Aglet* qua *Tahiti*. Cách tiếp cận này có nhược điểm là bất tiện cho người dùng (yêu cầu cài đặt MA khá phức tạp) và yêu cầu máy cục bộ phải nối mạng liên tục (một điều kiện bất khả thi ở Việt Nam hiện nay). Vì thế chúng tôi đã chọn cách tiếp cận cài *Tahiti* và mã lệnh *Aglet* trên các máy chủ ở xa, người sử dụng có thể kết nối đến và điều khiển các *cố vấn* qua giao diện *Web*. Cách tiếp cận này yêu cầu những kỹ năng lập trình cao cấp hơn, nhưng giải quyết được cả hai nhược điểm của cách trước.

Để tách rời lớp giao diện khỏi lớp nhân logic, nội dung các *trang Web* trao đổi thông tin với người dùng được lưu riêng biệt trong các file HTML trên máy chủ. Bên trong file HTML được nhúng một số thẻ đặc biệt để đánh dấu những đoạn cần phải chèn nội dung động khi chạy chương trình. Khi chương trình cần trả một *trang Web* về cho người sử dụng, nó sẽ đọc file HTML tương ứng trên đĩa cứng, tìm những thẻ nội dung động để điền thông tin mới, rồi trả *trang Web* về. Cách tiếp cận này, lấy ý tưởng từ mô hình *WebClass* của Microsoft, cho phép các họa sĩ thiết kế có thể thay đổi nội dung cũng như cách trình bày giao diện mà không phải dịch lại mã nguồn chương trình.

Để người dùng có thể biết được những thông tin mới nhất về giao dịch mà không cần kết nối thường xuyên đến máy chủ **Cybermart**, hệ thống cần phải cài đặt một cơ chế báo tin cho người dùng. Tiện nhất là người sử dụng có thể nhận thông báo qua *E-mail*, hoặc qua nhắn tin ngắn bằng điện thoại di động... Ở đây hệ thống sử sụng cách tiếp cận tương tự như cơ chế hỗ trợ thanh toán trực tuyến; xây dựng một lớp đối tượng riêng (*Postman*) tiến hành kết nối đến những dịch vụ *Web* trên Internet để thực hiện việc báo tin. Cách tiếp cận này tách chức năng truyền tin ra khỏi hệ thống, cho phép dễ dàng sử dụng những phương tiện truyền thông khác sau này, chẳng hạn như *fax* hoặc tổng hợp tiếng nói qua điện thoại... Mô hình đầy đủ của *Cybermart* được minh họa trong hình 3.



Hình 3. Mô hình đối tượng đầy đủ của hệ thống **Cybermart**

Hệ thống **Cybermart** đã được thí nghiệm với 3 phiên bản máy chủ A, B, C phục vụ cho hoạt động kinh doanh máy tính và đã cho các kết quả ban đầu khả quan.

5. KẾT LUẬN

Hệ thống thương mại điện tử **Cybermart** đã thực sự giúp người dùng tiết kiệm thời gian, công sức khi mua bán hàng trên mạng với những *"cố vấn điện tử"* có khả năng chọn hàng thương thuyết giá cả thay mặt cho họ. Trong thời gian thử nghiệm ban đầu, chúng tôi mới chỉ cài đặt các *agent cố vấn* mua bán cơ bản với độ thông minh hạn chế. Nhờ thiết kế mở và khả năng thêm tùy ý các *agent cố vấn*, có thể tiếp tục phát triển hệ thống **Cybermart** này với những *cố vấn* thông minh hơn rất nhiều, có khả năng tiến hành mua bán, thương thuyết khôn khéo như con người. Việc phát triển tiếp hầu như không có giới hạn và chỉ phụ thuộc vào khả năng sáng tạo của lập trình viên.

Trong bài báo này chúng tôi đã giới thiệu một trong những kết quả khởi đầu đã được thực hiện ở Khoa Công nghệ thông tin, Trường Đại học Bách khoa Hà Nội nhằm đưa một công nghệ mới đầy hiệu quả vào quá trình xây dựng và phát triển các ứng dụng phân tán, đặc biệt là các ứng dụng trong thương mại điện tử. Hệ thống ứng dụng được xây dựng mới ở dạng thử nghiệm (prototype) nhưng đã đủ để rút ra những kết luận bổ ích cho những bước đi tiếp theo nhằm hoàn thiện hệ thống theo hướng thông minh hóa ở mức độ cao cho các *tác tử di động* để có thể đưa vào khai thác thực sự trong hoạt động thương mại điện tử ở nước ta.

Mặc dù vẫn còn đang trong giai đoạn phát triển, mô hình MA đã tỏ ra rất nhiều hứa hẹn. Đây thực sự là bước tiến hóa tự nhiên, hợp lý của mô hình lập trình *client/server* truyền thống. Chi nhánh IBM ở Nhật đã tiến hành nghiên cứu và thử nghiệm thành công ứng dụng MA trong lĩnh vực tài chính ngân hàng. Viễn cảnh về một tương lai không xa, khi hàng triệu *agent* di chuyển trên mạng, tiến hành tương tác để phục vụ người dùng quả là rất hấp dẫn và chắc chắn sẽ trở thành hiện thực.

TÀI LIỆU THAM KHẢO

[1] Borland, *"JBuider OpenTools API"*, www.borland.com/techpubs/jbuilder/.

[2] Colin G. Harrison, David M. Chess, *"Mobile Agents, are they a good idea?"*, www.research.ibm.com/massdist/mobag.ps, 1995.

[3] David Kotz, Robert S. Gray, Mobile Agents and the future of the Internet, *Operating Systems Review*, 8-1999.

[4] Frederich Knabe, *"An Overview of Mobile Agent Programming"*, www.cs.virginia.edu/~knabe/lomaps96abs.html, 1996.

[5] IBM, *"Aglet SDK"*, www.trl.ibm.com/Aglets/documentation.html.

[6] Lưu Vĩnh Toàn, Nguyễn Thúc Hải, Phát triển ứng dụng phân tán bằng công nghệ tác tử di động, *Tạp chí Bưu chính Viễn thông* (2000).

[7] Michael S. Greeberg, Jennifer C. Byington, Thophany Holding, David G. Harper, Mobile Agents and security, *IEEE Communications Magazine* **7** (1998).

[8] Neeran M. Karnik, Anand R. Triphahi, Design issues in Mobile Agent programming system, *IEEE Concurrency* **6** (1998).

[9] Vu Anh Pham, Ahmed Karmouch, Mobile software Agent: an overview, *IEEE Communication Magazine* **36** (Issue 7) (1998).

[10] Yariv Aridor, Mitsuru Oshima, *Infrastructure for Mobile Agents: Requirements and Design*, 2nd, International Workshop on Mobile Agents, 1998.

Nhận bài ngày 30 - 5 - 2001

Khoa Công nghệ thông tin, Trường Đại học Bách khoa Hà Nội.