BỘ GIÁO DỤC VÀ ĐÀO TẠO

ĐẠI HỌC HUẾ

**TRƯỜNG ĐẠI HỌC KHOA HỌC**

HỒ NGUYỄN THÀNH NHÂN

TÌM HIỂU VỀ KIẾN TRÚC HƯỚNG DỊCH VỤ TRONG LĨNH VỰC CÔNG NGHỆ PHẦN MỀM VÀ ỨNG DỤNG

CHUYÊN NGÀNH:

MÃ SỐ:

KHOA HỌC MÁY TÍNH

60 48 01 01

LUẬN VĂN THẠC SĨ KHOA HỌC   
CÔNG NGHỆ THÔNG TIN

NGƯỜI HƯỚNG DẪN KHOA HỌC

PGS. TS. HOÀNG HỮU HẠNH

Huế, 2016

**LỜI CAM ĐOAN**

Với mục đích học tập, nghiên cứu để nâng cao kiến thức và trình độ chuyên môn, tôi làm luận văn này một cách nghiêm túc và trung thực.

Trong luận văn, tôi có sử dụng một số tài liệu tham khảo của một số tác giả. Danh sách các tài liệu tham khảo được liệt kê tại mục “Tài liệu tham khảo”.

Tôi xin cam đoan và chịu trách nhiệm về sự trung thực trong luận văn tốt nghiệp Thạc sĩ của mình.

Học viên

HỒ NGUYỄN THÀNH NHÂN

**LỜI CẢM ƠN**

Tôi xin cảm ơn quý Thầy giáo, Cô giáo khoa Công nghệ thông tin, Trường Đại học Khoa học, Đại học Huế, quý Thầy giáo tham gia giảng dạy lớp Cao học Khoa học máy tính niên khóa 2014-2016 đã truyền đạt nhiều kiến thức, bài giảng quý báu cho tôi trong thời gian học tại Trường.

Tôi xin cảm ơn Thầy giáo – PGS. TS. Hoàng Hữu Hạnh đã tận tình hướng dẫn và chỉ bảo tôi rất nhiều trong suốt thời gian làm luận văn.

Mặc dù đã cố gắng hoàn thành luận văn trong phạm vi và khả năng cho phép nhưng chắc chắn sẽ không tránh khỏi những thiếu sót, kính mong nhận được sự góp ý và tận tình chỉ bảo của quý Thầy Cô và các bạn.

Một lần nữa, xin chân thành cám ơn và mong luôn nhận được những tình cảm chân thành của tất cả mọi người.

Huế, tháng 4 năm 2016

Hồ Nguyễn Thành Nhân

**MỤC LỤC**

Lời cam đoan

Lời cảm ơn

Mục lục

Danh mục các thuật ngữ viết tắt

Danh mục các hình vẽ

[MỞ ĐẦU 1](#_Toc447614021)

[Chương 1. Tổng quan về kiến trúc hướng dịch vụ 4](#_Toc447614022)

[1.1. Công nghệ Web Services 4](#_Toc447614023)

[1.1.1. Tổng quan về Web Services 4](#_Toc447614024)

[1.1.2. Kiến trúc của Web Services 6](#_Toc447614025)

[1.1.3. Các thành phần của Web Services 7](#_Toc447614026)

[1.2. Kiến trúc hướng dịch vụ 19](#_Toc447614027)

[1.2.1. Kiến trúc hướng dịch vụ (SOA) là gì? 19](#_Toc447614028)

[1.2.2. Các tính chất của một hệ thống SOA 23](#_Toc447614029)

[1.2.3. Kiến trúc phân tầng chi tiết của SOA 27](#_Toc447614030)

[1.3. Ngôn ngữ thi hành quy trình nghiệp vụ - BPEL 29](#_Toc447614031)

[1.3.1. Giới thiệu 29](#_Toc447614032)

[1.3.2. Các khái niệm cơ bản 30](#_Toc447614033)

[1.4. Tiểu kết chương 1 33](#_Toc447614034)

[Chương 2. Khung ứng dụng hỗ trợ lập trình SOA 34](#_Toc447614035)

[2.1. Nền tảng Eclipse 34](#_Toc447614038)

[2.1.1. Giới thiệu 34](#_Toc447614039)

[2.1.2. Các thành phần và kiến trúc 34](#_Toc447614040)

[2.2. Kiến trúc plug-in của Eclipse 37](#_Toc447614041)

[2.2.1. Mô hình plug-in Eclipse **Error! Bookmark not defined.**](#_Toc447614042)

[2.2.2. Tập tin cấu hình (Manifest) của Plug-in 43](#_Toc447614043)

[2.2.3. Vòng đời của plug-in 43](#_Toc447614044)

[2.3. Tiểu kết chương 2 43](#_Toc447614045)

[Chương 3. Xây dựng ứng dụng trên nền tảng Eclipse 45](#_Toc447614046)

[3.1. Bài toán điều phối các lời gọi dịch vụ trong kiến trúc SOA 45](#_Toc447614048)

[3.1.1. Mục tiêu 45](#_Toc447614049)

[3.1.2. Giải pháp 45](#_Toc447614050)

[3.2. Mô tả chi tiết 46](#_Toc447614051)

[3.2.1. Kiến trúc hướng dịch vụ theo đường ống 46](#_Toc447614052)

[3.2.2. Services Bus 47](#_Toc447614053)

[3.2.3. Plug-n-Play Web Services 47](#_Toc447614054)

[3.2.4. Tính trong suốt của lời gọi dịch vụ 49](#_Toc447614055)

[3.2.5. Dịch vụ đường ống – Services Pipeline 50](#_Toc447614056)

[3.2.6. Tính năng kỹ thuật và các loại kịch bản của Pipeline 52](#_Toc447614057)

[3.3. tiểu kết chương 3 56](#_Toc447614058)

[KẾT LUẬN VÀ KIẾN NGHỊ 58](#_Toc447614059)

[TÀI LIỆU THAM KHẢO 59](#_Toc447614060)

[PHỤ LỤC 61](#_Toc447614061)

**DANH MỤC CÁC THUẬT NGỮ VIẾT TẮT**

|  |  |
| --- | --- |
| CSDL | Cơ sở dữ liệu |
| GUI | Graphical User Interface |
| IDE | Intergrated Development Environment |
| SEI | Service endpoint interface |
| SOA | Service Oriented Architecture |
| SOAP | Simple Object Access Protocol |
| UDDI | Universal Description, Discovery and Integration |
| WSDL | Web Services Description Language |
| XML | Extensible Markup Language |

**DANH MỤC CÁC HÌNH VẼ**

# MỞ ĐẦU

Ngày nay, công nghệ thông tin (CNTT) đang càng ngày càng phát triển, trở thành ngành mũi nhọn trong chiến lược phát triển kinh tế của mỗi quốc gia. Với sự phát triển của Internet và với xu thế hội nhập chung của toàn thế giới, các tổ chức, doanh nghiệp bắt tay, phối hợp hoạt động và chia sẻ tài nguyên với nhau để nâng cao hiệu quả hoạt động. Khi đó các sản phẩm sẽ có độ phức tạp lớn hơn, kéo theo các vấn đề liên quan như chi phí sản xuất, chi phí quản lý và bảo trì. Bên cạnh đó, ngành công nghệ phần mềm còn phải đối mặt với các khó khăn trong xu thế mới như vấn đề tái sử dụng và mở rộng các hệ thống sẵn có, vấn đề về sự không tương thích giữa các hệ thống khác nhau của nhiều tổ chức...

Để giải quyết vấn đề trên, nhiều giải pháp đã được nghiên cứu và ứng dụng, nhưng hầu hết đều không giải quyết các khó khăn một cách triệt để và kết quả đạt được cũng không như mong đợi. Một giải pháp đang được cộng đồng CNTT rất quan tâm, đó là “Kiến trúc hướng dịch vụ” (Service Oriented Architecture – SOA). Giải pháp này đã được ứng dụng trong thực tiễn và đã mang lại những kết quả có tính đột phá. Bây giờ mọi người đã có thể tin rằng SOA có thể giải quyết tốt những thách thức đã nêu ở trên, và sẽ là một xu thế trong tương lai.

Nhận thấy giải pháp SOA là một giải pháp mang xu thế thời đại, có nhiều triển vọng và được sự đồng ý của giảng viên hướng dẫn, PGS.TS. Hoàng Hữu Hạnh, tôi chọn hướng nghiên cứu của luận văn là **“Tìm hiểu về kiến trúc hướng dịch vụ trong lĩnh vực công nghệ phần mềm và ứng dụng”**

Mục đích chính của luận văn là tìm một giải pháp để hỗ trợ cho việc lập trình một ứng dụng dựa trên SOA. Để đạt được mục đích của luận văn thì ta thấy đối tượng nghiên cứu là về kiến trúc hướng dịch vụ, các công nghệ nền tảng, các khung ứng dụng, môi trường phát triển và thực thi để hỗ trợ lập trình theo kiến trúc hướng dịch vụ. Trên cơ sở nghiên cứu đó, phạm vi của luận văn là nghiên cứu để tạo ra một kiến trúc dựa trên một nền tảng có sẵn để hỗ trợ lập trình.

Luận văn sử dụng hai phương pháp nghiên cứu: nghiên cứu tài liệu và thực nghiệm.

Tham khảo sách, các bài báo đã đươc xuất bản trên các tờ báo, tạp chí uy tín trong nước và nước ngoài, các luận văn hay công trình nghiên cứu đã được công bố, các website uy tín và các nguồn khác.

Tổng hợp các kết quả nghiên cứu đã có để lựa chọn cách tiếp cận phù hợp nhất cũng như nhận thức về SOA một cách đúng đắn.

Nghiên cứu các Framework hỗ trợ lập trình SOA trên nhiều Platform khác nhau.

Dựa trên thực tế để đề ra giải pháp và xây dựng ứng dụng với SOA.

Trong luận văn này, tôi đề xuất một phương pháp để hỗ trợ việc lập trình theo kiến trúc hướng dịch vụ đó là xây dựng một Services Bus Plug-in dựa trên công nghệ Eclipse. Kết quả của luận văn hoàn toàn có thể được áp dụng vào việc lập trình cho kiến trúc hướng dịch vụ.

**CẤU TRÚC CỦA LUẬN VĂN**

**Chương 1: Tổng quan về kiến trúc hướng dịch vụ**

Chương này trình bày các vấn đề cơ bản về kiến trúc hướng dịch vụ và các công nghệ liên quan. Đồng thời cũng nêu ra những nét chính, cái nhìn tổng quan về kiến trúc hướng dịch vụ.

**Chương 2: Khung ứng dụng hỗ trợ lập trình SOA**

Đây là chương trình bày về kiến trúc nền tảng Eclipse, các cơ chế và kiến trúc plug-in của Eclipse hỗ trợ cho việc lập trình theo kiến trúc hướng dịch vụ.

**Chương 3: Xây dựng ứng dụng trên nền tảng Eclipse**

Kiến trúc hướng dịch vụ lấy dịch vụ (service) làm trung tâm, làm thành phần cơ bản và cốt lõi. Kiến trúc này hỗ trợ service - giao tiếp với nhau, có thể tái sử dụng và kết hợp với nhau để tạo thành các quy trình nghiệp vụ phục vụ cho lợi ích của người sử dụng. Vậy để xây dựng nên một ứng dụng theo kiến trúc hướng dịch vụ thì ta cần và có thể sử dụng nền tảng hay khung ứng dụng nào phù hợp? Ngôn ngữ lập trình nào? Chương này sẽ giới thiệu một giải pháp để hỗ trợ xây dựng ứng dụng theo hướng kiến trúc hướng dịch vụ.

# Chương 1. Tổng quan về kiến trúc hướng dịch vụ

## Công nghệ Web Services

### Tổng quan về Web Services

Web Services là một hệ thống phần mềm được thiết kế để hỗ trợ khả năng tương tác giữa các ứng dụng trên các máy tính khác nhau thông qua mạng Internet, giao diện chung và sự gắn kết của nó được mô tả bằng XML. Web Services là tài nguyên phần mềm có thể xác định bằng địa chỉ URL, thực hiện các chức năng và đưa ra các thông tin người dùng yêu cầu. Một Web Services được tạo nên bằng cách lấy các chức năng và đóng gói chúng sao cho các ứng dụng khác dễ dàng nhìn thấy và có thể truy cập đến những dịch vụ mà nó thực hiện, đồng thời có thể yêu cầu thông tin từ Web Services khác. Nó bao gồm các mô đun độc lập và bản thân nó được thực thi trên Server.

Web Services cho phép các ứng dụng khác nhau từ các nguồn khác nhau có thể giao tiếp với các ứng dụng khác mà không đòi hỏi nhiều thời gian coding, do tất cả các quá trình giao tiếp đều tuân theo định dạng XML, cho nên Web Services không bị phụ thuộc vào bất kì hệ điều hành hay ngôn ngữ lập trình nào. Web Services cho phép client và server có thể tương tác được với nhau trên các nền tảng khác nhau mà không cần bất cứ thay đổi hay yêu cầu đặc biệt nào. Ví dụ, chương trình viết bằng ngôn ngữ Java cũng có thể trao đổi dữ liệu với các chương trình viết bằng Perl, các ứng dụng chạy trên nền Windows cũng có thể trao đổi dữ liệu với các ứng dụng chạy trên nền Linux. Công nghệ Web Services không yêu cầu phải sử dụng trình duyệt và ngôn ngữ HTML.

Web Services cung cấp các chuẩn mở tập trung vào giao tiếp, việc cộng tác giữa con người và các ứng dụng đã tạo nên một môi trường nơi mà Web Services đang trở thành nền tảng cho việc tích hợp ứng dụng. Các ứng dụng được xây dựng sử dụng các Web Services từ nhiều nguồn khác nhau làm việc cùng với nhau bất kể là chúng ở đâu hoặc chúng đã được triển khai như thế nào. Có thể có các định nghĩa khác nhau về Web Services khi các công ty xây dựng chúng, nhưng hầu hết tất cả các định nghĩa đều có chung các điểm sau:

* Thứ nhất, Web Services đưa ra chức năng hữu dụng cho người sử dụng Web thông qua một giao thức chuẩn Web. Trong hầu hết các trường hợp, giao thức được sử dụng đó là SOAP.
* Thứ hai, Web Services mô tả các giao diện của chúng một cách chi tiết nhằm cho phép người sử dụng xây dựng một ứng dụng máy trạm để giao tiếp được với chúng. Mô tả này thường được cung cấp ở dạng một tài liệu XML gọi là một tài liệu về ngôn ngữ mô tả Web Services – WSDL (Web Services Description Language).
* Thứ ba, Web Services được đăng ký sao cho các khách hàng tiềm năng là người sử dụng có thể tìm thấy chúng một cách dễ dàng. Điều này được thực hiện với UDDI (Universal Discovery Description and Integration).

Xuất bản

Gởi/nhận thông điệp

Tìm kiếm

**Đăng ký dịch vụ (UDDI)**

**Người sử dụng dịch vụ**

**Nhà cung cấp dịch vụ**

Mô tả dịch vụ (WSDL)

Hình 1.1. Cơ chế hoạt động của Web Services

Web Services như một dịch vụ phần mềm được trình bày trên web thông qua giao thức SOAP, được mô tả bằng một tệp WSDL và được đăng ký trong UDDI. Các dịch vụ Web Services là nguồn thông tin mà ta có thể dễ dàng kết hợp vào các ứng dụng. Dễ dàng nhận ra toàn bộ lớp ứng dụng có thể được xây dựng để phân tích và tích hợp thông tin ta quan tâm và trình bày nó theo nhiều cách khác nhau.

Việc trình bày các ứng dụng đang có như các dịch vụ Web Services cho phép người sử dụng xây dựng các ứng dụng có các tính năng mạnh hơn thông qua việc sử dụng Web Services như những block được xây dựng sẵn. Ví dụ, người sử dụng có thể phát triển một ứng dụng mua bán để tự động lấy các thông tin về giá cả từ nhiều nhà cung cấp khác nhau, cho phép người dùng chọn một nhà cung cấp, chuyển đơn hàng và sau đó theo dõi việc chuyển hàng cho tới khi nhận được hàng. Ứng dụng của nhà cung cấp, khi trình bày các dịch vụ của họ trên Web, có thể quay ra sử dụng các dịch vụ Web Services để kiểm tra tín dụng của khách hàng, lấy tiền từ tài khoản của khách hàng và thiết lập việc chuyển hàng với một công ty vận tải.

### Kiến trúc của Web Services

Kiến trúc của Web Services bao gồm các tầng như sau:



Hình 1.2. Kiến trúc của Web Services

Tầng vận chuyển (Transport) với những công nghệ chuẩn là HTTP, SMTP và JMS. Có nhiệm vụ truyền thông điệp giữa các ứng dụng mạng.

Tầng giao thức tương tác dịch vụ (Service Communication Protocol) với công nghệ chuẩn là SOAP. SOAP là giao thức nằm giữa tầng vận chuyển và tầng mô tả thông tin về dịch vụ, SOAP cho phép người dùng triệu gọi một dịch vụ từ xa thông qua một message XML.

Tầng mô tả dịch vụ (Service Description) với công nghệ chuẩn là WSDL và XML. WSDL là một ngôn ngữ mô tả giao tiếp và thực thi dựa trên XML. Các Web Services sử dụng ngôn ngữ WSDL để truyền các tham số và các loại dữ liệu cho các thao tác, các chức năng mà các Web Services cung cấp.

Tầng dịch vụ (Service): cung cấp các chức năng của Services.

Tầng đăng ký dịch vụ (Service Registry) với công nghệ chuẩn là UDDI. UDDI dùng cho cả người dùng và SOAP server, nó cho phép đăng ký Services để người dùng có thể gọi thực hiện Services từ xa qua mạng, hay nói cách khác một Services cần phải được đăng ký để cho phép các khách hàng có thể gọi thực hiện.

Bên cạnh đó để cho các Services có tính an toàn, toàn vẹn và bảo mật thông tin trong kiến trúc Web Services chúng ta có thêm các tầng Policy, Security, Transaction, Management giúp tăng cường tính bảo mật, an toàn và toàn vẹn thông tin khi sử dụng Services.

### Các thành phần của Web Services

#### XML

XML được viết tắt của cụm từ Extensible Markup Language – Ngôn ngữ đánh dấu dữ liệu. Là một chuẩn mở do W3C đưa ra cho cách thức mô tả dữ liệu, nó cho phép các máy tính truyền dữ liệu giữa các hệ thống không đồng nhất.

Về hình thức XML có cấu trúc giống với HTML nhưng không tuân theo một đặc tả quy ước như HTML. HTML định nghĩa các thành phần được hiển thị như thế nào, còn XML lại định nghĩa các thành phần chứa cái gì.

Web Services là sự kết hợp của nhiều thành phần khác nhau nên nó sử dụng các tính năng và đặc trưng của các thành phần đó để giao tiếp, XML là công cụ chính để giải quyết vấn đề này và là kiến trúc nền tảng cho việc xây dựng một Web Services.

#### WSDL – Web Services Description Language

WSDL định nghĩa một tài liệu XML mô tả giao diện của các Web Services. Tài liệu WSDL này được sử dụng cho bên yêu cầu dịch vụ (services requester). Bên yêu cầu dịch vụ sẽ sử dụng thông tin về giao diện định nghĩa trong lược đồ WSDL để triệu gọi (invoke) Web Services.

Một tài liệu WSDL mô tả một Web Services như một tập các đối tượng trừu tượng gọi là các “ports” và “endpoint”. WSDL cũng định nghĩa bên trong nó các phương thức của Web Services. Các phương thức tương ứng với “operation” và dữ liệu trao đổi tương ứng với “message”. Một tập các phương thức liên quan được nhóm lại vào trong một “portType”. Một ràng buộc kết nối (binding) chỉ định một giao thức mạng và đặc tả định dạng dữ liệu cho một *portType* cụ thể. Kế đến một port được định nghĩa bằng cách kết hợp một địa chỉ mạng với một binding. Nếu một client có được một tài liệu WSDL và tìm thấy binding và địa chỉ cho mỗi port, nó có thể gọi các phương thức của dịch vụ theo đúng giao thức và định dạng dữ liệu đã đặc tả.

Phần tử gốc của tất cả các tài liệu WSDL luôn là phần tử <definitions>. Nó chứa bên trong sáu thành phần chia thành hai nhóm:

Thông tin trừu tượng (type, messages, portType)

Thông tin cụ thể (bindings, services)

WSDL định nghĩa cách mô tả Web Services theo cú pháp tổng quát của XML, bao gồm các thông tin:

Tên dịch vụ

Giao thức và kiểu mã hóa sẽ được sử dụng khi gọi các hàm của Web Services

Loại thông tin: thao tác, tham số, những kiểu dữ liệu (có thể là giao diện của Web Services cộng với tên cho giao diện này)

Cấu trúc của một WSDL :



Hình 1.3. Cấu trúc WSDL

Một WSDL hợp lệ gồm hai phần:

Service Inteface mô tả giao diện và giao thức kết nối

Service Implementation mô tả thông tin để truy xuất service



Hình 1.4. Cấu trúc WSDL

##### Các thành phần của WSDL

|  |  |  |
| --- | --- | --- |
| **Thành phần** | **Thông tin** | **Mô tả** |
| Service Interface | <type> | Định nghĩa các kiểu dữ liệu của thông điệp gửi |
| <message> | Mô tả thông điệp được gửi giữa client và server |
| <porttype> | WSDL mô tả cách gửi và nhận thông điệp |
| <binding> | Định nghĩa cách các Web Services kết hợp với nhau |
| Service Implementation | <service> | Nó sẽ thực hiện những gì đã được định nghĩa trong tập tin giao diện và cách gọi Web Services theo thủ tục và phương thức nào |
| <Port> | là một cổng đầu cuối, nó định nghĩa như một tập hợp của binding và một địa chỉ mạng |

Giải thích ý nghĩa các thành phần:

Type: định nghĩa kiểu dữ liệu được sử dụng cho Web Services để đảm bảo tính không phụ thuộc vào platform hoặc các phần tử XML được sử dụng cho các trao đổi thông báo, WSDL sử dụng cấu trúc của lược đồ XML để định nghĩa kiểu dữ liệu.

<wsdl:definitions .... >

< wsdl:types>

<xs:schema .... />\*

</ wsdl:types>

</ wsdl:definitions>

Message : định nghĩa các thành phần dữ liệu và các thông điệp mà nó được gọi tới. Mỗi thông điệp có thể bao gồm một hoặc nhiều phần, các thành phần này có thể so sánh với các câu lệnh của các lời gọi hàm trong các ngôn ngữ lập trình truyền thống. Những định nghĩa message đư­ợc sử dụng bởi phần tử thi hành dịch vụ. Nhiều thao tác có thể tham chiếu tới cùng định nghĩa message. Thao tác và những message đ­ược mô hình riêng rẽ để hỗ trợ tính linh hoạt và đơn giản hóa việc tái sử dụng lại. Chẳng hạn, hai thao tác với cùng tham số có thể chia sẻ một định nghĩa message.

< wsdl:definitions .... >

< wsdl:message name="nmtoken"> \*

< wsdl:part name="nmtoken" element="qname" /> \*

</ wsdl:message>

</ wsdl:definitions>

PortType : đây là thành phần quan trọng nhất trong một tài liệu WSDL. Nó được sử dụng để mô tả Web Services, các thao tác được thực thi và các lời gọi thông điệp. Thành phần PortType có thể được so sánh với các thư viện hàm (hoặc các module, các lớp ) trong các ngôn ngữ lập trình.

<wsdl:definitions .... >

<wsdl:portType name="nmtoken">

<wsdl:operation name="nmtoken" .... /> \*

</wsdl:portType>

</wsdl:definitions>

Trong thành phần < wsdl:porttype>, ta thường gặp 4 kiểu thao tác được WSDL định nghĩa dưới đây:

|  |  |
| --- | --- |
| **Kiểu thao tác** | **Mô tả** |
| One - way | Thao tác này thể hiện rằng nó chỉ nhận các lời gọi thông điệp nhưng không trả lại thông điệp đáp ứng |
| Request - response | Thao tác này bao gồm việc nhận các thông điệp yêu cầu và trả về các thông điệp đáp ứng |
| Solicit - response | Thao tác này sẽ gửi đi các yêu cầu và đợi các đáp ứng |
| Notification | Thao tác này sẽ gửi đi các yêu cầu nhưng không đợi để nhận các đáp ứng |



Hình 1.5. Bốn kiểu thao tác mà một cổng có thể hỗ trợ

Mỗi kiểu thao tác có cú pháp biến đổi tùy theo: thứ tự của các message nhập, xuất và lỗi.

Ví dụ :

<wsdl:definitions .... >

<wsdl:portType .... > \*

<wsdl:operation name="nmtoken" parameterOrder= "nmtokens">

<wsdl:input name="nmtoken"? message="qname"/>

<wsdl:output name="nmtoken"? message="qname"/>

<wsdl:fault name="nmtoken" message="qname"/>\*

</wsdl:operation>

</wsdl:portType >

</wsdl:definitions>

*Binding:* Định nghĩa cách thức truy cập Web Services thông qua các giao thức bên dưới. Mỗi phần tử Binding sẽ mô tả cách thức liên kết một PortType vào một Protocol nhất định. Web Services hỗ trợ bao nhiêu Protocol thì phải xây dựng bấy nhiêu phần tử Binding

<wsdl:binding name="…" type="ns:…">

<soap:binding transport="… " style="document" />

<wsdl:operation name="…">

<soap:operation soapAction="urn:…" style="document" />

<wsdl:input>

<soap:body use="literal" />

</wsdl:input>

<wsdl:output>

<soap:body use="literal" />

</wsdl:output>

</wsdl:operation>

</wsdl:binding>

Service (dịch vụ) : Nó sẽ thực hiện những gì đã được định nghĩa trong tập tin giao diện và cách gọi Web Services theo thủ tục và phương thức nào:

<wsdl:definitions .... >

<wsdl:service name="nmtoken"> \*

<wsdl:port .... />\*

</wsdl:service>

</wsdl:definitions>

Port (cổng dịch vụ) : Là một cổng đầu cuối, nó định nghĩa như­ một tập hợp của binding và một địa chỉ mạng.

<wsdl:definitions .... >

<wsdl:service .... > \*

<wsdl:port name="nmtoken" binding="qname"> \*

</wsdl:port>

</wsdl:service>

</wsdl:definitions>

Ở đây chúng ta thấy rằng thuộc tính kết hợp tên là qname. Nó tham chiếu tới một mối kết hợp. Một cổng chứa đựng chính xác một địa chỉ mạng. Bất kỳ cổng nào trong phần thi hành phải tương ứng chính xác với một tham chiếu trong phần giao diện.

#### UDDI – Universal Description, Discovery, and Integration

Về cơ bản Universal Description, Discovery, and Intergration (UDDI) là một nơi mà các tổ chức đăng ký và tìm kiếm các Web Services. Nó đóng vai trò như service broker cho phép người sử dụng dịch vụ tìm đúng nhà cung cấp dịch vụ cần tìm. UDDI hỗ trợ chức năng:

Thực hiện tìm kiếm, định vị những doanh nghiệp cung cấp dịch vụ hay sản phẩm theo vùng địa lý

Thông tin về một nhà cung cấp dịch vụ bao gồm địa chỉ, thông tin liên lạc và các định danh

Thông tin kỹ thuật (Technical information) về Web Services mà doanh nghiệp cung cấp (ví dụ như cách sử dụng dịch vụ được cung cấp)

Để sử dụng đến các dịch vụ của UDDI, bản thân UDDI cung cấp một tập hàm API dưới dạng SOAP Web Services. Tập API được chia làm hai phần: Inquiry API dùng truy vấn và Publisher’s API dùng đăng ký. Phần API dùng để truy vấn bao gồm hai phần con: một phần dùng để tạo ra các chương trình cho phép tìm kiếm và duyệt thông tin trên một UDDI registry, phần còn lại dùng để xử lý lỗi triệu gọi.

UDDI gồm 2 thành phần chính:

Phần đăng ký của tất cả các Web Services’s metadata, bao gồm cả việc trỏ đến tài liệu WSDL mô tả dịch vụ

Phần thiết lập WSDL Port type định nghĩa cho các thao tác và tìm kiếm thông tin đăng ký.

UDDI xây dựng dựa trên các giao thức chuẩn Internet được công bố bởi W3C và IETF như XML, HTTP, và DNS. UDDI sử dụng WSDL để mô tả giao diện của Web Services. Thêm nữa tính năng độc lập với nền tảng ngôn ngữ lập trình đã được điều hợp cùng với giao thức SOAP.

#### SOAP – Simple Object Access Protocol

SOAP là một giao thức dựa trên XML để trao đổi thông tin giữa các máy tính. SOAP có thể sử dụng một loạt các thông điệp hệ thống và có thể được gửi qua giao thức của tầng vận chuyển. Tập trung ban đầu là các thủ tục triệu gọi từ xa được vận chuyển thông qua HTTP. Do đó SOAP cho phép các ứng dụng của khách hàng kết nối dễ dàng đến các dịch vụ từ xa và gọi từ xa các phương pháp này.

Hay như định nghĩa của tổ chức W3C thì “SOAP là một giao thức hổ trợ việc trao đổi thông tin trong một môi trường tản quyền và phân tán”.

Khái niệm cơ bản nhất của mô hình SOAP là việc sử dụng các tài liệu XML như những thông điệp trao đổi. Điều này có nhiều ưu điểm hơn các giao thức truyền dữ liệu khác. Các thông điệp XML có thể được tổng hợp và đọc với một bộ soạn thảo text đơn giản, ta có thể làm việc với XML trên hầu hết mọi nền tảng.

##### Đặc trưng

SOAP có những đặc trư­ng sau:

SOAP đư­ợc thiết kế đơn giản và dễ mở rộng

Tất cả các message SOAP đều đư­ợc mã hóa sử dụng XML

SOAP sử dụng giao thức truyền dữ liệu riêng

Không có garbage collection phân tán, và cũng không có cơ chế tham chiếu. Vì thế SOAP client không giữ bất kỳ một tham chiếu đầy đủ nào về các đối tượng ở xa

SOAP không bị ràng buộc bởi bất kỳ ngôn ngữ lập trình nào hoặc công nghệ nào

Vì những đặc trư­ng này, nó không quan tâm đến công nghệ gì đư­ợc sử dụng để thực hiện miễn là người dùng sử dụng các message theo định dạng XML. T­ương tự, dịch vụ có thể đư­ợc thực hiện trong bất kỳ ngôn ngữ nào, miễn là nó có thể xử lý được những message theo định dạng XML.

##### Cấu trúc một message theo dạng SOAP

Cấu trúc một message theo dạng SOAP được mô tả như hình dưới đây



Hình 1.7. Cấu trúc message SOAP

Message theo dạng SOAP là một văn bản XML bình thường bao gồm các phần tử sau:

*Phần tử gốc - envelop:* phần từ bao trùm nội dung message, khai báo văn bản XML như là một thông điệp SOAP.

Ví dụ:

<SOAP-ENV:Envelope

xmlns:SOAP-ENV=”<http://schemas.xmlsoap.org/soap/envelope/>”>

Với SOAP 1.1 namespace URI là <http://schemas.xmlsoap.org/soap/envelope/>, đối với SOAP 1.2 namespace URI là <http://www.w3.org/2001/09/soap-envelope>.

*Phần tử đầu trang – header:* chứa các thông tin tiêu đề cho trang, phần tử này không bắt buộc khai báo trong văn bản. Những đầu mục còn có thể mang những dữ liệu chứng thực, những chữ ký số hóa, và thông tin mã hóa, hoặc những cài đặt cho giao tác.

Ví dụ:

<SOAP-ENV:Header>

<ns1:PaymentAccount

xmlns:ns1="urn:ecerami"SOAP-ENV:

mustUnderstand="true">orsenigo473

</ns1:PaymentAccount >

</SOAP-ENV:Header>

*Phần tử khai báo nội dung chính trong thông điệp – body:*chứa các thông tin yêu cầu và phản hồi. Thành phần khai báo nội dung là thành phần bắt buộc đối với tất cả các message dạng SOAP.

Ví dụ:

<env:Body>

<m:GetLastTradePrice Env:encodingStyle="http://www.w3.org/2001/09/soap-encoding"

xmlns:m="http://example.org/2001/06/quotes">

<symbol>DIS</symbol>

</m:GetLastTradePrice>

</env:Body>

*Phần tử phát sinh lỗi – Fault:* cung cấp thông tin lỗi xảy ra trong quá trình xử lý thông điệp.

Ví dụ:

<?xml version='1.0' encoding='UTF-8'?>

<SOAP-ENV:Envelope

xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"

xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"

xmlns:xsd="http://www.w3.org/1999/XMLSchema">

<SOAP-ENV:Body>

<SOAP-ENV:Fault>

<faultcode xsi:type="xsd:string">

SOAP- ENV:Client</faultcode>

<faultstring xsi:type="xsd:string">

Failed to locate method (ValidateCreditCard) in class (examplesCreditCard) at /usr/local/ActivePerl-5.6/lib/ site\_perl/5.6.0/SOAP/Lite.pm line 1555.

</faultstring>

</SOAP-ENV:Fault>

</SOAP-ENV:Body>

</SOAP-ENV:Envelope>

##### Mô hình dữ liệu

Mục đích của mô hình dữ liệu SOAP là cung cấp một sự trừu tượng hóa độc lập ngôn ngữ cho kiểu ngôn ngữ lập trình chung. Nó gồm có:

Những kiểu XSD đơn giản như những kiểu dữ liệu cơ bản trong đa số các ngôn ngữ lập trình như­ int, string, date, …

Những kiểu phức tạp, có hai loại là struct và array.

Tất cả các phần tử và những định danh có trong mô hình dữ liệu SOAP thì được định nghĩa bằng namespace SOAP-ENC.

## Kiến trúc hướng dịch vụ

### Kiến trúc hướng dịch vụ (SOA) là gì?

Kiến trúc hướng dịch vụ - SOA (Service Oriented Architecture) là một cách tiếp cận hay một phương pháp luận để thiết kế và tích hợp các thành phần khác nhau, bao gồm các phần mềm và các chức năng riêng lẻ thành một hệ thống hoàn chỉnh. Kiến trúc SOA rất giống với cấu trúc của các phần mềm hướng đối tượng gồm nhiều module. Tuy nhiên khái niệm module trong SOA không đơn thuần là một gói phần mềm, hay một bộ thư viện nào đó. Thay vào đó, mỗi module trong một ứng dụng SOA là một dịch vụ được cung cấp rải rác ở nhiều nơi khác nhau và có thể truy cập thông qua môi trường mạng, và mỗi module đóng vai trò là một “dịch vụ có tính kết nối lõng lẻo”. Nói một cách ngắn gọn, một hệ thống SOA là một tập hợp nhiều dịch vụ được cung cấp trên mạng, được tích hợp lại với nhau để cùng cộng tác thực hiện các tác vụ nào đó theo yêu cầu của khách hàng.

Dịch vụ (Service) là yếu tố then chốt trong SOA. Có thể hiểu dịch vụ như là một loại module thực hiện một quy trình nghiệp vụ nào đó. Một trong những mục đích của SOA là giúp các ứng dụng có thể “giao tiếp” được với nhau mà không cần biết các chi tiết kỹ thuật bên trong. Để thực hiện điều đó SOA định ra một chuẩn giao tiếp (dùng để gọi hàm dịch vụ) được định nghĩa rõ ràng và độc lập với nền tảng hệ thống và có thể tái sử dụng. Như vậy, SOA là cấp độ cao hơn của phát triển ứng dụng, chú trọng đến quy trình nghiệp vụ và dùng giao tiếp chuẩn để giúp che đi sự phức tạp kỹ thuật bên dưới. Sự trừu tượng là cốt lõi của khái niệm dịch vụ, nó giúp cho các doanh nghiệp có thể tích hợp các thành phần hiện có vào các ứng dụng mới và các thành phần này có thể được chia sẻ hoặc tái sử dụng trong nhiều lĩnh vực khác nhau của công ty đó mà không cần phải chỉnh sửa mã nguồn hay phải tái cấu trúc lại hệ thống.

Có nhiều cách khác nhau để kết nối các dịch vụ, chẳng hạn dùng các giao thức mạng có sẵn, hoặc tạo một giao thức riêng. Nhưng trong nhiều năm trở lại đây, các dịch vụ web (Web Services) được xây dựng dựa trên nền tảng web toàn cầu, bất cứ nơi nào cũng có và đã trở thành một phương pháp phổ biến cho việc kết nối các thành phần của hệ thống SOA với nhau. Thoạt nhìn SOA và Web Services trông có vẻ giống nhau nhưng chúng không phải là một.



Hình 1.8. Mô hình SOA cơ bản

Trong mô hình SOA cơ bản, nhà cung cấp dịch vụ nhận yêu cầu sử dụng dịch vụ từ người sử dụng, và phản hồi yêu cầu đến người sử dụng

Register

Find

Bind,

Execute

Hình 1.9. Mô hình tổng quan của SOA

Gồm các thành phần:

Service Provider: Cung cấp các service phục vụ cho một nhu cầu nào đó. Người sử dụng dịch vụ (service consumer) không cần quan tâm đến vị trí thực sự của service đó mà cần quan tâm service đó là gì.

Serive Consumer: khách hàng dịch vụ hay những người sử dụng service được cung cấp bởi Service Provider.

Service Registry: Nơi lưu trữ thông tin về các service của các Service Provider khác nhau, Service Consumer dựa trên những thông tin này để tìm kiếm và lựa chọn Service Provider phù hợp.

Service Provider sẽ đăng ký thông tin về service mà mình có thể cung cấp (các chức năng có thể cung cấp, khả năng của hệ thống [resource, performance, giá cả dịch vụ...]) vào Service Registry. Service Consumer khi có nhu cầu về một service nào đó sẽ tìm kiếm thông tin trên Service Registry. Ngoài chức năng hỗ trợ tìm kiếm, Service Registry còn có thể xếp hạng các Service Provider dựa trên các tiêu chí về chất lượng dịch vụ, bầu chọn từ các khách hàng đã sử dụng service... Những thông tin này sẽ hỗ trợ thêm cho quá trình tìm kiếm của Service Consumer. Khi đã xác định được Service Provider mong muốn, Service Consumer thiết lập kênh giao tiếp trực tiếp với Service Provider nhằm sử dụng service hoặc tiến hành thương lượng thêm (về mặt giá cả, resource sử dụng...).

SOA cung cấp giải pháp để giải quyết các vấn đề tồn tại của các hệ thống hiện nay như: phức tạp, không linh hoạt và không ổn định. Một hệ thống triển khai theo mô hình SOA có khả năng dễ mở rộng, liên kết tốt. Đây chính là cơ sở và nền tảng cho việc tích hợp, tái sử dụng lại những tài nguyên hiện có.

So với kiểu thiết kế Component-Based (hướng thành phần), điểm khác biệt chính của SOA là cung cấp khả năng giao tiếp giữa các thành phần trong hệ thống sử dụng thông điệp (message) dựa trên các giao thức đã được chuẩn hóa (HTTP, FTP, SMTP...). Chính nhờ đặc điểm này, hệ thống SOA trở nên độc lập với nền tảng (platform independent). Các service hoạt động trên các nền tảng khác nhau vẫn có thể giao tiếp với nhau nhờ vào các interface giao tiếp đã được chuẩn hóa để cộng tác xử lý một tác vụ nào đó.



Hình 1.10. Message được truyền nhận giữa các dịch vụ

### Các tính chất của một hệ thống SOA

#### Kết nối lỏng (Loose coupling)

Vấn đề kết nối nói tới một số ràng buộc giữa các module lại với nhau. Có 2 loại kết nối là lỏng lẻo và chặt chẽ. Các module có tính chất kết nối lỏng lẻocó một số ràng buộc được mô tả rõ ràng trong khi các module có tính kết nối chặt lại có nhiều ràng buộc không thể biết trước. Hầu như mọi kiến trúc phần mềm đều hướng đến tính kết nối lỏng lẻogiữa các module. Mức độ kết nối của hệ thống ảnh hưởng trực tiếp đến khả năng chỉnh sửa hệ thống. Kết nối càng chặt thì càng có chỉnh sửa khi có sự thay đổi nào đó xảy ra. Mức độ kết nối tăng dần khi bên sử dụng dịch vụ cần biết nhiều thông tin ngầm định của bên cung cấp dịch vụ để sử dụng dịch vụ được cung cấp. Nghĩa là nếu bên sử dụng dịch vụ biết vị trí và chi tiết định dạng dữ liệu của bên cung cấp dịch vụ thì quan hệ sẽ càng trở nên chặt chẽ. Ngược lại, nếu bên sử dụng dịch vụ không cần biết mọi thông tin chi tiết của dịch vụ trước khi triệu gọi nó thì quan hệ giữa 2 bên càng có tính lỏng lẻo*.*

Kết nối lỏng lẻo làm cho sự phụ thuộc ở mức tối thiểu. Khi đó, những sự thay đổi sẽ có ảnh hưởng ít nhất tới hệ thống và hệ thống vẫn có thể hoạt động khi có thành phần nào đó bị hư hỏng. Tối thiểu hóa sự phụ thuộc giúp hệ thống linh hoạt và ít xảy ra sự cố.

Tính kết nối lỏng lẻo giúp gỡ bỏ những ràng buộc điều khiển giữa những hệ thống đầu cuối. Mỗi hệ thống có thể tự quản lý độc lập nhằm tăng năng xuất, khả năng mở rộng và khả năng đáp ứng cao. Những thay đổi cài đặt cũng được che dấu. Tính chất kết nối lỏng lẻo đem đến sự độc lập giữa bên cung cấp và bên sử dụng nhưng nó đòi hỏi các giao diện phải theo chuẩn và cần một thành phần trung gian quản lý, trung chuyển yêu cầu giữa các hệ thống đầu cuối.

#### Tái sử dụng dịch vụ

Bởi vì các dịch vụ được cung cấp trên môi trường mạng và được đăng ký ở một nơi nhất định nên chúng dễ dàng được tìm thấy và sử dụng lại. Nếu một dịch vụ không có khả năng tái sử dụng, nó cũng không cần đến giao diện mô tả. Các dịch vụ có thể được tái sử dụng lại bằng cách kết hợp lại với nhau theo nhiều mục đích khác nhau. Tái sử dụng lại các dịch vụ còn giúp loại bỏ những thành phần trùng lặp và tăng tốc độ vững chắc trong cài đặt, nó còn giúp đơn giản hóa việc quản trị. Thực ra tái sử dụng dịch vụ lại dễ dàng hơn tái sử dụng thành tố hay lớp. Những dịch vụ được dùng chung bởi tất cả các ứng dụng của một hệ thống SOA gọi là những dịch vụ chia sẻ cơ sở hạ tầng.

#### Quản lý chính sách

Tập các chính sách là tập tất cả các qui tắc chung mà mọi thành phần trong hệ thống đều phải tuân thủ. Khi sử dụng các dịch vụ chia sẻ trên mạng, tùy theo mỗi ứng dụng sẽ có một luật kết hợp riêng gọi là các chính sách. Các chính sách cần được quản lý và áp dụng cho mỗi dịch vụ cả trong quá trình thiết kế và trong thời gian triển khai.

Việc đó làm tăng khả năng tạo ra các dịch vụ có đặc tính tái sử dụng. Bởi vì các chính sách được thiết kế tách biệt và tùy vào mỗi ứng dụng nên giảm tối đa các thay đổi phần mềm. Nếu không sử dụng các chính sách, thì các nhân viên phát triển phần mềm, nhóm điều hành và nhóm hỗ trợ phải làm việc với nhau trong suốt thời gian phát triển để cài đặt và kiểm tra những chính sách. Ngược lại, nếu sử dụng các chính sách, những nhân viên phát triển phần mềm chỉ cần tập trung vào quy trình nghiệp vụ trong khi nhóm điều hành và nhóm hỗ trợ tập trung vào các luật kết hợp.

#### Tự động dò tìm và ràng buộc động

SOA hỗ trợ khái niệm khai thác dịch vụ (service discovery). Một người sử dụng cần đến một dịch vụ nào đó có thể tìm kiếm dịch vụ dựa trên một số tiêu chuẩn khi cần. Người sử dụng chỉ cần hỏi một registry về một dịch vụ nào thỏa yêu cầu tìm kiếm. Ví dụ, một hệ thống chuyển khoản, khách hàng yêu cầu một registry tìm tất cả các dịch vụ có khả năng kiểm tra thẻ tín dụng. Registry trả về một tập các danh mục thỏa mãn yêu cầu. Các mục đó chứa thông tin về dịch vụ, bao gồm cả chi phí giao dịch. Bên sử dụng sẽ chọn một dịch vụ có phí giao dịch thấp nhất trong danh sách các dịch vụ trả về, kết nối đến nhà cung cấp dịch vụ dựa trên thông tin địa chỉ registry đã cung cấp để sử dụng dịch vụ kiểm tra thẻ tín dụng. Trong phần mô tả dịch vụ kèm theo đã có tất cả các tham số cần thiết dùng để thực thi dịch vụ, bên sử dụng chỉ cần định dạng dữ liệu yêu cầu đúng theo mô tả và gửi đi. Nhà cung cấp dịch vụ sẽ thực thi kiểm tra thẻ tín dụng và trả về một thông điệp có định dạng đúng như trong phần mô tả dịch vụ. Mối ràng buộc duy nhất giữa bên cung cấp và bên sử dụng là bản hợp đồng được cung cấp bởi registry trung gian. Mối ràng buộc này là ràng buộc trong thời gian chạy. Tất cả thông tin cần thiết về dịch vụ được lấy về và sử dụng trong khi chạy. Vậy với SOA, bên sử dụng dịch vụ không cần biết định dạng của thông điệp yêu cầu và thông điệp trả về, cũng như địa chỉ dịch vụ cho đến khi cần.

#### Khả năng tự phục hồi

Với kích cỡ và độ phức tạp của những hệ thống phân tán ngày nay, khả năng phục hồi của một hệ thống sau khi bị sự cố là một yếu tố rất quan trọng. Một hệ thống tự phục hồi là hệ thống có khả năng tự phục hồi sau khi lỗi mà không cần sự can thiệp của con người.

Độ tin cậy là mức độ đo khả năng của một hệ thống xử lý tốt như thế nào trong tình trạng hỗn loạn. Trong SOA, các dịch vụ luôn có thể hoạt động hay ngừng hoạt động bất cứ lúc nào, nhất là đối với những áp dụng tổng hợp từ nhiều dịch vụ của nhiều tổ chức khác nhau. Độ tin cậy phụ thuộc vào khả năng phục hồi của phần cứng sau khi bị lỗi. Hạ tầng mạng phải cho phép các kết nối động từ nhiều hệ thống khác nhau kết nối đến trong khi chạy. Một khía cạnh khác ảnh hưởng đến độ tin cậy là kiến trúc mà dựa trên đó những ứng dụng được xây dựng. Một kiến trúc hỗ trợ kết nối và thực thi động sẽ có khả năng tự phục hồi hơn một hệ thống không hỗ trợ những tính năng trên.

Ngoài ra, những hệ thống dựa trên dịch vụ yêu cầu tách biệt giữa giao diện và cài đặt, nên có thể có nhiều cài đặt khác nhau cho cùng một giao diện. Nếu một thể hiện service nào đó không hoạt động thì một thể hiện khác vẫn có thể hoàn tất giao dịch cho khách hàng mà không bị ảnh hưởng gì. Khả năng này chỉ có được khi client tương tác với giao diện của dịch vụ chứ không tương tác trực tiếp cài đặt của dịch vụ. Đây là một trong những tính chất cơ bản của hệ thống hướng dịch vụ (SOA).

#### Khả năng cộng tác

Kiến trúc hướng dịch vụ nhấn mạnh đến khả năng cộng tác (*Interoperability*), khả năng mà các hệ thống có thể giao tiếp với nhau trên nhiều nền tảng và ngôn ngữ khác nhau. Mỗi dịch vụ cung cấp một interface có thể được triệu gọi thông qua một dạng kết nối. Một kết nối gọi là interoperable chứa bên trong nó một giao thức và một định dạng dữ liệu mà mỗi client kết nối đến nó đều hiểu. Khả năng cộng tác đạt được bằng cách hỗ trợ các giao thức và định dạng dữ liệu chuẩn của dịch vụ và các client. Kỹ thuật này đạt được bằng cách ánh xạ mỗi tính chất và ngôn ngữ qua một đặc tả trung gian. Đặc tả trung gian sẽ chịu trách nhiệm ánh xạ giữa định dạng của dữ liệu khả kết (interoperable) đến định dạng dữ liệu tùy thuộc vào nền tảng hệ thống. Ví dụ Web Services là một đặc tả trung gian cho giao tiếp giữa các hệ thống, JAX-RPC và JAXM chuyển đối tượng dạng Java thành SOAP.

### Kiến trúc phân tầng chi tiết của SOA

Nhấn mạnh rằng, SOA là một phương pháp luận giúp chúng ta tận dụng sức mạnh của các nguồn lực, nguồn tài nguyên khác nhau trong mạng máy tính để trở thành một hệ thống nhất. Nên hiện tại không có mô hình thống nhất cho các thành phần của hệ thống SOA, mỗi công ty, tổ chức khi phát triển một hệ thống SOA có thể đưa ra mô hình các thành phần của SOA khác nhau. Đây là mô hình các thành phần của hệ thống SOA theo quan điểm của công ty IBM [5] và đây cũng là một mô hình khá phổ biến cho kiến trúc của hệ thống SOA.



Hình 1.11. Kiến trúc phân tầng của SOA

***Enterprise Service Bus (ESB):***Là thành phần cơ bản của SOA cung cấp khả năng kết nối cần thiết cho những dịch vụ trong toàn bộ hệ thống, bao gồm cả dịch vụ liên quan tới thực hiện giao vận (transport), quản lý tình huống (event) và điều phối (mediation). ESB cho phép nhà phát triển tận dụng giá trị của phương thức giao tiếp qua gửi nhận thông điệp mà không phải thực hiện viết những đoạn mã chuyên biệt.

***Dịch vụ tương tác (interaction services)***có trách nhiệm trình bày các mô hình nghiệp vụ. Nói cách khác, đây là những thành phần giúp các ứng dụng và người dùng cuối giao tiếp với nhau, ở đây người dùng cuối (enduser) không chỉ là con người, mà cũng có thể là cảm biến, robot,...

***Dịch vụ quy trình (Process services)***chịu trách nhiệm cho logic thành phần. Thành phần là tập hợp các dịch vụ mà được tạo một luồng tiến trình nghiệp vụ. Và các dịch vụ quy trình tạo ra các cơ chế thành phần.

***Dịch vụ thông tin (Information services)***chịu trách nhiệm về tính logic của dữ liệu, dịch vụ này cung cấp các chức năng tập hợp, thay thế và chuyển đổi nhiều nguồn dữ liệu khác nhau được thực hiện bởi nhiều cách thức khác nhau. Những dịch vụ này có mặt ở hai cấp độ: cấp độ bên ngoài đảm bảo việc cung cấp truy cập vào các dữ liệu của doanh nghiệp và cấp độ bên trong đảm bảo luồng dữ liệu trong tổ chức.

***Dịch vụ tích hợp với bên ngoài (Partner services)***có trách nhiệm thu thập thông tin về đối tác (ví dụ như các chính sách và hạn chế) và cung cấp tài liệu, giao thức, các chức năng quản lý đối tác cho những quy trình nghiệp vụ có yêu cầu tương tác với đối tác bên ngoài và nhà cung cấp.

***Dịch vụ ứng dụng nghiệp vụ (Business application services)***chịu trách nhiệm về logic nghiệp vụ cốt lõi. Đây là những dịch vụ được tạo ra đặc biệt để thực hiện các mô hình nghiệp vụ. Chúng đại diện cho các khối xây dựng cơ bản, cho việc thiết kế quy trình nghiệp vụ . Những dịch vụ này không thể bị phân hủy, thay vì kết nối với các dịch vụ khác để tạo thành một quy trình nghiệp vụ.

***Dịch vụ truy cập (Access services)***có trách nhiệm kết nối các ứng dụng và chức năng vào kiến trúc hướng dịch vụ. Cung cấp các chức năng bắc cầu cho những ứng dụng cũ (legacy applications), kho dữ liệu chính, và ESB nhằm kết hợp dịch vụ có trong những ứng dụng hiện tại vào hệ thống SOA.

***Dịch vụ đổi mới và tối ưu hóa nghiệp vụ (Business innovation and optimization services)***có trách nhiệm cung cấp các công cụ và các cấu trúc siêu dữ liệu để đại diện cho thiết kế nghiệp vụ, bao gồm cả chính sách và mục tiêu nghiệp vụ.

***Dịch vụ phát triển (Development services)***cung cấp môi trường tích hợp cho thiết kế và tạo ra giải pháp.

***Dịch vụ cơ sở hạ tầng (Infrastructure services)***Những dịch vụ này có trách nhiệm lưu trữ các ứng dụng SOA, giúp cung cấp sử dụng hiệu quả các nguồn tài nguyên để tối ưu băng thông, sẵn sàng và hiệu năng.

Đây chỉ là một tổng quan về các thành phần chính của mô hình SOA. Như đã đề cập, các công ty khác nhau cung cấp cách nhìn khác nhau trên cùng một vấn đề. Tuy nhiên, các nguyên tắc chính vẫn như cũ. SOA là một kiến trúc dựa trên dịch vụ, đóng gói, tái sử dụng và kết nối lỏng lẻo. Đây là dịch vụ tạo ra giá trị kinh doanh, hỗ trợ tin học và thế giới nghiệp vụ để giao tiếp một cách thích hợp. Dịch vụ có thể được truy cập và sử dụng từ bên trong tổ chức với sự giúp đỡ của ESB.

## Ngôn ngữ thi hành quy trình nghiệp vụ - BPEL

### Giới thiệu

Web Services Business Process Execution Language (viết tắt là WS-BPEL hay được gọi là BPEL) là một ngôn ngữ thi hành quy trình nghiệp vụ dùng để hỗ trợ phát triển các ứng dụng phức tạp, lớn đòi hỏi phải tổng hợp nhiều Web Services khác nhau. Phiên bản BPEL đầu tiên (BPEL 1.0) ra đời vào tháng 07/2002. Vào tháng 05/2003 BPEL1.1 ra đời dựa trên việc kết hợp BPEL 1.0 với một số ngôn ngữ khác và được đệ trình lên tổ chức OASIS [16] (một tổ chức chuyên đưa ra các chuẩn thông tin). Tháng 04/2007 tổ chức OASIS chuẩn hóa BPEL và đổi tên thành WS-BPEL 2.0 được dùng cho đến nay.

BPEL cho phép đặc tả và xử lý luồng công việc bằng cách cung cấp sẵn các thẻ mô tả các đối tượng và các hoạt động xử lý của một quy trình nghiệp vụ thông thường. Ngoài ra BPEL còn định nghĩa các cách quản lý các sự kiện và ngoại lệ, cơ chế bảo toàn dữ liệu khi có ngoại lệ xảy ra. BPEL hoạt động dựa trên nguyên tắc gửi các thông điệp dạng XML đến một dịch vụ khác, thao tác trên cấu trúc XML, nhận các thông điệp XML (đồng bộ hay không đồng bộ) từ các dịch vụ bên ngoài. Nó phụ thuộc vào bốn chuẩn XML cơ bản được xem như là các đặc tả để thực thi một tiến trình BPEL: WSDL, XML Schema, XPath và WS-Addressing.



Hình 1.12. Quy trình tích hợp với các dịch vụ đối tác

### Các khái niệm cơ bản

#### Nguyên tắc hoạt động của một tiến trình BPEL

Trong số các đặc tả: WSDL, XML Schema, XPath và WS-Addressing thì WSDL đóng vai trò cốt lõi trong một tiến trình của BPEL. Cốt lõi BPEL là khái niệm peer-to-peer là sự tương tác giữa các Web Services được mô tả trong WSDL. Một tiến trình BPEL làm thế nào để phối hợp giữa các dịch vụ khác nhau và kết hợp các dịch vụ đó lại thành một tiến trình hoàn chỉnh. Điều này có nghĩa một tiến trình BPEL phải có đặc tả của riêng nó đồng thời sử dụng các đặc tả WSDL được cung cấp bởi các dịch vụ khác trong quá trình tổng hợp các dịch vụ này.

Tuy nhiên có một vấn đề đặt ra là làm thế nào để tiến trình BPEL có thể phân biệt được từng dịch vụ mà nó tổng hợp trên đó để mà tương tác trên những dịch vụ đó; cũng như mỗi dịch vụ được sử dụng trong tiến trình BPEL có vai trò như thế nào trong toàn bộ tiến trình… Để giải quyết vấn đề này, BPEL đã đưa ra hai khái niệm mới là kiểu liên kết ngoài (*partnerLinkType)* và liên kết ngoài (*partnerLink)*

Kiểu liên kết ngoài (partnerLinkType): Đặc tả mối quan hệ giữa hai dịch vụ bằng cách định nghĩa vai trò (role) của mỗi dịch vụ trong mối liên hệ và quy định portType cụ thể cung cấp cho mỗi dịch vụ bằng cách nhận các thông báo bên trong mối liên hệ cụ thể. Mỗi vai trò đặc tả chính xác một WSDL portType. Cú pháp khai báo:

<partnerLinkType name=”PNLT\_Name”>

<role name=”role\_name1” portType name=”pname1”/>

<role name=”role\_name2” portType name=”pname2”/>

</partnerLinkType>

* Tên của partnerLinkType được khai báo trong thuộc tính name của thẻ partnerLinkType.
* Hai thẻ bên trong nút gốc khai báo hai vai trò tương ứng với hai PortType.

Liên kết ngoài (partnerLink): Tương tác giữa các dịch vụ với tiến trình nghiệp vụ được mô hình hóa giống như một partner links trong WS-BPEL. Mỗi <partnerLink> được cụ thể hóa bằng một partnerLinkType. Nhiều hơn một <partnerLink> có thể được cụ thể hóa bằng cùng một partnerLinkType. Cú pháp khai báo:

<partnerLink>

< partnerLink name=”name”

partnerLinkType=”plt\_name”

myRole=”mr\_name”?

partnerRole=”pr\_name”?

initializePartnerRole=”yes|no”? />+

</partnerLink>

* Có thể khai báo nhiều partnerLink khác nhau trong thẻ <partnerLinks>
* Tên của partnerLink được khai báo trong thuộc tính name của thẻ partnerLink.
* Thuộc tính partnerLinkType khai báo tên của partnerLinkType sử dụng
* Thuộc tính myRole khai báo vai trò của tiến trình WS-BPEL
* Thuộc tính partnerRole khai báo vai trò của đối tác
* Thuộc tính initializePartnerRole xác định liệu bộ xử WS-BPEL cần thiết khởi tạo giá trị partnerRole của một partnerLink hay không. Việc khởi tạo này không ảnh hưởng đến giá trị của partnerRole sau khi khởi tạo nó. Nó chỉ là một quy định để chắc chắn rằng đối tác sẽ luôn cung cấp dịch vụ trong quá trình triển khai

#### Cấu trúc của một tiến trình

<bpel:process name=*"test"* >

<!-- Import the client WSDL -->

<bpel:import location=*"testArtifacts.wsdl"*

namespace=*"http://eclipse.org/bpel/sample"*

importType=*"http://schemas.xmlsoap.org/wsdl/"* />

<bpel:partnerLinks>

………………

</bpel:partnerLinks>

<bpel:variables>

………………

</bpel:variables>

<bpel:sequence name=*"main"*>

………………

</bpel:sequence>

</bpel:process>

Một tiến trình BPEL là một mô tả dưới dạng tài liệu XML.

Hình 1.13. Cấu trúc file BPEL

Ý nghĩa của các thành phần:

**<bpel:process>:** Mọi file BPEL đều bắt đầu với thẻ <bpel:*process*>. Các mô tả cho tiến trình cũng như các namespace liên quan được định nghĩa ở thẻ này.

**<bpel:imports>:** Chứa thông tin các tập tin WSDL được import vào.

**<bpel:partnerLinks>:** Chứa tập hợp các partnerLink được sử dụng trong tiến trình. Mỗi partnerLink sẽ thiết lập một quan hệ giữa bản thân process với một service bên ngoài.

**<bpel:variables>:** Phần này định nghĩa các biến được dùng trong tiến trình. Mỗi biến đều phải được tham chiếu đến một kiểu thông điệp (*messageType*) được mô tả trong tập tin WSDL.

**<bpel:sequence>:** Đây là phần chính mô tả logic của tiến trình. Trong một **<bpel:sequence>** sẽ chứa nhiều tác vụ (được trình bày chi tiết bên dưới). Mỗi tác vụ có một nhiệm vụ cụ thể trong tiến trình BPEL. Bản thân **<bpel:sequence>** cũng là một tác vụ, có thể chứa các cấu trúc song song cũng như cấu trúc tuần tự khác.

## Tiểu kết chương 1

Qua các kiến thức tổng quan ta đã thấy rõ ràng kiến trúc hướng dịch vụ (SOA) là một kiểu kiến trúc có khả năng tái sử dụng lại các tài nguyên sẵn có, khả năng mở rộng và liên kết tốt với các hệ thống mới để tạo nên một môi trường đồng nhất, nó bao gồm các dịch vụ nghiệp vụ độc lập, không đồng nhất được kết hợp với nhau trong quy trình nghiệp vụ linh hoạt mềm dẻo. Và để triển khai kiến trúc hướng dịch vụ, công nghệ Web Services là lựa chọn lý tưởng bởi khả năng đáp ứng mềm dẻo và linh hoạt của nó. Thêm vào đó để kết hợp được các dịch vụ Web thành một quy trình nghiệp vụ hoàn chỉnh, người ta sử dụng ngôn ngữ mô phỏng và thực thi tiến trình nghiệp vụ có tên là BPEL. Ngôn ngữ BPEL sẽ định nghĩa tiến trình, các dịch vụ ngoài và sử dụng các tác vụ, các phép toán logic để tạo thành một quy trình.

Tóm lại, công nghệ Web Services cùng với ngôn ngữ thi hành quy trình nghiệp vụ - BPEL đã hiện thực hóa kiến trúc hướng dịch vụ (SOA), cho phép kết hợp các dịch vụ đơn lẻ và các hệ thống ứng dụng thành một quy trình nghiệp vụ đầy đủ.

# Chương 2. Khung ứng dụng hỗ trợ lập trình SOA



## Nền tảng Eclipse

### Giới thiệu

Eclipse là một nền tảng phát triển phần mềm mã nguồn mở, bao gồm một IDE (Intergrated Development Environment) và một hệ thống plug-in có khả năng mở rộng được. Trong khi hầu hết người dùng sử dụng Eclipse như là môi trường phát triển tích hợp Java (IDE), một số khác lại có nhu cầu mở rộng Eclipse. Do đó, Eclipse xây dựng PDE (Plug-in Development Environment) dành cho người dùng muốn mở rộng Eclipse. PDE cho phép người xây dựng các công cụ tích hợp vào môi trường Eclipse và mọi thứ trong PDE đều là plug-in.

Mặc dù Eclipse được viết bằng ngôn ngữ Java, nhưng Eclipse có cung cấp các plug-in hỗ trợ các ngôn ngữ lập trình khác như C/C++, COBOL, Fortran, Javascript, php, ruby.... Eclipse framework có thể được sử dụng như là nền tảng cho các loại ứng dụng khác không liên quan đến phát triển phần mềm, ví dụ: hệ quản trị nội dung.

Dự án Eclipse được IBM bắt đầu vào tháng 11 năm 2001 và được các nhà bán phần mềm hỗ trợ. Eclipse là cộng đồng mã nguồn mở, các project của nó tập trung vào việc xây dựng nền tảng mã nguồn mở bao gồm các framework có thể mở rộng, các tool và thư viện để xây dựng, triển khai và quản lý các phần mềm.

Eclipse là một tổ chức phi lợi nhuận, là nơi phát triển cộng đồng mã nguồn mở và hệ thống các sản phẩm, dịch vụ. Những người phát triển plug-in cho Eclipse hoặc những người sử dụng Eclipse như là nền tảng cho các ứng dụng phát triển phần mềm cần tuân thủ giấy phép EPL.

### Các thành phần và kiến trúc

Kiến trúc của Eclipse được xây dựng dựa trên hai thành phần chính: thành phần lõi (core) và các thành phần gắn thêm (plug-in).

Thành phần lõi (core): bao gồm các chức năng, dịch vụ mà các hệ phát triển ứng dụng phải có như chức năng cung cấp giao diện, trình soạn thảo văn bản, gỡ lỗi… cần cho mọi nền tảng lập trình (cần cho các plug-in).

Thành phần gắn thêm (plug-in): bao gồm nhiều thành phần dễ dàng tích hợp vào nhiều ứng dụng chạy trên nền Eclipse. Các chức năng của thành phần core tách biệt với các chức năng của phần giao diện.

Kiến trúc Eclipseđược thiết kế và xây dựng để đáp ứng các yêu cầu sau:

Hỗ trợ việc xây dựng các công cụ khác nhau để phát triển ứng dụng

Hỗ trợ các công cụ để thao tác trên bất kỳ loại nội dung nào.Ví dụ: HTML, Java, C, C++, JSP, PHP, EJB, XML và GIF...

Hỗ trợ tích hợp các công cụ một cách dễ dàng

Hỗ trợ môi trường phát triển ứng dụng giao diện và không giao diện

Chạy được trên nhiều hệ điều hành, gồm Window, Linux, Android,…

Kiến trúc nền tảng Eclipsecung cấp các công cụ cùng cơ chế sử dụng và quy định để có thể tích hợp các công cụ một cách dễ dàng. Những cơ chế này được thể hiện thông qua giao diện API, lớp và phương thức. Ngoài ra, nền tảng Eclipsecòn cung cấp các frameworkđể dễ dàng phát triển các công cụ mới. EclipseSDKbao gồm 3 phần chính:

Platform

Java Development Toolkit (JDT)

Plug-in Development Environment (PDE)

Với JDT, Eclipse là một môi trường hỗ trợ phát triển Java. JDT cũng có thể được coi như là một Plug-in cho Eclipse như là một môi trường lập trình tích hợp (Java IDE -Integrated Development Enviroment). PDE hỗ trợ việc mở rộng Eclipse, tích hợp các Plug-in vào Eclipse Platform. Eclipse Platform là nền tảng của toàn bộ phần mềm Eclipse, cung cấp những dịch vụ cần thiết cho việc tích hợp những bộ công cụ phát triển phần mềm dưới dạng Plug-in. Eclipse Platform được xây dựng dựa trên cơ chế phát hiện, tích hợp và chạy các thành phần gắn thêm. Plug-in là thành phần gắn thêm có thể hoạt động độc lập hoặc cùng với các thành phần khác. Thông thường, một ứng dụng có thể tổ hợp nhiều thành phần gắn thêm, mỗi thành phần gắn thêm này lại nối với các thành phần gắn thêm khác tùy vào mục đích sử dụng



Hình 2.1. Kiến trúc tổng quan Eclipse

**The Platform runtime** với công việc chính là quản lý plug-inđang có trong thư mục plug-incủa Eclipse. Mỗi plug-inđều có 1 tập tin Manifestliệt kê những kết nối mà plug-incần. Plug-inchỉ được tải vào Eclipsekhi thực sự cần thiết để giảm lượng tài nguyên yêu cầu và thời gian khởi tạo.

**Workspace** chịu trách nhiệm quản lý tài nguyên người dùng được tổ chức dưới dạng Project. Mỗi dự án là một thư mục con trong thư Workspace. Mỗi dự án có chứa các tập tin và thao tác của người dùng. Workspace bảo quản cấp thấp lịch sử những sự thay đổi tài nguyên, tránh thất thoát tài nguyên người dùng, đồng thời chịu trách nhiệm thông báo những công cụ cần thiết cho việc thay đổi tài nguyên. Tài nguyên được tổ chức dưới dạng cấu trúc cây rất hữu ích và hiệu quả. Từ đó, mỗi nút dữ liệu chỉ ra nơi mà các công cụ cần thêm, loại bỏ, hoặc làm mới.

**Workbench** là giao diện đồ họa người dùng của Eclipse, gồm có Standard Widget Toolkit (SWT) và Jface. Eclipse không hoàn toàn bắt buộc phải sử dụng SWT hay JFace để lập trình giao diện, người lập trình vẫn có thể sử dụng AWT hay SWING của Java thông qua việc cài đặt các plug-ins.

***SWT (***Standard Widget Toolkit) là một gói công cụ mã nguồn mở, cung cấp cho các lập trình viên Java giải pháp để phát triển giao diện đồ họa người dùng.

***JFace*** là gói công cụ để xây dựng giao diện người dùng cấp cao, JFace là tầng trên cùng của SWT, cung cấp các lớp thuộc mô hình MVC (Model-View- Controller) để phát triển các ứng dụng đồ họa dễ dàng hơn.

**Team support** trang bị hệ thống quản trị để quản lý dự án của người dùng: Concurrent Versions System (CVS)

**Help c**ung cấp hệ thống tài liệu mở rộng, có thể là định dạng HTML hay XML.

**Java Development Tools (JDT):** tập các plug-in đem lại công cụ hỗ trợ phát triển các ứng dụng viết bằng Java.

**Plug-in Development Environment (PDE):** tập các plug-in đem lại công cụ hỗ trợ phát triển plug-in cho Eclipse.

## Kiến trúc Mô hình plug-in Eclipse

Plug-in trong Eclipse là một thành phần cung cấp một loại dịch vụ nhất định trong môi trường lập trình Eclipse. Tương tự, Eclipse cung cấp cơ sở để hỗ trợ kết nối và thực thi một tập các plug-in, hình thành một môi trường lập trình đồng nhất. Khi một chương trình Eclipse được thực thi, plug-in sẽ là một đối tượng được thể hiện trong các plug-in class. Plug-in class sẽ cung cấp cấu hình và hỗ trợ quản lý cho các đối tượng plug-in. Plug-in class phải được kế thừa từ org.eclipse.core.runtime.Plugin.

Trong cấu trúc thư mục của Eclipse, các plug-in đã được cài đặt sẽ được chứa trong thư mục plugins. Plug-in được mô tả bởi một tập tin manifest XML, gọi là plugin.xml. Tập tin manifest sẽ chỉ ra cách để Eclipse kích hoạt plug-in đó.

Nội dung truyền vào tập tin manifest của plug-in bằng cách lập trình thông qua plug-in registry API. Các đặc tả của plug-in được lưu trữ trong bộ nhớ gọi là plug-in registry. Khi Eclipse thực thi, nó sẽ khởi tạo đối tượng của plug-in bằng cách sử dụng plug-in registry API, API này sau đó sẽ đoạn mã được cung cấp để lấy thông tin về plug-in.

Một đoạn manifest plug-in có dạng như sau:

<?xml version="1.0" encoding="UTF-8"?>

<plugin

name="JUnit Testing Framework"

id="org.junit"

version="3.7"

provider-name="Eclipse.org">

<runtime>

<library name="junit.jar">

<export name="\*"/>

</library>

</runtime>

</plugin>

Tập tin manifest này mô tả một plug-in cung cấp dịch vụ kiểm thử JUnit. Trong đoạn manifest trên, ta chú ý rằng mỗi plug-in có một số định danh duy nhất (thuộc tính XML id). Giá trị định danh này sẽ được sử dụng để tham chiếu đến plug-in. Giá trị này cũng có thể được sử dụng để truy cập đến các đối tượng đang thực thi của plug-in, bằng đoạn mã sau:

Plugin p = Platform.getPlugin(pluginID);

Các đối tượng plug-in được quản lý bởi quá trình thực thi của Eclipse, và được truy cập như giới thiệu ở trên. Nó không được tạo ra bởi các chương trình ứng dụng.

Thông tin mô tả chi tiết plug-in nằm trong tập tin manifest. Eclipse sẽ sử dụng tập tin manifest để tích hợp plug-in vào khung ứng dụng. Tập tin manifest chứa những thông tin chung về plug-in bao gồm: tên plug-in, phiên bản, tên lớp, tên tập tin JAR. Ngoài ra, tập tin manifest còn chứa danh sách những plug-in mà plug-in hiện tại sử dụng.



Hình 2.2. Minh họa một tập tin plug-in manifest

### Cài đặt và kích hoạt Plug-in

Để sử dụng một plug-in cần sao chép các tài nguyên tạo nên plug-in (tập tin manifest, tập tin jar, và các tập tin tài nguyên khác) vào trong thư mục plugins ở đường dẫn cài đặt Eclipse. Plug-in này sẽ được kích hoạt khi Eclipse thực thi chương trình, và có yêu cầu thực hiện chức năng liên quan đến plug-in đó. Kích hoạt plug-in đồng nghĩa với nạp các class và khởi tạo các biến đối tượng ban đầu.

Chức năng chính của một plug-in class là thực hiện xử lý trong quá trình kích hoạt và bỏ kích hoạt, ví dụ như cấp phát và giải phóng tài nguyên. Với một số plug-in đơn giản, như JUnit trên, không yêu cầu kích hoạt hay bỏ kích hoạt, và do đó không cần cung cấp plug-in class. Trong trường hợp này, Eclipse sẽ tự động cung cấp plug-in class mặc định cho đối tượng của plug-in.

Nếu plug-in có liên quan đến kích hoạt hay bỏ kích hoạt, lập trình viên cần sử dụng subclass *org.eclipse.core.runtime.Plugin*, cung cấp cách thức kích hoạt hay bỏ kích hoạt, như là startup hay shutdown, bao gồm cả tên của một plug-in cụ thể nào đó như là giá trị cho thuộc tính class tương ứng trong tập tin manifest.

Trong mô hình Eclipse, một plug-in có thể liên quan đến plug-in khác theo dạng quan hệ sau:

Phụ thuộc (Dependency): Vai trò trong mối quan hệ này là plug-in phụ thuộc và plug-in tiên quyết. Một plug-in tiên quyết sẽ hỗ trợ chức năng cho plug-in phụ thuộc.

Mở rộng (Extension): Vai trò trong mối quan hệ này là plug-in chính và plug-in mở rộng. Plug-in mở rộng sẽ bổ sung chức năng cho plug-in chính.

Mối quan hệ này được đặc tả trong tập tin manifest thông qua các thành phần XML là *requires* và *extension*.

### Phụ thuộc – Dependency

Khi một plug-in phụ thuộc vào chức năng của các plug-in khác, mối quan hệ này sẽ được đặc tả bởi thuộc tính *requires* trong tập tin manifest của plug-in. Dưới đây là ví dụ:

<?xml version="1.0" encoding="UTF-8"?>

<plugin

id="com.bolour.sample.eclipse.demo"

name="Extension Processing Demo"

version="1.0.0">

<runtime>

<library name="demo.jar"/>

</runtime>

<requires>

<import plugin="org.eclipse.ui"/>

</requires>

</plugin>

   Trong ví dụ này, plug-in *com.bolour.sample.eclipse.demo* được khai báo là phụ thuộc vào plug-in UI của Eclipse *org.eclipse.ui*.

Mối quan hệ phụ thuộc này được xác định trong tập tin manifest ở cả quá trình thực thi và quá trình biên dịch. Tại thời điểm thực thi, Eclipse sẽ đảm bảo các plug-in tiên quyết luôn khả dụng đối với các plug-in phụ thuộc khi nó được kích hoạt. Tại thời điểm biên dịch, Eclipse sẽ được dẫn hướng để cung cấp đường dẫn class (classpath) để biên dịch các plug-in phụ thuộc bằng các file jar và tất cả các plug-in tiên quyết của nó.

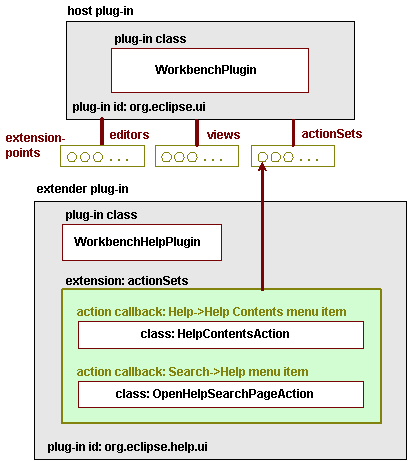
### Mở rộng – Extension

Khi các thành phần của plug-in được sử dụng bởi người sử dụng, một hay nhiều các giao diện người sử dụng (UI) của plug-in sẽ được thêm vào trong Eclipse. Quá trình này gọi là mở rộng (extension). Tuy nhiên, quá trình này không chỉ giới hạn cho các yếu tố UI. Bất cứ plug-in nào cũng có thể cho phép các plug-in khác mở rộng nó bằng cách thêm vào một số thành phần. Một extension được xác định bởi plug-in mở rộng (plug-in extender) và khiến plug-in chính (host plug-in) thay đổi chức năng của nó. Thông thường, việc thay đổi chức năng này có thể là thêm vào chức năng mới hoặc tùy chỉnh chức năng có sẵn thông qua các dịch vụ cung cấp bởi plug-in mở rộng.

Trong trường hợp đơn giản, một mở rộng thêm một hoặc nhiều đối tượng callback vào môi trường, thông qua đó để plug-in chính và plug-in mở rộng liên lạc với nhau. Tuy nhiên, lưu ý là các mô hình mở rộng là khá chung, và không nhất thiết phải yêu cầu một plug-in mở rộng cung cấp cho đối tượng callback tùy chỉnh.

Một plug-in có thể có nhiều loại mở rộng khác nhau. Đối với mỗi loại mở rộng, nó phải đi liền với tập các yêu cầu về cấu hình và trạng thái khác nhau. Chính vì vậy, một plug-in có khả năng mở rộng phải cung cấp nhiều loại tương thích (hay khe cắm - slot) khác nhau đối với các loại mở rộng tương ứng. Các loại tương thích này được gọi là điểm mở rộng (extension point).

Hình 2.3. minh họa mối quan hệ giữa các thành phần trong extension. Trong trường hợp này, phần mở rộng là các mục trong menu của chức năng Help của Eclipse. Trong mở rộng này, plug-in chính là giao diện người sử dụng của Eclipse, *org.eclipse.ui*, trong đó, các menu sẽ được mở rộng thông qua các điểm mở rộng là *actionSets*. Plug-in mở rộng là giao diện của Help trong Eclipse, *org.eclipse.help.ui*. Để sử dụng chức năng này, plug-in mở rộng này sẽ sử dụng các *actionSets* để mở rộng giao diện bằng cách thêm các mục trong menu của mình, ví dụ như *Help→Help Contents* và *Search→Help.* Mở rộng được định nghĩa bởi plug-in mở rộng.



Hình 2.3. Các thành phần của một plug-in extension

Hình trên cũng chỉ ra các đối tượng callback nằm trong class của extension. Đó là các class để xử lý menu của chức năng Help. Ta có thể thấy các class này được chỉ ra bởi tên trong phần khai báo của mỗi mở rộng. Ví dụ, với mở rộng menu *Help→Help Contents* khai báo class callback của nó là *HelpContentsAction*. Và mở rộng menu *Search→Help* khai báo class callback của nó là *OpenHelpSearchPageAction.*

#### Các thành phần của Extension

##### Plug-in chính (Host Plug-in)

Đối với một extension cụ thể, plug-in đóng vai trò chính sẽ cung cấp các điểm mở rộng (extension point) và được mở rộng. Bên cạnh việc cung cấp dịch vụ của riêng nó, plug-in này cũng làm nhiệm vụ điều phối và điều khiển các mở rộng khác.

Trong tập tin manifest của plug-in, các *extension-point* được khai báo là một thành phần XML *extension-point*. Như ví dụ bên dưới:

<?xml version="1.0" encoding="UTF-8"?>

<plugin

(1)

id="org.eclipse.ui"

name="Eclipse UI"

version="2.1.0"

provider-name="Eclipse.org"

class="org.eclipse.ui.internal.UIPlugin">

(2)

<extension-point id="actionSets" name="Action Sets"

schema="schema/actionSets.exsd"/>

<!-- Other specifications omitted. -->

</plugin>

Khai báo một extension-point

Điểm mở rộng (2) xác định định danh duy nhất cho điểm mở rộng trong plug-in chính. Để xác định điểm định danh duy nhất toàn cục, định danh mở rộng này cần phải sử dụng thêm một định danh nữa (là (1)) để hình thành định danh đầy đủ. Ví dụ, định danh đầy đủ của extension-point *actionSets* là *org.eclipse.ui.actionSets*.

##### Plug-in mở rộng (Extender plug-in)

Một plug-in có vai trò là mở rộng, khi nó thêm vào một số tính năng cho plug-in chính. Một mở rộng được khai báo bằng cách sử dụng thành phần XML *extension* trong file manifest.

Dưới đây là ví dụ của một plug-in mở rộng, mở rộng *actionSets* bằng cách thêm vào hai mục của menu.

<plugin

id="org.eclipse.help.ui"

name="Help System UI"

version="2.1.0"

provider-name="Eclipse.org"

class="org.eclipse.help.ui.internal.WorkbenchHelpPlugin">

<!-- ... -->

<!-- Action Sets -->

<extension

(1)

point="org.eclipse.ui.actionSets">

<actionSet

label="Help"

visible="true"

id="org.eclipse.help.internal.ui.HelpActionSet">

(2)

<action

label="&Help Contents"

icon="icons/view.gif"

helpContextId="org.eclipse.help.ui.helpContentsMenu"

tooltip="Open Help Contents"

class="org.eclipse.help.ui.internal.HelpContentsAction"

menubarPath="help/helpEnd"

id="org.eclipse.help.internal.ui.HelpAction">

</action>

<!-- ... other actionSet elements -->

(3)

<action

label="&Help..."

icon="icons/search\_menu.gif"

helpContextId="org.eclipse.help.ui.helpSearchMenu"

(4)

class="org.eclipse.help.ui.internal.OpenHelpSearchPageAction"

menubarPath="org.eclipse.search.menu/dialogGroup"

id="org.eclipse.help.ui.OpenHelpSearchPage">

</action>

</actionSet>

</extension>

<!-- ... -->

</plugin>

Hình 2.4. Khai báo một Extension

Lưu ý rằng trong plug-in mở rộng, điểm mở rộng *actionSets* được tham chiếu đến định danh ở (1).

Các hoạt động mở rộng được chỉ ra ở (2), (3) sẽ tạo ra hai mục menu mới là *Help→Help Contents* và *Search→Help.*

##### Extension callback

Một đối tượng callback là một đối tượng thuần Java (không phải là plug-in) được gọi bởi plug-in chính trong một số trường hợp nhất định tương ứng với các điểm mở rộng được xác định. Giao diện của các đối tượng callback được cung cấp bởi plug-in chính. Để sử dụng các đối tượng callback này, cần khai báo class đối với một extension cụ thể. Vì việc sử dụng các đối tượng callback này là mở rộng tham chiếu đến giao diện callback, mà thông thường được đóng gói trong plug-in chính, nên có thể nói các plug-in mở rộng cũng phụ thuộc vào plug-in chính.

**Đối tượng dịch vụ chi tiết**

Mỗi đối tượng callback có một vai trò nhất định trong mở rộng. Trong định nghĩa XML về mở rộng, callback được xác định bởi các thành phần XML con. Ví dụ, với điểm mở rộng *actionSets*, một callback được khai báo là *action*. Có thể có nhiều đối tượng callback trong *actionSets*, mỗi đối tượng phục vụ một mục trong menu hay các nút bấm.

Thông thường, trong các khai báo của các điểm mở rộng thường bao gồm một thuộc tính để xác định tên của class callback. Ví dụ với plug-in *help* UI, thuộc tính của class *action* là “*class*” như được chỉ ra ở (4) Hình 2.4.

Nhưng khi plug-in mở rộng xác định các đối tượng callback cần thiết, và khai báo các class của nó, các đối tượng này lúc này chỉ tồn tại là kết quả của hành động cụ thể của plug-in chính. Ví dụ ở (4), class callback *action* cho menu *Search→Help* được xác định bởi plug-in mở rộng *org.eclipse.help.ui* sẽ là *OpenHelpSearchPageAction* (package *org.eclipse.help.ui.internal*). Nhưng giá trị callback tương ứng được tạo bởi plug-in chính *org.eclipse.ui.* Trong trường hợp này, giá trị của callback sẽ được tạo lần đầu tiên khi mục menu *Search→Help* được gọi.

**Đối tượng dịch vụ không chi tiết**

Lưu ý rằng không phải tất cả các thành phần XML dùng để định nghĩa một mở rộng đều tương ứng với một vai trò callback. Một số thành phần có thể được mô tả thuần, cung cấp một số tham số cho plug-in để hình thành các UI tương ứng, hoặc để tạo ra các đối tượng bên trong như là một phần của mở rộng. Trong ví dụ trên, thành phần *actionSets* bản thân nó không định nghĩa đối tượng callback. Nhưng thành phần gây ra một đối tượng bên trong được hình thành trong plug-in *org.eclipse.ui* lại biểu diễn cho *actionSets* đó.

#### Mối quan hệ giữa Plug-in và Extension

Một hoạt động mở rộng là một khái niệm chung trong Eclipse và để hiểu rõ một cách khái quát, ta có thể tóm tắt lại mối quan hệ có thể tồn tại giữa đối tượng plug-in, extension-points, và đối tượng callback.

Có thể có nhiều điểm mở rộng có thể tồn tại trong một plug-in chính

Một plug-in có thể vừa là một plug-in chính, chỉ ra các điểm mở rộng, vừa là một plug-in mở rộng, mở rộng các plug-in khác.

Nhiều plug-in có thể mở rộng một điểm mở rộng cho trước

Một plug-in cho trước có thể một điểm mở rộng cho trước nhiều lần

Một plug-in mở rộng có thể bao gồm nhiều mở rộng của các plug-in chính khác nhau.

Một hành động mở rộng của một điểm mở rộng bởi một plug-in cụ thể có thể tạo ra nhiều đối tượng callback.

Một plug-in có thể xác định mở rộng thông qua điểm mở rộng của riêng nó.

#### Lược đồ extension-points

Một mở rộng cụ thể được xác định bởi một thành phần cấu hình XML trong plug-in mở rộng. Các thành phần này cung cấp các thông tin cần thiết để khởi tạo các đối tượng callback cần thiết cho phần mở rộng, cũng như các thông tin cần thiết để tùy chỉnh giao diện và hành vi của các plug-in chính. Khi một điểm mở rộng được tạo ra, ngoài việc khai báo trong file manifest của nó, cũng cần xác định cú pháp cấu hình cho phần mở rộng cho điểm mở rộng đó. Cú pháp này là một lược đồ XML và lưu trữ trong một tập tin có đuôi *.exsd*, ví dụ, *actionSets.exsd*. Các định nghĩa lược đồ này sau đó được tài liệu hóa của plug-in.

Dưới đây là một bản giản lược của file *actionSets.exsd:*

<schema targetNamespace="org.eclipse.ui">

<element name="extension">

<complexType>

<sequence>

<element ref="actionSet" minOccurs="1" maxOccurs="unbounded"/>

</sequence>

<attribute name="point" type="string" use="required"> </attribute>

<attribute name="id" type="string"> </attribute>

<attribute name="name" type="string"> </attribute>

</complexType>

</element>

<element name="actionSet">

<complexType>

<sequence>

<element ref="menu" minOccurs="0" maxOccurs="unbounded"/>

<element ref="action" minOccurs="0" maxOccurs="unbounded"/>

</sequence>

<attribute name="id" type="string" use="required"> </attribute>

<attribute name="label" type="string" use="required"> </attribute>

<attribute name="visible" type="boolean"> </attribute>

<attribute name="description" type="string"> </attribute>

</complexType>

</element>

<element name="action">

<complexType>

<choice>

<element ref="selection" minOccurs="0" maxOccurs="unbounded"/>

<element ref="enablement" minOccurs="0" maxOccurs="1"/>

</choice>

<attribute name="id" type="string" use="required"> </attribute>

<attribute name="label" type="string" use="required"> </attribute>

<attribute name="toolbarPath" type="string">

<attribute name="icon" type="string"> </attribute>

<attribute name="tooltip" type="string"> </attribute>

<attribute name="class" type="string"> </attribute>

</complexType>

</element>

</schema>

Định nghĩa một lược đồ extension-points

Lược đồ định nghĩa điểm mở rộng *actionSets* như là một thành phần gồm các thuộc tính và một chuỗi *actionSet*, và một *actionSet* bao gồm một số thuộc tính và chuỗi *menu* và *actions*.

## Tiểu kết chương 2

Kiến trúc nền tảng Eclipse bao gồm nhiều thành phần. Các chức năng của thành phần lõi tách biệt với các chức năng của phần giao diện. Eclipse cung cấp các công cụ cùng cơ chế sử dụng và quy định để có thể tích hợp các công cụ một cách dễ dàng. Những cơ chế này được thể hiện thông qua giao diện API, lớp và phương thức. Ngoài ra, nền tảng Eclipse còn cung cấp các framework để dễ dàng phát triển các công cụ mới.

Eclipse SDK (Software development kit) gồm Eclipse Platform, JDT và PDE. JDT và PDE là các plug-in được gắn vào Platform. Eclipse Platform được xây dựng dựa trên cơ chế “*phát hiện, tích hợp và chạy*”. Plug-in là đơn vị chức năng nhỏ nhất có thể được phát triển và phân phối một cách riêng biệt. Thông thường, công cụ nhỏ được viết trong một plug-in riêng biệt và công cụ phức tạp được tách ra thành nhiều plug-in.

Từ kiến trúc của Eclipse có thể dễ dàng thấy các ưu điểm của nó :

Có khả năng mở rộng, phụ thuộc vào các thành phần gắn thêm như cho ngôn ngữ mới, cho bộ xử lý mới

Ứng dụng được cho việc phát triển mọi kiểu ứng dụng, từ ứng dụng trong doanh nghiệp, ứng dụng trên máy tính cá nhân cho đến các ứng dụng nhúng cho các thiết bị

Mọi người có thể tự làm thêm các thành phần gắn thêm theo yêu cầu riêng của mình

Tuy nhiên, kiến trúc này cũng có nhược điểm là người phát triển cần phải hiểu biết về nó để biết lúc nào thì gắn thêm (hay gỡ ra) và gắn thêm cái gì. Ngoài ra, các thành phần gắn thêm (các dự án) không ngừng phát triển nên phải biết lúc nào thì nâng cấp lên đời mới hơn (mặc dù Eclipse cũng có khả năng tự động tìm thành phần gắn thêm).

# Chương 3. Xây dựng ứng dụng trên nền tảng Eclipse



## Bài toán điều phối các lời gọi dịch vụ trong kiến trúc SOA

### Mục tiêu

Nền tảng Eclipse cho phép dễ kết hợp các thành phần để tạo ra các giao diện người sử dụng nâng cao. Mặt khác các Web Services dựa trên java cung cấp một nền tảng rất tốt để phát triển khung ứng dụng quy trình nghiệp vụ theo định hướng dịch vụ . Cả hai khía cạnh trên, giao diện người sử dụng dựa trên plug-in và quy trình nghiệp vụ theo định hướng dịch vụ (service-oriented business processes) là không thể thiếu cho sự thành công của một sản phẩm phần mềm doanh nghiệp mạnh mẽ và tích hợp đầy đủ. Chính vì vậy, trước hết, giải pháp này tập trung vào mở rộng nền tảng Eclipse cho phép phát triển và triển khai các Web Services như các plug-in khác trong Eclipse. Do đó, các dịch vụ được tích hợp trong một kịch bản định hướng đường ống để giải quyết các kịch bản nghiệp vụ cụ thể - dẫn tới việc ta xây dựng một kiến trúc hướng dịch vụ theo đường ống - Service-oriented Pipeline Architecture (SOPA).

### Giải pháp

Vấn đề đặt ra là Java và đặc biệt là các lập trình viên Eclipse rất thích sử dụng plug-in và cơ chế mở rộng của Eclipse để phát triển các ứng dụng giao diện người sử dụng nâng cao. Như là cơ chế plug-n-play cho việc phát triển và triển khai của Web Services hoàn toàn có thể làm phong phú thêm sự linh hoạt của các ứng dụng nghiệp vụ hướng dịch vụ.

Giải pháp là xây dựng một dịch vụ web đăng ký (Web Services registry), gọi là bus dịch vụ (Services Bus), phát triển như một plug-in Eclipse. Services Bus cung cấp các điểm mở rộng cho các nhà phát triển để xuất bản các lớp Java tiêu chuẩn của họ như các Web Services. Cơ chế điểm mở rộng của Eclipse tạo điều kiện thuận lợi cho việc cấu hình các phần mở rộng với nhà cung cấp phần mở rộng. Như vậy, Services Bus sẽ tải tất cả phần mở rộng đã được kết nối với nhau và tự động triển khai chúng trong khi ứng dụng khởi động trên máy chủ Jetty và Apache AXIS. Vì vậy các lập trình viên có thể trong cùng một khoảng thời gian, hưởng lợi thế từ môi trường hỗ trợ người sử dụng nâng cao của Eclipse và Java Web Services trong cơ chế tương tự và chặt chẽ. Ngoài ra, Services Bus cũng sử dụng tiêu chuẩn WSDD và WSDL cho việc cấu hình dịch vụ.

Một phần nữa trong giải pháp là triển khai một plug-in đường ống (Pipeline plug-in), đóng vai trò trung tâm trong việc điều phối dịch vụ của hệ thống và trong việc tạo ra các dịch vụ nghiệp vụ mới. Pipeline được định nghĩa bằng một cấu trúc XML để quy định các bước trong Pipeline và các chuyển đổi liên quan. Nó bao gồm một số câu lệnh điều kiện kiểu XSLT và trao đổi dữ liệu bởi biểu thức XPATH. Pipeline được thống nhất truy cập qua Web Services plug-in, quản lý toàn bộ hệ thống dịch vụ bao gồm plug-in services, pipelines và các dịch vụ web bên ngoài (external Web Services). Web Services plug-in sẽ quản lý ngữ nghĩa của pipeline giống như tất các các dịch vụ khác.

## Mô tả chi tiết

### Kiến trúc hướng dịch vụ theo đường ống

Services và pipelines là dựa trên các thành phần của kiến trúc hướng dịch vụ theo đường ống – Service-oriented Pipeline Architecture (SOPA). Các dịch vụ có thể bao gồm từ giao diện người sử dụng dịch vụ tới các thành phần quy trình nghiệp vụ (Web Services). Mặt khác, pipelines mô tả các thành phần của các dịch vụ để thực hiện các nhiệm vụ cụ thể. Chính thức thì SOPA={S,P}, nơi mà các dịch vụ S có thể là các giao diện người sử dụng dịch vụ (GUI services) Sv , web-services nội bộ Sw , và web-services từ bên ngoài Sx. Tức là S ={ Sv , Sw , Sx}. Pipelines P sẽ điều phối các dịch vụ nghiệp vụ khác nhau (Sw và Sx) và áp dụng các phép chuyển đổi T để trả lại kết quả cho người sử dụng hoặc dịch vụ khác, tức là P ={Sw , Sx ,T}

### Services Bus

Phần cốt lõi của khung ứng dụng Eclipse là plug-in và cơ chế mở rộng của nó. Một plug-in có thể coi là điểm mở rộng để các plug-in khác có thể kết nối đến. Eclipse cũng cung cấp cơ chế để quản lý tập hợp các điểm mở rộng cơ bản và các đăng ký tương ứng của nó. Ví dụ như ViewsRegistry trong Views. Mặc dù có một số plug-in của hãng thứ ba được phát triển cho Web Services và triển khai trên các máy chủ đã được cấu hình, tuy nhiên Eclipse vẫn chưa hỗ trợ cơ chế plug-n-play cho Web Services như đối với Views. Cơ chế này có thể làm quá trình phát triển Web Services trở nên đơn giản hơn bằng cách chuyển phần khó khăn của việc triển khai dịch vụ của lập trình viên sang cho phần quản lý các thành phần dịch vụ. Do đó, các lập trình viên chỉ cần phải tập trung vào việc phát triển các tính năng của dịch vụ thay vì phải quan tâm đến chi tiết vận hành của nó.

Services Bus cung cấp các điểm mở rộng cho các nhà phát triển để xuất các lớp Java tiêu chuẩn của họ như các dịch vụ web. Cơ chế điểm mở rộng của Eclipse tạo điều kiện thuận lợi cho việc cấu hình các phần mở rộng với nhà cung cấp phần mở rộng. Khi khởi chạy ứng dụng, services bus sẽ tải tất cả các dịch vụ đã được kết nối với nhau và tự động triển khai chúng bằng cách sử dụng máy chủ nhúng Jetty và Apache AXIS. Các script triển khai được tạo ra từ các mô tả dịch vụ. Ngoài ra, Services Bus cũng sử dụng tiêu chuẩn WSDD và WSDL để cấu hình dịch vụ.

### Plug-n-Play Web Services

Điểm cơ bản trong phát triển Service Bus chính là đạt được mục tiêu của plug-n-play Web Services sử dụng các plug-in và cơ chế mở rộng của nền tảng Eclipse. Đầu tiên, một điểm mở rộng phải được cấu hình theo các đặc tả dịch vụ và các tiêu chuẩn triển khai như WSDL và WSDD.

|  |
| --- |
| <schema targetNamespace="org.nhan.services">  <element name="**service**">  <complexType>  <sequence>  <element ref="operation" minOccurs="1" MaxOccurs="unbounded"/>  </sequence>  <attribute name="name" type="string" use="required">  </attribute>  <attribute name="class" type="string" use="required">  <annotation>  <appInfo>  <meta.attribute kind="java"/>  </appInfo>  </annotation>  </attribute>  </complexType>  </element>  <element name="**operation**">  <complexType>  <sequence>  <element ref="parameter" minOccurs="1" maxOccurs="unbounded"/>  </sequence>  <attribute name="name" type="string" use="required">  </attribute>  <attribute name="returns" use="required">  </attribute>  </complexType>  </element>  <element name="**parameter**">  <complexType>  <attribute name="type" use="required">  </attribute>  <attribute name="name" type="string">  </complexType>  </element>  </schema> |

Phiên bản tóm tắt của lược đồ điểm mở rộng

Điều quan trọng nhất của Service Bus đối với điểm mở rộng này là các lập trình viên phải xuất ra các class Java tiêu chuẩn như các dịch vụ. Điều này sẽ làm quá trình lập trình và bảo dưỡng Web Services thuận tiện hơn đối với các plug-in Eclipse khác. Ngoài ra, Services Bus đọc các chi tiết cấu hình của các dịch vụ được kết nối trong quá trình khởi động. Nó sẽ tự động tạo ra các script dựa trên WSDD và sử dụng máy chủ Jetty nhúng và Apache AXIS để hoàn thành nhiệm vụ.

|  |
| --- |
| <extension point="org.nhan.services.services">  <**service** name="org.example.arithmatics" **class**="org.example.Arithmatics" >  <**operation** name="multiply" returnType="xsd:double">  <parameter name="first" type="xsd:double"/>  <parameter name="second" type="xsd:double"/>  </operation>  </service>  <extension> |

Mô tả dịch vụ như phần mở rộng

### Tính trong suốt của lời gọi dịch vụ

Service Bus có thể được xem như cánh cửa để đi vào hệ thống SOPA dựa trên Eclipse. Nó có chức năng định tuyến các yêu cầu dịch vụ đến các dịch vụ được kết nối hiện thời. Một chức năng khác quan trọng của Service Bus đó là nâng cao tính linh hoạt của hệ thống SOPA. Từ đó, nó cung cấp một lớp truy cập đồng nhất và trong suốt đến các dịch vụ bên trong và dịch vụ bên ngoài hệ thống.

Lời gọi dịch vụ (Service call): Các dịch vụ cắm trong Service Bus có thể được gọi bởi các class tương ứng hoặc sử dụng máy chủ Apache AXIS. Cả hai cách này đều sử dụng các cách đặt tên truyền thống cho các class và các hàm, và trả về kết quả giống nhau. Ngoài ra, đối với các dịch vụ bên trong chỉ cần gọi tên là đủ, tuy nhiên các dịch vụ bên ngoài còn yêu cầu cung cấp đầy đủ end-point URI.

|  |
| --- |
| Object[] params = ...  Call client = new Call("org.example.arithmatics");  Object result = client.invoke("multiply", params); |

Gọi một dịch vụ được cắm vào trong Services Bus

### Dịch vụ đường ống – Services Pipeline

Một pipeline trong thuật ngữ SOPA là một tập hợp có tên duy nhất các lời gọi service và chuyển đổi trung gian. Pipeline plug-in cho phép hệ thống SOPA hiểu rõ kịch bản dựa trên dịch vụ cơ bản và các pipelines. Ý tưởng dựa trên các thành phần đường ống (component pipeline), mỗi thành phần trong pipelines chỉ định một hoạt động cụ thể, các thành phần móc nối với nhau vào pipelines mà không cần yêu cầu lập trình.

Các pipeline và cấu trúc tương ứng của chứng được định nghĩa bằng cách sử dụng một cấu trúc XML để chỉ rõ các thành phần pipeline và các chuyển đổi liên quan.

|  |
| --- |
| 1. < pipeline name="square"> 2. < parameters> 3. < parameter name="num" type="xsd:double"/> 4. </parameters> 5. < call service="org.example.arithmatics" operation="multiply"/> 6. < parameter>{num}</parameter> 7. < parameter>{num}</parameter> 8. </call> 9. < transform method="xml" stylesheet="result.xsl"/> 10. </pipeline> |

Cấu trúc cơ bản của một pipeline đơn giản

Dòng 1 ta định nghĩa tên của một pipeline. Mỗi pipleline có thể nhận được một số tham số đầu vào, có thể sử dụng ở bất cứ đâu trong phạm vi của pipeline đó. Dòng 2-4 cho thấy phần tham số và định nghĩa của một tham số gọi là “num”. Phần thú vị và khác biệt nhất trong giải pháp là cách gọi dịch vụ. Tại dòng 5 là một phương thức “multiply” của dịch vụ “org.example.arithmatics” là được yêu cầu. Phương thức được gọi có thể sử dụng tham số của pipeline. Điều quan trọng được đề cập là các dịch vụ trong một hệ thống SOPA là không chỉ giới hạn cho những người cung cấp dịch vụ bởi các plug-in khác mà còn bao gồm các pipeline và các dịch vụ web bên ngoài (phân biệt bởi các end-point URI).

Các kết quả trả về bởi các dịch vụ có thể được chuyển đổi trong thời gian thực hiện của một pipeline. Tính năng này cho phép các kết quả được biến đổi và chuyển đổi sang định dạng cần thiết. Việc chuyển đổi được thực hiện bằng cách áp dụng một biến đổi XSLT để cho ra kết quả hiện tại ở pipeline. Pipeline plug-in giữ kết quả bên trong và cuối cùng ở giai đoạn tuần tự các kết quả được xuất ra theo định dạng yêu cầu. Những định dạng được hỗ trợ xuất bản là TEXT, XML, HTML và XSWT.

Các dịch vụ có sẵn trong môi trường SOPA được định tuyến thông qua Services Bus plug-in, tức là tất cả dịch vụ sẽ được yêu cầu từ Services Bus, nơi chịu trách nhiệm cho việc tìm thấy và triệu gọi các dịch vụ tương ứng để làm nhiệm vụ. Tính năng này cung cấp sự trong suốt của dịch vụ trong toàn bộ môi trường SOPA. Các dịch vụ trong SOPA là không giới hạn để các dịch vụ tiếp xúc với nhau nhưng có thể tùy chọn bao gồm các pipeline, các Plug-in services và các dịch vụ web từ bên ngoài (External Web Services). Kết quả là hệ thống SOPA mang đến cho kịch bản tích hợp dịch vụ một chân trời mới. Các kịch bản nghiệp vụ phát triển theo khung lập trình Eclipse có thể kết hợp các thành phần từ nội bộ hoặc bên ngoài thông qua một dịch vụ định tuyến (Services Bus plug-in).

Plug-in Services

Services

Services

Pipelines



External

Web Services

Web Services

Services Bus

Plug-in



Tính trong suốt của dịch vụ trong SOPA

Các pipeline được tạo ra có thể sử dụng bởi các thành phần khác của hệ thống và cung cấp nhiều loại dịch vụ bao gồm nghiệp vụ logic, các tính năng trực quan hoặc kết hợp cả hai. Một pipeline chứa các lời gọi đến các dịch vụ và thu thập kết quả để tạo thành thông tin, được lưu trữ và tái sử dụng như một dịch vụ mới. Biểu diễn hình ảnh và định dạng kết quả cũng là một trong những chức năng của pipeline. Kết quả là tập các kết quả có thể được biểu diễn khác nhau dựa trên ngữ cảnh và yêu cầu của người sử dụng.

### Tính năng kỹ thuật và các loại kịch bản của Pipeline

Đầu tiên ta xét một pipeline đơn giản để bình phương một số. Ví dụ này chỉ ra các thành phần cơ bản của một pipeline. Tiếp theo chúng ta sẽ xét ví dụ đối với một dịch vụ tính toán đơn giản, để kiểm tra tính năng của pipeline có thể hoạt động đối với phần lớn các loại dữ liệu XSD và bất kỳ đối tượng Java được trả về từ phương thức của dịch vụ.

#### Đa lời gọi

Một pipeline có thể có nhiều lời gọi đến các dịch vụ SOPA. Kết quả trả về có thể kết hợp với nhau để tạo ra tập các kết quả cho từng loại dịch vụ. Theo ví dụ dưới đây, phương thức “add” và “multiply” của dịch vụ “org.example.arithmatics” là được gọi.

|  |
| --- |
| 1. < pipeline name="combined" serialization="xml"> 2. <parameters> 3. < parameter name="first" type="xsd:double"/> 4. < parameter name="second" type="xsd:double"/> 5. </parameters> 6. < call id="firstCall" service="org.example.arithmatics" operation="add"/> 7. < parameter>{first}</parameter> 8. < parameter>{second}</parameter> 9. </call> 10. < call id="secondCall" service="org.example.arithmatics" operation="multiply"/> 11. < parameter>{first}</parameter> 12. < parameter>{second}</parameter> 13. </call> 14. </pipeline> |

Với các tham số đầu vào là 5 và 6, kết quả sau khi xử lý sẽ như bên dưới:

|  |
| --- |
| 1. <result> 2. <firstCall> 3. 11 4. </firstCall> 5. <secondCall> 6. 30 7. </secondCall> 8. </result> |

Các tên node kết quả được lấy từ thuộc tính ID của lời gọi trong Pipeline

#### Lời gọi lồng nhau

Cũng có thể sử dụng những lời gọi lồng trong SOPA; ví dụ, các dịch vụ có thể liên kết với nhau để trao đổi tham số và kết quả. Ví dụ tiếp theo để minh họa cho quá trình này.

|  |
| --- |
| 1. < pipeline name="nested" serialization="xml"> 2. < parameters> 3. < parameter name="input" type="xsd:double"/> 4. </parameters> 5. < call id="parent" service="org.example.arithmatics" operation="multiply"> 6. < parameter type="xsd:double">2</parameter> 7. < parameter> 8. < call service="org.example.arithmatics" operation="add"> 9. < parameter type="xsd:double">15</parameter> 10. < parameter>{input}</parameter> 11. </call> 12. </parameter> 13. </call> 14. </pipeline> |

Theo ví dụ trên, lời gọi dịch vụ đầu tiên tại dòng 5 nhận hai tham số đầu vào, tham số đầu tiên là giá trị 5 và tham số thứ hai là đầu ra của một lời gọi khác. Kết quả của quá trình này với tham số đầu vào là 5 sẽ là:

|  |
| --- |
| 1. <result> 2. <parent> 3. 40 4. </parent> 5. </result> |

#### Lời gọi có điều kiện

Pipeline plug-in cung cấp các loại thẻ cho phép gọi đến các dịch vụ dựa trên điều kiện kiểu XPATH. Để thực hiện điều này, có hai cầu trúc điều kiện được sử dụng là “if” và “choose”. Ở cả hai cấu trúc này, điều kiện dạng XPATH sẽ được kiểm tra và dựa trên kết quả của dịch vụ tương ứng được gọi. Ví dụ bên dưới sử dụng cấu trúc “if” với điều kiện “{input} > 0”.

|  |
| --- |
| 1. <pipeline name="nested" serialization="xml"> 2. <parameters> 3. <parameter name="input" type="xsd:double"/> 4. </parameters> 5. <xsl:if test="{input} > 0"> 6. < call id="parent" service="org.example.arithmatics" operation="multiply"> 7. < parameter>2</parameter> 8. < parameter>{input}</parameter> 9. </call> 10. </xsl:if> 11. </pipeline> |

Một ví dụ khác sử dụng cấu trúc “choose” như sau:

|  |
| --- |
| 1. <pipeline name="nested" serialization="xml"> 2. <parameters> 3. <parameter name="input" type="xsd:double"/> 4. </parameters> 5. < xsl:choose> 6. <xsl:when test="{input} > 0"> 7. <call id="parent" service="org.example.arithmatics" operation="multiply"> 8. <parameter>2</parameter> 9. <parameter>{input}</parameter> 10. </call> 11. </xsl:when> 12. <xsl:otherwise> 13. <call id="parent" service="org.example.arithmatics" operation="multiply"> 14. <parameter>2</parameter> 15. <parameter>10</parameter> 16. </call> 17. </xsl:otherwise> 18. </xsl:choose> 19. </pipeline> |

#### Các tham số Xpath trích xuất

Hiện tại chúng ta đã giới thiệu ba loại tham số cho các lời gọi dịch vụ:

Literal (dạng trực tiếp): truyền các giá trị tham số trực tiếp vào dịch vụ

Pipeline parameter: các giá trị tham số được thay bằng các giá trị tham số pipeline trong quá trình chạy

Service call parameter: kết quả của dịch vụ này là đầu vào cho dịch vụ khác

Bây giờ chúng ta sẽ giới thiệu một loại tham số mới. Tham số này sẽ trích giá trị của nó bằng cách áp dụng các lệnh XPATH vào kết quả XML. Ví dụ bên dưới chỉ ra cách hoạt động:

|  |
| --- |
| 1. < call id="parent" service="org.example.arithmatics" operation="multiply"> 2. < parameter>{xpath:/result/previous}</parameter> 3. < parameter>{input}</parameter> 4. </call> |

#### Pipes trong pipes

Một ưu điểm của việc kết hợp các dịch vụ thành pipeline là việc tạo ra dịch vụ phức dựa trên các dịch vụ sẵn có. Các pipeline này có thể được gọi và tái sử dụng bởi các pipeline khác hoặc các thành phần SOPA thông qua tên gọi. Ví dụ bên dưới chỉ ra cách tái sử dụng pipeline. Ở dòng thứ 4, một lời gọi đến “square” pipeline và kết quả được truyền vào tham số thứ hai của dịch vụ “multiply”.

|  |
| --- |
| 1. <call id="parent" service="org.example.arithmatics" operation="multiply"> 2. <parameter>5</parameter> 3. <parameter> 4. <call id="parent" service="org.nhan.pipeline" operation="square"> 5. <parameter>6</parameter> 6. </call> 7. </parameter> 8. </call> |

<Ghi chú>: Do giới hạn về số trang, nên các listing và hình chụp ứng dụng được trình bày trong Phụ lục.

## tiểu kết chương 3

Việc xây dựng kiến trúc hướng dịch vụ theo đường ống – SOPA đã giúp các dịch vụ có thể giao tiếp với nhau, tích hợp lại thành các quy trình nghiệp vụ mới đáp ứng yêu cầu của người sử dụng. Đặc biệt là nó đã hỗ trợ cho các lập trình viên thuận tiện hơn trong việc sử dụng và khai báo các tập dịch vụ có sẵn để hình thành nên các quy trình nghiệp vụ mới phục vụ cho lợi ích của các doanh nghiệp. Một vấn đề phát triển và nghiên cứu quan trọng trong việc xây dựng plug-in đó là làm đơn giản quá trình tạo pipeline, dựa trên các yêu cầu đến và các dịch vụ hiện có để hình thành nên dịch vụ mới.

# KẾT LUẬN VÀ KIẾN NGHỊ

Hướng tiếp cận SOA là một trong những lựa chọn tối ưu cho các doanh nghiệp hiện nay trong việc chuẩn hóa lại kiến trúc hệ thống. Và để triển khai kiến trúc hướng dịch vụ, công nghệ Web Services là lựa chọn lý tưởng bởi khả năng đáp ứng mềm dẻo và linh hoạt của nó. Trong cuốn luận văn này, tôi đã trình bày những khái niệm tổng quan về công nghệ Web Services, những ưu điểm của công nghệ này trong việc giải quyết bài toán tích hợp giữa các hệ thống, cũng như khả năng ứng dụng của nó trong việc xây dựng mô hình kiến trúc hướng dịch vụ. Bài luận văn cũng đã xây dựng được một hệ thống plug-in dựa trên công nghệ Eclipse để có thể triển khai lập trình ứng dụng theo kiến trúc hướng dịch vụ.

Hướng nghiên cứu tiếp theo của luận văn là tiếp tục hoàn thiện hơn nữa hệ thống plug-in, thử nghiệm với nhiều Web Services phức tạp hơn nữa, đồng thời nghiên cứu tăng cường khả năng bảo mật.

# TÀI LIỆU THAM KHẢO

**Tiếng Việt**

1. Nguyễn Trọng Dũng, *Giáo trình Java – Chương 3:* *Phát triển Web Services với các công nghệ chuẩn của java* , Trường ĐHSP Hà Nội.
2. Hồ Bảo Thanh, Nguyễn Hoàng Long (2005), *Nghiên cứu kiến trúc hướng dịch vụ (Service-Oriented Architecture) và ứng dụng,* Luận văn cử nhân tin học khoa CNTT – ĐH KHTN Tp.HCM.

**Tiếng Anh**

1. Abdaldhem Albreshne, Patrik Fuhrer, Jacque Pasquier Spasquier (2009), *Web Services Technologie: State of the Art,* Definitions, Standards, Case Study, pp. 1-22.
2. Azad Bolour (2003), “Notes on the Eclipse Plug-in Architecture”*,* Bolour Computing, <http://www.eclipse.org/articles/Article-Plug-in-architecture/plugin_architecture.html>, (Ngày truy cập 4/4/2016).
3. Dr Alex Blewitt (2013), *Eclipse 4 Plug-in Development by Example*, Packt Publishing, pp. 7-34.
4. [Bertrand Portier](http://www.ibm.com/developerworks/webservices/library/ws-soa-term1/index.html#authorN1001E) (2007), “SOA terminology overview”, IT Architect IBM, <http://www.ibm.com/developerworks/library/ws-soa-term1/> (truy cập ngày 4/4/2016).
5. David Gallardo (2002), “Developing Eclipse plug-ins”, IBM <http://www.ibm.com/developerworks/opensource/library/os-ecplug/> (truy cập ngày 4/4/2016).
6. Eric Clayberg, Dan Rubel (2009), *Eclipse Plug-ins*, Third Edition, pp. 21-153, 657-680.
7. Hartwig Gunzer (2002), *Introduction to Web Services*, Sales Engineer, Borland.
8. Kiet T. Tran, bookcoon.com (2013), *Introduction to Web Services with Java*, 1st edition, pp. 15-115.
9. Mark D. Hansen (2007), *SOA Using Java Web Services*, Printed in the United States of America.
10. Martin Kalin (2013), *Java Web Services : Up and Running*, Second Edition, O’Reilly Media, pp. 1-9, 291-329.
11. Yuli Vasiliev (2007), *SOA and WS-BPEL*, Packt publishing, pp. 5-37.
12. Chris Aniszczyk, David Gallardo, “Get Started with the Eclipse platform”, <http://www.ibm.com/developerworks/opensource/library/os-eclipse-platform/#N101FF>
13. OASIS (2007), “Web Services Business Process Execution Language Version 2.0”, <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html>

# PHỤ LỤC

Phần xây dựng Plug-in được kết hợp từ hai plug-in là:

Plug-in org.nhan.sbus: đây là plug-in hỗ trợ việc khai báo và triển từ các services nội bộ và các services từ bên ngoài với cơ chế plug-n-play.

Plug-in org.nhan.pipeline: đây là plug-in chính, được xây dựng cùng với sự kết hợp của plug-in org.nhan.sbus, để thiết lập các pipeline cho việc tích hợp và triển khai các quy trình nghiệp vụ đáp ứng một yêu cầu cụ thể.

**Plug-in org.nhan.sbus**

Cấu hình Extension Points trong Eclipse



Với nội dung các Extension Points được mô tả bởi một shema “.exsd”

File “services.exsd” – File này mô tả những dịch vụ từ nội bộ được cắm vào theo cơ chế “plug-n-play”.

|  |
| --- |
| <?xml version='1.0' encoding='UTF-8'?>  <!-- Schema file written by PDE -->  <schema targetNamespace="org.nhan.services">  <annotation>  <appInfo>  <meta.schema plugin="org.nhan.services" id="services" name="services"/>  </appInfo>  <documentation>  [Enter description of this extension point.]  </documentation>  </annotation>  <element name="extension">  <complexType>  <sequence>  <element ref="service" minOccurs="1" maxOccurs="unbounded"/>  </sequence>  <attribute name="point" type="string" use="required">  <annotation>  <documentation>  </documentation>  </annotation>  </attribute>  <attribute name="id" type="string">  <annotation>  <documentation>  </documentation>  </annotation>  </attribute>  <attribute name="name" type="string">  <annotation>  <documentation>  </documentation>  <appInfo>  <meta.attribute translatable="true"/>  </appInfo>  </annotation>  </attribute>  </complexType>  </element>  <element name="service">  <complexType>  <sequence>  <element ref="operation" minOccurs="1" maxOccurs="unbounded"/>  </sequence>  <attribute name="name" type="string" use="required">  <annotation>  <documentation>  </documentation>  </annotation>  </attribute>  <attribute name="class" type="string" use="required">  <annotation>  <documentation>  </documentation>  <appInfo>  <meta.attribute kind="java"/>  </appInfo>  </annotation>  </attribute>  <attribute name="namespace" type="string">  <annotation>  <documentation>  </documentation>  </annotation>  </attribute>  </complexType>  </element>  <element name="operation">  <complexType>  <sequence>  <element ref="parameter" minOccurs="1" maxOccurs="unbounded"/>  </sequence>  <attribute name="name" type="string" use="required">  <annotation>  <documentation>  </documentation>  </annotation>  </attribute>  <attribute name="returns" type="string" use="required">  <annotation>  <documentation>  </documentation>  </annotation>  </attribute>  </complexType>  </element>  <element name="parameter">  <complexType>  <attribute name="name" type="string" use="required">  <annotation>  <documentation>  </documentation>  </annotation>  </attribute>  <attribute name="type" type="string" use="required">  <annotation>  <documentation>  </documentation>  </annotation>  </attribute>  </complexType>  </element>  <annotation>  <appInfo>  <meta.section type="since"/>  </appInfo>  <documentation>  [Enter the first release in which this extension point appears.]  </documentation>  </annotation>  <annotation>  <appInfo>  <meta.section type="examples"/>  </appInfo>  <documentation>  [Enter extension point usage example here.]  </documentation>  </annotation>  <annotation>  <appInfo>  <meta.section type="apiInfo"/>  </appInfo>  <documentation>  [Enter API information here.]  </documentation>  </annotation>  <annotation>  <appInfo>  <meta.section type="implementation"/>  </appInfo>  <documentation>  [Enter information about supplied implementation of this extension point.]  </documentation>  </annotation>  <annotation>  <appInfo>  <meta.section type="copyright"/>  </appInfo>  <documentation>  </documentation>  </annotation>  </schema> |

File “webservices.exsd” – File này mô tả những dịch vụ từ bên ngoài được liên kết, tích hợp vào.

|  |
| --- |
| <?xml version='1.0' encoding='UTF-8'?>  <!-- Schema file written by PDE -->  <schema targetNamespace="org.nhan.services">  <annotation>  <appInfo>  <meta.schema plugin="org.nhan.services" id="webservices" name="Web Services"/>  </appInfo>  <documentation>  [Enter description of this extension point.]  </documentation>  </annotation>  <element name="extension">  <complexType>  <sequence>  <element ref="webservice" minOccurs="1" maxOccurs="unbounded"/>  </sequence>  <attribute name="point" type="string" use="required">  <annotation>  <documentation>  </documentation>  </annotation>  </attribute>  <attribute name="id" type="string">  <annotation>  <documentation>  </documentation>  </annotation>  </attribute>  <attribute name="name" type="string">  <annotation>  <documentation>  </documentation>  <appInfo>  <meta.attribute translatable="true"/>  </appInfo>  </annotation>  </attribute>  </complexType>  </element>  <element name="webservice">  <complexType>  <attribute name="name" type="string" use="required">  <annotation>  <documentation>  </documentation>  </annotation>  </attribute>  <attribute name="wsdd" type="string" use="required">  <annotation>  <documentation>  </documentation>  <appInfo>  <meta.attribute kind="resource"/>  </appInfo>  </annotation>  </attribute>  </complexType>  </element>  <annotation>  <appInfo>  <meta.section type="since"/>  </appInfo>  <documentation>  [Enter the first release in which this extension point appears.]  </documentation>  </annotation>  <annotation>  <appInfo>  <meta.section type="examples"/>  </appInfo>  <documentation>  [Enter extension point usage example here.]  </documentation>  </annotation>  <annotation>  <appInfo>  <meta.section type="apiInfo"/>  </appInfo>  <documentation>  [Enter API information here.]  </documentation>  </annotation>  <annotation>  <appInfo>  <meta.section type="implementation"/>  </appInfo>  <documentation>  [Enter information about supplied implementation of this extension point.]  </documentation>  </annotation>  <annotation>  <appInfo>  <meta.section type="copyright"/>  </appInfo>  <documentation>  </documentation>  </annotation>  </schema> |

Đối với plug-in này, ta tạo những lớp Java để hỗ trợ xử lý, dưới đây là những lớp Java xử lý chính của plug-in.

Class MessageHandler.java đây là lớp chính, quản lý plug-in

|  |
| --- |
| package org.nhan.services;  import java.util.HashMap;  import org.eclipse.core.runtime.IStatus;  import org.eclipse.core.runtime.Plugin;  import org.eclipse.core.runtime.Status;  import org.osgi.framework.BundleContext;  /\*\*  \* The main plugin class to be used in the desktop.  \*/  public class MessageHandler extends Plugin {  protected static final String PLUGIN\_ID = "org.nhan.services";  protected static final String EXT\_SERVLETS = PLUGIN\_ID + ".servlets";  protected static final String EXT\_WEBSERVICES = PLUGIN\_ID + ".webservices";  public static final String EXT\_SERVICES = PLUGIN\_ID + ".services";  protected static final int SERVER\_PORT = 8088;  // The shared instance.  private static MessageHandler plugin;  // Jetty HTTP server  private WebServer webServer;  // Loaded extensions  private HashMap<String, ServiceProxy> serviceCache;  /\*\*  \* The constructor.  \*/  public MessageHandler() {  plugin = this;  }  /\*\*  \* This method is called upon plug-in activation  \*/  public void start(BundleContext context) throws Exception {  super.start(context);  plugin=this;  startWebServer();  }  /\*\*  \* This method is called when the plug-in is stopped  \*/  public void stop(BundleContext context) throws Exception {  super.stop(context);  if (webServer != null && webServer.isStarted()) {  webServer.stop();  }  plugin = null;  }  /\*\*  \* Returns the shared instance.  \*/  public static MessageHandler getDefault() {  return plugin;  }  private void startWebServer() throws Exception {  new Thread() {  public void run() {  webServer = new WebServer(SERVER\_PORT); webServer.loadServlets(MessageHandler.EXT\_SERVLETS);  webServer.start(); webServer.loadWebServices(MessageHandler.EXT\_WEBSERVICES);  }  }.start();  }  protected static void log(String message) {  plugin.getLog().log(  new Status(IStatus.INFO, PLUGIN\_ID, 0, message, null));  }  /\*\*  \* Logs the error message  \*/  protected static void log(String message, Exception exp) {  plugin.getLog().log(  new Status(IStatus.ERROR, PLUGIN\_ID, 0, message, exp));  }  public static Object getService(String service) {  ServiceProxy proxy = getDefault().getServiceProxy(service);  return proxy.getDelegate();  }  protected ServiceProxy getServiceProxy(String service) {  if (serviceCache == null) {  serviceCache = ServiceProxy  .loadServices(MessageHandler.EXT\_SERVICES);  }  ServiceProxy proxy = serviceCache.get(service);  if (proxy == null)  throw new IllegalArgumentException("No such service: " + service);  return proxy;  }  protected Object invoke(String service, String operation, Object... params)  throws Exception {  ServiceProxy sproxy = getServiceProxy(service);  OperationProxy oproxy = sproxy.getOperation(operation);  long t1 = System.currentTimeMillis();  Object response = oproxy.invoke(params);  long t2 = System.currentTimeMillis();  log("Total time in service invokation " + (t2 - t1) + " ms");  return response;  }  } |

WebServer.class

|  |
| --- |
| package org.nhan.services;  import java.io.InputStream;  import javax.servlet.Servlet;  import org.apache.axis.client.AdminClient;  import org.apache.axis.transport.http.AdminServlet;  import org.apache.axis.transport.http.AxisServlet;  import org.apache.axis.utils.Options;  import org.eclipse.core.runtime.FileLocator;  import org.eclipse.core.runtime.IConfigurationElement;  import org.eclipse.core.runtime.Path;  import org.eclipse.core.runtime.Platform;  import org.mortbay.jetty.MimeTypes;  import org.mortbay.jetty.Server;  import org.mortbay.jetty.servlet.Context;  import org.mortbay.jetty.servlet.ServletHolder;  import org.mortbay.jetty.servlet.ServletMapping;  import org.osgi.framework.Bundle;  public class WebServer {  // Jetty HTTP server  private final Server server;  // Default servlet and application context  private final Context context;  // Client for Web Services deployment  private AdminClient adminClient;  public WebServer(final int port) {  server = new Server(port);  context = new Context(server, "/", Context.SESSIONS);  // Deploy Axis servlets  deploy("AxisServlet", new AxisServlet(), new String[] {  "/servlet/AxisServlet", "/services/\*" });  deploy("AdminServlet", new AdminServlet(),  new String[] { "/servlet/AdminServlet" });  MimeTypes mimeTypes = context.getMimeTypes();  if (mimeTypes == null) {  mimeTypes = new MimeTypes();  context.setMimeTypes(mimeTypes);  }  mimeTypes.addMimeMapping("xsd", "text/xml");  mimeTypes.addMimeMapping("wsdl", "text/xml");  adminClient = new AdminClient();  String[] args = { "-lhttp://localhost:" + port  + "/services/AdminService" };  try {  Options op = new Options(args);  adminClient.processOpts(op);  } catch (Exception exp) {  // This must not happen  MessageHandler.log("Can't instantiate deployment client", exp);  }  }  protected void loadServlets(String extensionPoint) {  IConfigurationElement[] extensions = ExtPointUtil  .getConfigElements(extensionPoint);  if (extensions == null || extensions.length < 1) {  MessageHandler.log("No attached servlet extensions");  return;  }  for (IConfigurationElement conElServlet : extensions) {  try {  Servlet servletClass = (javax.servlet.Servlet) conElServlet  .createExecutableExtension("class");  String servletName = conElServlet.getAttribute("name");  // Find mappings  IConfigurationElement[] conElMappings = conElServlet  .getChildren();  String[] mappings = new String[conElMappings.length];  for (int i = 0; i < mappings.length; i++) {  mappings[i] = conElMappings[i].getAttribute("pathSpec");  }  deploy(servletName, servletClass, mappings);  } catch (Exception exp) {  MessageHandler.log("Can't deploy servlet", exp);  }  }  }  protected void loadWebServices(String extensionPoint) {  IConfigurationElement[] extensions = ExtPointUtil  .getConfigElements(extensionPoint);  if (extensions == null || extensions.length < 1) {  MessageHandler.log("No attached Web Services");  return;  }  for (IConfigurationElement conElWebs : extensions) {  try {  String wsddPath = conElWebs.getAttribute("wsdd");  deploy(wsddPath);  } catch (Exception exp) {  MessageHandler.log("Can't deploy Web Services: " + conElWebs,exp);  }  }  }  public void deploy(String name, Servlet servlet, String[] pathSpecs) {  if (name == null || name.length() < 1) {  // Use servlet class name as servlet name in the servlet context  name = servlet.getClass().getName();  }  ServletHolder holder = new ServletHolder();  holder.setName(name);  holder.setServlet(servlet);  ServletMapping mapping = new ServletMapping();  mapping.setServletName(name);  mapping.setPathSpecs(pathSpecs);  context.addServlet(holder, null);  context.getServletHandler().addServletMapping(mapping);  }  public String deploy(InputStream wsdd) throws Exception {  // If server is not completely started give it some  // time to do the initialization.  while (server.isStarting()) {  Thread.sleep(1000);  }  return adminClient.process(wsdd);  }  public String deploy(String wsddPath) throws Exception {  // Translate this plugin relative path to Eclipse terminology  Path path0 = new Path("schema/feeds.wsdd");  // Now get the input stream  Bundle bundle = Platform.getBundle(MessageHandler.PLUGIN\_ID);  InputStream stream = FileLocator.openStream(bundle, path0, false);  // Finally deploy the Web Services from the wsdd content stream  return deploy(stream);  }  public boolean start() {  try {  server.start();  // server.join();  MessageHandler.log("Jetty HTTP server now running");  return true;  } catch (Exception exp) {  MessageHandler.log("Can't start Jetty", exp);  return false;  }  }  public boolean isStarted() {  return server.isStarted();  }  public boolean isRunning() {  return server.isRunning();  }  public boolean stop() {  try {  server.stop();  return true;  } catch (Exception exp) {  MessageHandler.log("Can't stop Jetty", exp);  return false;  }  }  } |

Call.java đây là class quan trọng để triệu gọi các services

|  |
| --- |
| package org.nhan.services;  import java.io.IOException;  import javax.xml.namespace.QName;  import javax.xml.rpc.ParameterMode;  import org.apache.axis.encoding.XMLType;  public class Call {  public Call(String service) {  int pos = service.indexOf("@");  if (pos > 0) {  String host = service.substring(pos + 1);  if (!(host.equalsIgnoreCase("LOCALHOST") || host  .equalsIgnoreCase("127.0.0.1")))  this.endpoint = "http://" + host  + ":8088/services/MessageHandler";  this.service = service.substring(0, pos);  } else {  this.service = service;  }  }  public String getOperation() {  return operation;  }  public void setOperation(String operation) {  this.operation = operation;  }  public Object invoke(String operation, Object... params) throws Exception {  setOperation(operation);  return invoke(params);  }  public void invokeAsyn(ICallback callback, String operation,  Object... params) {  setOperation(operation);  invokeAsyn(callback, params);  }  public Object invoke(Object... params) throws Exception {  if (service == null || operation == null)  throw new IllegalArgumentException(  "Service or operation name must NOT be null");  else  return MessageHandler.getDefault().invoke(service, operation,  params);  }  public void invokeAsyn(final ICallback callback, final Object... params) {  Runnable runner = new Runnable() {  public void run() {  try {  Object res = invoke(params);  callback.callback(res);  } catch (Exception exp) {  callback.callback(exp);  }  }  };  new Thread(runner).start();  }  public Object invoke(String operation, String... params) throws Exception {  throw new NoSuchMethodException("Method not supported anymore");  }  public Object invoke(String... params) throws Exception {  throw new NoSuchMethodException("Method not supported anymore");  }  @SuppressWarnings("unused")  private String invokeRemote(String params) throws IOException {  javax.xml.rpc.Call call = new org.apache.axis.client.Call(endpoint);  call.setOperationName(new QName("urn:cominfo", "invoke"));  call.addParameter("service", XMLType.XSD\_STRING, ParameterMode.IN);  call.addParameter("operation", XMLType.XSD\_STRING, ParameterMode.IN);  call.addParameter("param", XMLType.XSD\_STRING, ParameterMode.IN);  call.setReturnType(XMLType.XSD\_STRING);  return (String) call  .invoke(new Object[] { service, operation, params });  }  private String service;  private String operation;  private String endpoint;  } |

Ngoài các class quan trọng như trên, thì còn các class khác hỗ trợ được đính kèm trong luận văn.

**Plug-in org.nhan.pipeline**

Ta cấu hình như sau trong Extensions để sử dụng plug-in org.nhan.sbus. Sử dụng plug-in org.sbus để khai báo pipeline như một services, bao gồm một phương thúc invokePipe để triệu gọi pipeline, với hai tham số là tên của pipeline (pipeName) và một tham số là dạng cấu trúc XML (xmlParams) – để trỏ tới class xử lý pipeline



Với plug-in org.nhan.pipeline ta phải xây dựng một file “.xml” để khai báo các pipeline cần sử dụng, gọi các services nào, sử dụng loại kịch bản nào, giải quyết quy trình nghiệp vụ nào.

File .xml của một pipeline demo như sau:

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <pipelines>  <pipeline name="complex1">  <parameters>  <parameter name="input" type="int"/>  </parameters>  <call id="parent" service="org.nhan.example.test" operation="multiply">  <parameter type="int">2</parameter>  <parameter type="int">  <call service="at.slife.test" operation="add">  <parameter type="int">  <call service="org.nhan.pipeline" operation="square">  <parameter type="int">{input}</parameter>  </call>  </parameter>  <parameter type="int">  <call service="org.nhan.example.test" operation="multiply">  <parameter type="int">15</parameter>  <parameter type="int">{input}</parameter>  </call>  </parameter>  </call>  </parameter>  </call>  <transform xsl="test1.xsl"/>  <transform xsl="test2.xsl"/>  <serialize type="xswt"/>  </pipeline>  <pipeline name="complex2">  <parameters/>  <call id="parent" service="org.nhan.example.test" operation="multiply">  <parameter type="int">21</parameter>  <parameter type="int">  <call service="org.nhan.pipeline" operation="square">  <parameter type="int">10</parameter>  </call>  </parameter>  </call>  <transform xsl="test1.xsl"/>  <transform xsl="test2.xsl"/>  <serialize type="xswt"/>  </pipeline>  <pipeline name="square">  <parameters>  <parameter name="input" type="int"/>  </parameters>  <call service="org.nhan.example.test" operation="multiply" returns="int">  <parameter type="int">{input}</parameter>  <parameter type="int">10</parameter>  </call>  <serialize type="xml"/>  </pipeline>  </pipelines> |

Class PipelinePlugin.java – lớp quản lý plug-in org.nhan.pipeline

|  |
| --- |
| package org.nhan.pipeline;  import java.io.File;  import java.io.IOException;  import java.net.URL;  import org.eclipse.ui.plugin.\*;  import org.eclipse.core.runtime.Path;  import org.eclipse.core.runtime.Platform;  import org.eclipse.jface.resource.ImageDescriptor;  import org.osgi.framework.BundleContext;  /\*\*  \* The main plugin class to be used in the desktop.  \*/  public class PipelinePlugin extends AbstractUIPlugin {  //The shared instance.  private static PipelinePlugin plugin;  /\*\*  \* The constructor.  \*/  public PipelinePlugin() {  plugin = this;  }  /\*\*  \* This method is called upon plug-in activation  \*/  public void start(BundleContext context) throws Exception {  super.start(context);  }  /\*\*  \* This method is called when the plug-in is stopped  \*/  public void stop(BundleContext context) throws Exception {  super.stop(context);  plugin = null;  }  /\*\*  \* Returns the shared instance.  \*/  public static PipelinePlugin getDefault() {  return plugin;  }  /\*\*  \* Returns an image descriptor for the image file at the given  \* plug-in relative path.  \*  \* @param path the path  \* @return the image descriptor  \*/  public static ImageDescriptor getImageDescriptor(String path) {  return AbstractUIPlugin.imageDescriptorFromPlugin("org.nhan.pipeline", path);  }  protected static File getResource(String res) throws IOException {  Path path = new Path(res);  URL url = Platform.find(plugin.getBundle(), path);  url = Platform.resolve(url);  return new File(url.getFile());  }  } |

Class Pipeline.java – xử lý các pipeline

|  |
| --- |
| package org.nhan.pipeline;  import java.io.File;  import java.io.IOException;  import java.io.StringReader;  import java.util.HashMap;  import java.util.Iterator;  import java.util.List;  import java.util.Vector;  import javax.xml.transform.Transformer;  import javax.xml.transform.TransformerConfigurationException;  import javax.xml.transform.TransformerException;  import javax.xml.transform.TransformerFactory;  import javax.xml.transform.stream.StreamSource;  import org.dom4j.Document;  import org.dom4j.DocumentException;  import org.dom4j.DocumentHelper;  import org.dom4j.Element;  import org.dom4j.Node;  import org.dom4j.Text;  import org.dom4j.io.DocumentResult;  import org.dom4j.io.DocumentSource;  import org.dom4j.io.SAXReader;  import org.nhan.services.Call;  public class Pipeline {  private HashMap<String, Parameter> parameters;  /\*\*  \* body of pipeline as XML  \*/  private Element body = null;  /\*\*  \* the pipeline space name  \*/  private String pipeSpace = null;  /\*\*  \* callMap as XML document taht stores all pipelines  \*/  private static Document callMap = null ;  /\*\*  \* pipeline results  \*/  public String result = "";  /\*\*  \* pipeline constructor, to initialize the callMap at first run  \*/  public Pipeline() {  if (null == callMap){  loadPipelines();  }    }    /\*\*  \* reads the specific pipeline and extract the releated parameters  \* @param pipeName  \*/  public void loadPipeline(String pipeName){  setPipeSpace(pipeName.substring(0,pipeName.indexOf(':')));  Node node = callMap.selectSingleNode( "/pipelines/pipeline[@name='"+pipeName+"']" );  body = (Element)node;  parameters = new HashMap<String, Parameter>();  Element params = (Element)body.selectSingleNode("parameters");  System.out.println(params.asXML());  for ( Iterator i = params.elementIterator( "parameter" ); i.hasNext(); ) {  Parameter param = new Parameter((Element) i.next());  parameters.put(param.getName(),param);  }  }  public Pipeline(String theName) {  if (null == callMap){  loadPipelines();  }  setPipeSpace(theName.substring(0,theName.indexOf(':')));  Node node = callMap.selectSingleNode( "/pipelines/pipeline[@name='"+theName+"']" );  body = (Element)node;  parameters = new HashMap<String, Parameter>();  Element params = (Element)body.selectSingleNode("parameters");  System.out.println(params.asXML());  for ( Iterator i = params.elementIterator( "parameter" ); i.hasNext(); ) {  Parameter param = new Parameter((Element) i.next());  parameters.put(param.getName(),param);  }  }  public Parameter getParameter(String paramName){  return parameters.get(paramName);  }  private void setPipeSpace(String thePipeSpace) {  pipeSpace = thePipeSpace;  }  public String getPipeSpace(){  return pipeSpace;  }  public String invokePipe(String pipeName,String xmlParams){  result = "";  loadPipeline(pipeName);  setParameterValues(Util.parseParams(xmlParams));  return processPipeline();  }  public String processPipeline(){  for ( int i = 0, size = body.nodeCount(); i < size; i++ ) {  Node node = body.node(i);  String name = node.getName();  if (null != name){  // the sub-node is aso a call  if (name.startsWith("call")){  CallElement call = new CallElement((Element) node,this);  // the previous results will be cleaned  if (name.equals("call-clean"))  result = call.invoke();  else  result += call.invoke();  }  // the next node is an XSLT transformation  else if (name.equals("transform")){  Element elm = (Element) node;  transform(elm.attributeValue("xsl"));  }  // the pipeline process is finished, and result should be serializes  else if (name.equals("serialize")){  Element elm = (Element) node;  // call the relevant serializer (browser, form, xml, etc)  Call client = new Call(elm.attributeValue("service"));  try {  Object[] content = {getResultDocument().asXML()};  client.invoke(elm.attributeValue("operation"),content);  } catch (Exception e) {  e.printStackTrace();  }    }  }  }  return result;  }  public void setParameterValue(String name, String val) {  Parameter param = parameters.get(name);  param.setValue(val);  parameters.get(name).setValue(val);  }  public void setParameterValues(String[] vals) {  int i = 0;  for (String paramName : parameters.keySet()){  parameters.get(paramName).setValue(vals[i++]);  }  }  public void setParameterValues(Object[] vals) {  int i = 0;  for (String paramName : parameters.keySet()){  parameters.get(paramName).setValue(""+vals[i++]);  }  }  public String getParameterValue(String name) {  return parameters.get(name).getValue();  }  private void loadPipelines(){  File dir = null;  try {  dir = PipelinePlugin.getResource("pipelines");  } catch (IOException e1) {  e1.printStackTrace();  }  SAXReader reader = new SAXReader();  callMap = DocumentHelper.createDocument();  Element root = callMap.addElement( "pipelines" );;  File[] files = dir.listFiles();  for(int i=0,count=files.length;i<count;i++) {  String fileName = files[i].getName();  if (fileName.toUpperCase().endsWith(".XML")) {  String prefix = fileName.substring(0,fileName.indexOf('.')); try {  Document doc = reader.read(files[i]);  Element pipes = doc.getRootElement();  for ( int j = 0, size = pipes.nodeCount(); j < size; j++ ) {  Node node = pipes.node(j);  if (null != node && node.hasContent()){  Element elm = (Element)node;  elm.addAttribute("name", prefix+":"+elm.attributeValue("name"));  root.add(elm.detach());  }  }  } catch (DocumentException e) {  e.printStackTrace();  }  }  }  System.out.println(callMap.asXML());  }  public void transform(String xslFile){  File xsl = null;  try {  xsl = PipelinePlugin.getResource("styles/"+xslFile);  // load the transformer using JAXP  TransformerFactory factory = TransformerFactory.newInstance();  Transformer transformer;  transformer = factory.newTransformer(  new StreamSource(xsl)  );  // now lets style the given document  DocumentSource source = new DocumentSource(getResultDocument());  DocumentResult target = new DocumentResult();  transformer.transform( source, target );  // return the transformed document  result = target.getDocument().getRootElement().asXML();  } catch (IOException e1) {  e1.printStackTrace();  } catch (TransformerConfigurationException e) {  e.printStackTrace();  } catch (TransformerException e) {  e.printStackTrace();  }  }  public String runXPathQuery(String xpathQuery) {  String items="";  List list = getResultDocument().selectNodes( "/result"+xpathQuery );  boolean isArray = false;  if (list.size() > 1)  isArray = true;  for (Object obj : list){  String name = obj.getClass().getName();  String temp = "";  if (name.equals("org.dom4j.tree.DefaultElement"))  temp = ((Element)obj).asXML();  else if (name.equals("org.dom4j.tree.DefaultAttribute"))  temp = ((Node) obj).getText();  else if (name.equals("org.dom4j.tree.DefaultText"))  temp = ((Text) obj).getText();  if (isArray)  items += "<item>"+temp+"</item>";  else  items += temp;  }  return items;  }  public Document getResultDocument(){  SAXReader reader = new SAXReader();  Document doc = null;  try {  if (result.startsWith("<?xml"))  result = result.substring(result.indexOf("?>")+2);  doc = reader.read(new StringReader("<result>"+result+"</result>"));  } catch (DocumentException e) {  e.printStackTrace();  }  return doc;  }  public String[] getParamArray(){  String[] params = new String[parameters.size()];  int i = 0;  for (String paramName : parameters.keySet())  params[i++] = paramName;  return params;  }  } |

Ngoài các class quan trọng như trên, thì còn các class khác hỗ trợ được đính kèm trong luận văn.