Thuật ngữ viết tắt

WSDL – Web Services Description Language

CSDL – Cơ sở dữ liệu

XML – Extensible Markup Language

SEI – service endpoint interface

IDE – Intergrated Development Environment

EMF – Eclipse Modeling Framework

# Phần mở đầu

# Phần nội dung

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

## Công nghệ Java Web Services

### Tổng quan về Web Services

Web service 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[[1]](#footnote-1). Web service 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 Service 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 Service khác. Nó bao gồm các mô đun độc lập cho hoạt động của khách hàng và doanh nghiệp 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 Service không bị phụ thuộc vào bất kì hệ điều hành hay ngôn ngữ lập trình nào. Web service 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 Service không yêu cầu phải sử dụng trình duyệt và ngôn ngữ HTML.

Web Services cũng có thể được nói một cách khác là các khối cơ bản được xây dựng để di chuyển trong hệ thống máy tính phân tán trên Internet. Các chuẩn mở và việc tập trung vào giao tiếp và làm việc cộng tác giữa con người và các ứng dụng đã tạo nên một môi trường nơi mà Web service đang trở thành nền tảng cho việc tích hợp ứng dụng. Các ứng dụng được xây dựng sử dụng các Web service các loại 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 service khi các công ty xây dựng chúng, nhưng hầu hết tất cả các định nghĩa đều có chung các điểm sau:

* Thứ nhất, Web service 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 service đưa ra cách mô tả các giao diện của chúng một cách đủ 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 service – WSDL (Web service Description Language).
* Thứ ba, Web service 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).



Hình 1.1 Cơ chế hoạt động của Web Services

Web service 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 service 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 service cho phép người sử dụng xây dựng các ứng dụng có các tính năng mạnh hơn thông qua việc sử dụng Web service như những block được xây 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 service 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.

Một tài liệu XML bao gồm hai thành phần:

Cấu trúc logic : định nghĩa các phần tử, các thuộc tính, các kiểu dữ liệu…

Cấu trúc vật lý : chứa dữ liệu và các phần tử chứa dữ liệu như text, image, media…

Ví dụ về XML:

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

<Employer>

<name>Hồ Nhân</name>

<old>28</old>

</Employer>

#### 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 Service như một tập các đối tượng trừu tượng gọi là các “ports” và “endpoint”. Một tài liệu WSDL cũng định nghĩa bên trong nó các phương thức của web service. 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)

Cấu trúc của một WSDL :



Hình 1.3 Cấu trúc WSDL

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)

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 Service 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 Service, 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 Service 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 Service 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 Service 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 nghiã 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

##### Tổng quan

Về cơ bản Universal Description, Discovery, and Intergration (UDDI) là một nơi mà các tổ chức đăng ký và tìm kiếm các Web Service. 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 phần loại 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 service 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 Service. 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.

Thành phần xử lý chính là bộ đăng ký UDDI, đó là một file XML dùng để mô tả một thực thể kinh doanh (business entity) kèm theo các Web service đi cùng. Sử dụng các dịch vụ của UDDI, các doanh nghiệp đăng ký thông tin về những Web service mà họ định cung cấp. Thông tin này được thêm vào UDDI registry thông qua Website hoặc sử dụng các công cụ lập trình sử dụng các dịch vụ theo đúng đặc tả UDDI programmer’s API.

##### Các thành phần của UDDI

UDDI gồm 2 thành phần chính:

Phần đăng ký của tất cả các Web Service’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 Service. Thêm nữa tính năng độc lập với nền tảng ngôn ngữ lập trình đã được điều hợp cùng với giao thức SOAP.

##### Mô hình dữ liệu của UDDI

UDDI bao gồm lược đồ XML, mô tả bốn kiểu cấu trúc dữ liệu dưới đây:

businessEntity

businessService

bindingTemplate

tModel

publisherAssertion



Hình 1.6 Mô hình dữ liệu của UDDI

###### businessEntity

Một businessEntity chứa các thông tin về công ty, bao gồm danh sách liên lạc, thông tin, phân biệt các tổ chức thương mại, và danh sách các nhà cung cấp dịch vụ web. Sau khi đăng kí thì mỗi công ty, doanh nghiệp… sẽ nhận được một định danh duy nhất.

###### businessService

Liên kết với mỗi business entity là một danh sách các business service cung cấp bởi business entity đó. Mỗi thành phần chứa thông tin mô tả về dịch vụ, về thông tin phân loại của dịch vụ và danh sách các binding template liên quan đến thông tin kỹ thuật của dịch vụ. Mỗi business service cần có ít nhất một binding template.

###### bindingTemplate

Gắn với mỗi business service là một danh sách các binding template cung cấp thông tin về địa điểm có thể tìm thấy Web Service và làm cách nào để sử dụng nó. Một cấu trúc binding template mô tả thông tin interface của Web Service và các địa chỉ URL. Mỗi bindingTemplate được định danh duy nhất thông qua số phát sinh tự động UUID lưu trong bindingKey.

###### tModel

tModel là lõi trong cùng của kiểu dữ liệu, nhưng rất khó có khả năng để có thể nắm bắt được hết. tModel là chuẩn cho mô hình kĩ thuật.

tModel là phương pháp để mô tả một vài quy trình thương mại, dịch vụ và các cấu trúc mẫu lưu trữ trong UDDI registry. Bất kì một khái niệm trừu tượng nào đều có thể được đăng ký trong UDDI như là một tModel. Ví dụ: chúng ta có thể định nghĩa ra một kiểu cổng (port type) WSDL mới, và đồng nghĩa với đó ta có thể định nghĩa ra một tModel mới mà trình bày kiểu cổng đó trong UDDI. Sau đó, ta có thể chỉ định ra dịch vụ thương mại mà thực thi kiểu cổng đó bằng việc kết hợp với tModel với một business service’s binding template.

Mục đích của tModel là dùng để liên kết đến metadata bên ngoài UDDI. Thành phần quan trọng nhất của tModel là một URL trỏ đến một tài liệu mô tả thông tin metadata. Tài liệu này có thể là tài liệu bất kì HTML, Word, .. tùy ý mô tả một đặc tả kỹ thuật nào đó, ví dụ như giao thức mạng, dạng thức trao đổi hoặc luật tuần tự mà thông thường nhất là file mô tả thông tin service WSDL. Có hai thuộc tính cơ bản bên trong một tModel : tModelKey đóng vai trò định danh duy nhất giữa các tModel với nhau và name dùng cung cấp một tên với đầy đủ ngữ nghĩa cho tModel.

###### publisherAssertion

Đây là một cấu trúc dữ liệu quan hệ mà nó đặt sự kết hợp giữa hai hoặc nhiều cấu trúc dữ liệu businessEntity theo một kiểu quan hệ cụ thể, chẳng hạn như một công ty con hoặc một phòng ban.

Cấu trúc dữ liệu pubisherAssertion bao gồm ba thành phần chính: fromkey (BusinessKey đầu tiên), toKey (bussinesskey thứ hai) và keyedReference. KeyReference thiết kế ra kiểu mỗi quan hệ kết hợp trong cặp thuật ngữ keyName, keyValue trong tModel. Tham chiếu duy nhất bởi tModelkey.

#### SOAP – Simple Object Access Protocol

##### Tổng quan

Làm thế nào chúng ta truy xuất dịch vụ khi tìm thấy ? Câu trả lời là các dịch vụ Web có thể truy xuất bằng một giao thức là Simple Object Access Protocol – SOAP. Nói cách khác chúng ta có thể truy xuất đến UDDI registry bằng các lệnh gọi hoàn toàn theo kiểu SOAP.

SOAP là một giao thức dựa trên XML để trao đổi thông tin giữa các máy tính. Mặc dù 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. SOAP do đó 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>

##### Những kiểu truyền thông

SOAP hỗ trợ hai kiểu truyền thông khác nhau:

Remote Procedure Call (RPC): cho phép gọi hàm hoặc thủ tục qua mạng. Kiểu này đư­ợc khai thác bởi nhiều Các Dịch vụ Web và có nhiều trợ giúp.

Document: đư­ợc biết nh­ư kiểu hư­ớng message, kiểu này cung cấp một lớp thấp của sự trừu tư­ợng hóa, và yêu cầu người lập trình nhiều hơn khi làm việc.

Các định dạng message, tham số, và lời gọi đến các API thì tương ứng trong RPC và document là khác nhau. Nên việc quyết định chọn cái nào tùy thuộc vào thời gian xây dựng và sự phù hợp của dịch vụ cần xây dựng.

##### Mô hình dữ liệu

Mục đích của mô hình dữ liệu SOAP là cung cấp những 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.

### Các thư viện dùng để tạo Web Services trong Java

Trong Java có hai thư viện phổ biến nhất thường được sử dụng để xây dựng một Web Service, đó là JAX-WS và Apache Axis2.

#### JAX-WS

##### Tổng quan

JAX-WS (Java API for XML Web Services) là một bộ thư viện được phát triển dưới sự quản lý của Sun Microsystem. Trang chủ của dự án <https://jax-ws.java.net/> là nơi cung cấp các tài liệu hướng dẫn, các bản cập nhật thường xuyên của bộ thư viện này.

JAX-WS giúp tạo ra các dịch vụ web sủ dụng giao thức XML. Với việc được tích hợp sẵn trong bộ Java EE SDK và được đóng gói kèm theo trong các bộ cộng cụ tích hợp dành cho Java EE. nên JAX-WS là bộ thư viện rất phổ biến để tạo ra những dịch vụ web.

JAX-WS làm tăng tốc độ phát triển dịch vụ Web thông qua việc cung cấp một thư viện các chú giải để chuyển các lớp đối tượng Java cũ đơn giản (plain old Java object) (POJO) thành các Web Service. Nó cũng quy định một sự ánh xạ chi tiết từ một dịch vụ được định nghĩa trong WSDL đến các lớp Java thực hiện dịch vụ đó.

Phát triển một Web Service bằng JAX-WS với một trong hai cách tiếp cận:

Theo mục đích: bắt đầu bằng một WSDL và tạo một lớp Java để thực hiện dịch vụ. Tuy nhiên các tiếp cận này đòi hỏi một sự hiểu biết tốt về WSDL và XSD (định nghĩa lược đồ XML) để định nghĩa các định dạng thông điệp

Theo mã: bắt đầu bằng một lớp Java và sử dụng công cụ để tạo cả tệp WSDL và giao diện Java

##### Các bước cơ bản để tạo Web Service với JAX-WS

Thông thường một web service cần có một service endpoint interface (SEI) là một interface dùng để khai báo các phương thức mà client có thể gọi đến. Tuy nhiên khi xây dựng web service với JAX-WS các SEI cần thiết phải có. Sau đó các interface SEI sẽ được cụ thể hóa bằng phiên bản cài đặt của web service (implementation class). Các bước cơ bản gồm có:

Cài đặt các phương thức cho web service

Biên dịch các class của web service thành một tập tin WAR

Cài đặt và triển khai tập tin WAR lên server. Các class thành phần dùng cho việc giao tiếp với client sẽ được tạo ra bởi server trong quá trình cài đặt (deploy)

##### Một số lưu ý khi cài đặt một SEI

Lớp cài đặt của web service phải được đánh dấu là các class javax.jws.WebService hoặc javax.jws.WebServiceProvider (dùng annotation @)

Lớp cài đặt phải được chỉ rõ là cài đặt cho interface SEI nào thông qua thuộc tính endpointInterface của @WebService, nhưng việc này không bắt buộc. Nếu không có endpointInterface nào được chỉ ra, thì class cài đặt đó sẽ được ngầm định là có một SEI

Các phương thức của lớp cài đặt phải là các phương thức public và không được là các phương thức tĩnh static hay final

Các phương thức của web service cho phép client gọi cần phải có khai báo annotation là @javax.jws.WebMethod

Các tham số và các kiểu trả về của các phương thức web service cung cấp cho client phải là các kiểu nằm trong JAX-B-compatible parameters and return types. Xem [Default Data Type Bindings](http://java.sun.com/webservices/docs/2.0/tutorial/doc/JAXBWorks4.html#wp82947)

Lớp cài đặt không được là một lớp final hay abstract

Lớp cài đặt phải có một constructor mặc định

Lớp cài đặt không được phép định nghĩa phương thức finalize

Lớp cài đặt cần sử dụng các annotation là javax.annotation. PostConstruct hoặc javax.annotation.PreDestroy trong các hàm phục vụ việc quản lý vòng đời của các sự kiện trong Web service:một phương thức @PostConstruct được gọi bởi web service container trước khi class cài đặt của web service phản hồi lại client; các phương thức được khai báo với @PreDestroy web service container trước khi web service được gỡ bỏ

#### Apache Axis2

##### Tổng quan

Đây là bộ thư viện mã nguồn mở được phát triển bởi Apache Software Foundation. Trang chủ của dự án [http://axis.apache.org/axis2/java/core/](http://axis.apache.org/axis2/java/core/%20) là nơi cung cấp các tài liệu hướng dẫn, các bản cập nhật thường xuyên của bộ thư viện này. Dự án Axis2 là một sự cài đặt dựa trên ngôn ngữ lập trình Java cho cả phía Client và Server trong mô hình Web Services.

Apache Axis2 cung cấp một mô hình đối tượng hoàn chỉnh và một kiến trúc môđun hóa cho phép dễ dàng việc thêm vào các chức năng và hỗ trợ những đề xuất và các đặc tả mới liên quan đến Web services.

##### Các đặc điểm của Apache Axis2

Axis 2 cung cấp một khung ứng dụng để xử lý các thông điệp SOAP. Khung ứng dụng này có thể được mở rộng trong các xử lý thông điệp SOAP hoặc trong các hoạt động cơ sở. Hơn nữa, Axis2 cho phép mô hình hóa các dạng trao đổi thông điệp khác nhau (MEPs) để được sử dụng ở trong khung xử lý (processing framework)

* Axis2 có khả năng triển khai một dịch vụ Web với sự hỗ trợ của WSDL hoặc không cần WSDL
* Axis2 cung cấp giao diện thao tác người dùng (Client API) sử dụng để gọi các dịch vụ Web. Giao diện thao tác người dùng có thể hỗ trợ cho các hai mô hình đồng bộ và bất đồng bộ
* Axis2 cho phép thay đổi thông tin cấu hình hoặc ở các thành phần của nó trong triển khai dịch vụ Web
* Axis2 cung cấp khả năng nhận và gửi các thông điệp SOAP với nhiều giao thức khác nhau

##### Mô hình xử lý thông điệp SOAP

Nhận và gửi thông điệp SOAP có thể coi là hai trong số những công việc chủ yếu của máy xử lý SOAP. Trong kiến trúc Axis2 cung cấp hai đường ống để thực hiện hai hành động cơ bản là nhận và gửi thông điệp SOAP. Hai đường ống này có tên là InPipe và OutPipe. Các mẫu tin nhắn trao đổi phức tạp ký hiệu MEPs, nó được xây dựng bởi kết nối hai loại ống InPipe và OutPipe. Ngoài ra còn hai đường ống khác có mục đích hỗ trợ cho các thông điệp gửi đến và gửi đi bị lỗi



Hình 1.8 Mô hình xử lý thông điệp SOAP

Khả năng mở rộng của mô hình xử lý SOAP được cung cấp thông qua cơ chế xử lý (handler). Khi một thông điệp đang được xử lý, thì các điều khiển này đã được đăng ký để sẽ thực hiện. Các điều khiển này được đăng ký ở mức độ toàn cục, dịch vụ, hoặc trong một phạm vi hoạt động giới hạn của máy chủ Axis2. Cuối cùng Axis2 kết nối các điều khiển của các phạm vi trên thành một chuỗi điều khiển. Các điều khiển có thể hoạt động như một máy đánh chặn, nghĩa là chúng xử lý một phần thông điệp và sau đó cung cấp các tiện ích dịch vụ (add on services) cho hệ thống khác. Mô tả các bước truyền thông điệp trong mô hình xử lý thông điệp SOAP ở Hình 1.8

Khi một thông điệp được gửi đến thông qua Client API, ống OutPipe trên vùng người dùng được kích hoạt. Ống OutPipe sẽ gọi các xử lý (handlers)

Bộ giao vận gửi thông điệp (Transport sender) trên vùng người dùng sẽ gửi thông điệp SOAP tới ống InPipe trên vùng dịch vụ.Thông điệp SOAP này cuối cùng được bộ giao vận gửi thông điệp (Transport sender) trên vùng dịch vụ chuyển tới cuối điểm mục tiêu (endpoint) trên vùng dịch vụ.

Thông điệp SOAP này sau đó được nhận bởi bộ giao vận nhận (Transport reviever) của vùng người dùng ở điểm mục tiêu (endpoint). Thông điệp SOAP tiếp tục được chuyển vào ống InPipe ở vùng người dùng. Trong ống InPipe này bao gồm các thao các xử lý( handlers) và kết thúc việc nhận thông điệp SOAP. Đồng thời kết thúc quá trình gửi và nhận thông điệp phản hồi SOAP cho Client API.

Quá trình trao đổi thông điệp ở trên xảy ra với mỗi thông điệp tại một thời điểm và với mọi thông điệp. Sau khi xử lý xong một thông điệp, Axis2 có thể quyết định tạo ra một thông điệp SOAP khác. Axis2 luôn luôn xem các thông điệp SOAP ở dạng xử lý đơn. Sự quản lý và kết nối các thông điệp được thực hiện ở lớp trên của khung (framework). Hai ống InPipe và OutPipe không có sự phân biệt khác nhau giữa vùng máy chủ và vùng người dùng.

##### Các phiên bản chính của Apache Axis2

Standard Binary Distribution: là phiên bản hoàn chỉnh của Axis, chứa sample và nhiều script hữu ích. Nó được sử dụng để cài đặt như một server độc lập và dùng trong quá trình phát triển ứng dụng

WAR (Web Archive) Distribution (Axis2.war): là ứng dụng web của Axis2, được triển khai trên hầu hết các trình chứa Servlet Container, ví dụ Tomcat…

## 1.2 Kiến trúc hướng dịch vụ

### 1.2.1 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 lại 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 khác hàng.

Một trong những cách hiểu sai lầm nhất về SOA là coi SOA là một công nghệ. Mặc dù SOA hoạt động được là nhờ công nghệ, nhưng khách hàng cần phải chuyển đổi từ chỗ chỉ việc tích hợp công nghệ SOA sang việc phải điều chỉnh các phương pháp thực hiện dự án, chính sách bảo trì và thay đổi để đạt được các lợi ích về khả năng trưởng thành và đáp ứng.

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 Service) được xây dựng dựa trên nền tảng web toàn cầu, bất cứ nơi nào cũng có, 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 Service trông có vẻ giống nhau nhưng chúng không phải là một.



Hình 1.9 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 1.10 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 họ cần sử dụng đang hoạt động. Họ chỉ cần quan tâm dịch vụ đó là gì.

Serive Consumer: khách hàng dịch vụ hay những user 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 platform 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 1.11 Message được truyền nhận giữa các dịch vụ

Sử dụng thông điệp (message) để giao tiếp có các lợi thế sau:

Độc lập nền: thông điệp (message) trở thành ngôn ngữ chung của các platform và các ngôn ngữ lập trình khác nhau. Điều này đảm bảo các service trên các platform khác nhau hoạt động với cấu trúc dữ liệu đặc thù của platform đó.

Giao tiếp bất đồng bộ: Người gửi và người nhận không cần phải chờ thông điệp trả lời sau khi đã gởi đi một thông điệp. Điều này giúp cho người gửi và người nhận tiếp tục xử lý công việc sau khi gửi thông điệp mà không cần dững thực thi để chờ thông điệp trả lời.

Giao tiếp tin cậy: các thông điệp từ bên gửi có thể được gửi đến một service trung gian có nhiệm vụ lưu trữ (store) các thông điệp. Service trung gian sẽ chuyển tiếp (forward) thông điệp cho bên nhận khi bên nhận có thể xử lý yêu cầu tiếp theo. Cơ chế Store-and-Forward này đảm bảo các thông điệp sẽ không bị thất lạc trong trường hợp Receiver bị quá tải và không thể nhận thêm yêu cầu mới.

Quản lý luồng: Việc trao đổi thông điệp theo cơ chế bất đồng bộ giúp ứng dụng không cần ngừng thực thi để chờ một tác vụ kết thúc mà có thể tạo ra các luồng (thread) xử lý các công việc khác nhau.

* Giao tiếp từ xa: Các thông điệp lưu trữ thông tin về các đối tượng dữ liệu dưới dạng đặc tả hình thức thay thế việc phải serialization and deserialization các đối tượng dữ liệu truyền qua mạng khi ứng dụng thực hiện gọi từ xa một ứng dụng khác.
* Bảo mật end-to-end: Thông điệp có thể lưu trữ thông tin về hình thức bảo mật của kênh giao tiếp. Điều này cung cấp khả năng điều khiển liên quan đến bảo mật như xác thực và phân quyền.

### Các nguyên tắc chính của hệ thống SOA

#### Sự phân định rạch ròi giữa các dịch vụ

Do có sự tách biệt giữa thành phần giao tiếp và thành phần thực hiện dịch vụ trong kiến trúc hướng dịch vụ. Các dịch vụ này sẽ thực hiện quá trình tương tác chủ yếu thông qua thành phần giao tiếp. Thành phần này sẽ quy định những dạng thông điệp trong quá trình trao đổi: thông điệp nào sẽ được chấp nhận và thông điệp nào sẽ không được xử lý. Và đây chính là cách duy nhất để các đối tượng bên ngoài có thể truy cập vào thông tin và chức năng của dịch vụ, ta chỉ cần gửi thông điệp được định dạng đến trước để yêu cầu dịch vụ mà không cần biết thông điệp đó sẽ được xử lý như thế nào.

#### Các dịch vụ tự hoạt động

Các dịch vụ cần được triển khai và hoạt động như một thực thể độc lập mà không phụ thuộc và các dịch vụ khác.

Dịch vụ phải có tính bền vững cao, nghĩa là nó không bị sụp đổ khi có sự cố. Để thực hiện điều này, các dịch vụ cần duy trì đầy đủ thông tin cần thiết cho quá trình hoạt động của mình để có thể tiếp tục hoạt động trong trường hợp dịch vụ cộng tác của nó bị hỏng, đồng thời sử dụng các biện pháp bảo mật để tránh các cuộc tấn công ồ ạt từ bên ngoài vào như gửi thông điệp lỗi hoặc thông điệp ồ ạt. Đây chính là ý nghĩa của khái niệm kết nối lõng lẻo.

#### Các dịch vụ chia sẻ lược đồ

Các dịch vụ nên cung cấp thành phần giao tiếp (Interface) của nó ra bên ngoài và hỗ trợ chia sẻ các cấu trúc thông tin, các ràng buộc dữ liệu thông qua các lược đồ dữ liệu (Schema) chuẩn. Như thế hệ thống sẽ có tính dễ liên kết và dễ dàng mở rộng.

#### Tính tương thích của các dịch vụ dựa trên chính sách

Một dịch vụ muốn tương tác với các dịch vụ khác thì phải thỏa mãn các chính sách (Policy), các yêu cầu (Requirements) của dịch vụ đó như: mã hóa, bảo mật…Mỗi dịch vụ phải cung cấp công khai các chính sách và các yêu cầu bảo mật của mình.

### Các tính chất của một hệ thống SOA

#### Kết nối lõng lẻo

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 bao nhiêu thì có nhiều thay đổi 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 cúng được che dấu đi. 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 chuần 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 ràng 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 rành 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 điếp 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. Interoperability is achieved 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 Service 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 và đây cũng là một mô hình khá phổ biến cho kiến trúc của hệ thống SOA.



Hình 1.12 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 đối tác (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 để 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 để lưu trữ các ứng dụng SOA và giúp cung cấp sử dụng hiệu quả các nguồn tài nguyên đế 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 thế được truy cập và sử dụng từ bên trong tổ chức với sự giúp đỡ của dịch vụ ESB.

## Quy trình xây dựng SOA

Hiện nay, chưa có một quy trình cụ thể để phát triển các ứng dụng theo kiến trúc hướng dịch vụ. Tuy nhiên, dựa trên thực tế, 12 bước sau đã được đưa ra nhằm tham khảo khi quyết định chuyển sang định hướng dịch vụ.

12 bước trong quy trình xây dựng ứng dụng theo kiến trúc hướng dịch vụ:

1. Hiểu nghiệp vụ
2. Xác định phạm vi (miền) của vấn đề
3. Hiểu tất cả các ngữ nghĩa ứng dụng trong miền đó
4. Hiểu tất cả các dịch vụ hiện có trong miền
5. Hiểu tất cả các nguồn và đích của thông tin có trong miền
6. Hiểu tất cả các quy trình trong miền
7. Xác định và phân loại tất cả các giao diện bên ngoài miền cần thiết cho việc xây dựng ứng dụng (các dịch vụ và thông tin)
8. Định nghĩa các dịch vụ mới và các ràng buộc thông tin của các dịch vụ đó.
9. Định nghĩa các quy trình mới, cũng như các dịch vụ và ràng buộc thông tin cho các quy trình này.
10. Lựa chọn tập công nghệ.
11. Triển khai công nghệ SOA.
12. Kiểm thử và đánh giá.

**Bước 1: Hiểu các mục đích nghiệp vụ, và xác định thành công**

Đây là nhiệm vụ thu thập yêu cầu cơ bản. Nó đòi hỏi phải tiếp xúc với tài liệu, nhân sự và hệ thống để xác định thông tin cho phép việc tích hợp ứng dụng được xác định đúng để có thể phân tích, mô hình hóa và cải tiến. Chỉ sau khi thực hiện bước này thì tập giải pháp thích hợp mới được đưa ra.

Cần lưu ý rằng có cả lợi ích hữu hình và vô hình từ việc cài đặt bất kỳ loại công nghẹ nào. Lợi ích hữu hình bao gồm việc tiết kiệm chi phí tức thì, như việc tự động hóa một hệ thống bán theo đơn đặt hàng thay cho việc bán hàng thủ công. Lợi ích vô hình thì khó nhận biết hơn, như sự thỏa mãn của khách hàng dẫn tới việc tăng doanh số bán hàng, hoặc cộng tác chặt chẽ hơn tăng cường chất lượng và cho phép các nhân công trao đổi thông tin tốt hơn.

**Bước 2: Xác định miền vấn đề**

Phải xác định phạm vi của việc ứng dụng SOA trong một tổ chức. Hãy chia nhỏ tổ chức để áp dụng SOA thay vì áp dụng vào toàn bộ tổ chức. Cùng với thời gian, những thành công nhỏ sẽ dẫn tới các chiến lược thành công lớn hơn, phải thiết lập các đường ranh giới khi bắt đầu dự án để có thế tiến hành xây dựng ứng dụng một cách trọng tâm hơn.

**Bước 3: Hiểu tất cả các ngữ nghĩa ứng dụng trong miền vấn đề**

Chúng ta không thể giải quyết các vấn để mà bản thân mình không hiểu rõ. Vì vậy, bước tiếp theo này là cực kỳ quan trọng để xác định tất cả các siêu dữ liệu ngữ nghĩa tồn tại trong miền ứng dụng nhằm giải quyết vấn đề một cách hoàn hảo. Sự hiểu biết về ngữ nghĩa ứng dụng thiết lập cách thức và khuôn dạng trong đó ứng dụng tham khảo các thuộc tính của quy trình nghiệp vụ. Ví dụ, cùng một số hiệu khách hàng nhưng trong một ứng dụng này có thể có một giá trị và ý nghĩa hoàn toàn khác trong một ứng dụng khác. Việc hiểu ngữ nghĩa của ứng dụng đảm bảo rằng sẽ không có bất kỳ sự xung đột thông tin nào khi nó được tích hợp với các ứng dụng khác ở mức độ thông tin hoặc dịch vụ. Xác định ngữ nghĩa của ứng dụng là một công việc khó khăn vì có thể nhiều hệ thống mà chúng ta đang phân tích đã cũ hoặc mang tính độc quyền. Bước đầu tiên trong việc xác định và định vị ngữ nghĩa là tạo ra một danh sách các hệ thống ứng viên. Danh sách này sẽ cho phép chúng ta có thể xác định những kho chứa dữ liệu nào tồn tại trong các hệ thống đó. Bất kỳ công nghệ nào có thể xây dựng lại các lược đồ dữ liệu vật lý và logic cũng sẽ có ích trong việc xác định dữ liệu trong các miền vấn đề. Tuy nhiên, trong khi lược đồ và mô hình dữ liệu có thể cho phép nhìn vào cấu trúc của cơ sở dữ liệu thì chúng lại không thể xác định những thông tin đó được sử dụng như thế nào trong ngữ cảnh của dịch vụ hoặc ứng dụng; đó là lý do chúng ta cần tới các bước tiếp theo. Bằng việc hiểu rõ các ngữ nghĩa của ứng dụng, chúng ta có thể hiểu trọn vẹn việc tích hợp ứng dụng, và đảm bảo rằng tất cả các hệ thống nguồn và đích trong và giữa các tổ chức được tích hợp một cách hoàn hảo.

**Bước 4: Hiểu tất cả các dịch vụ hiện có trong miền**

Tìm hiểu các thông tin về dịch vụ, bao gồm:

Vị trí của dịch vụ

Mục đích của dịch vụ

Thông tin ràng buộc của dịch vụ

Các phụ thuộc (nếu đó là một dịch vụ phức hợp)

Các vấn đề bảo mật

Vị trí tốt nhất để bắt đầu tìm hiểu là thư mục dịch vụ. Giống như các thư mục khác, đây là một kho chứa các thông tin được thu thập về các dịch vụ hiện có, cùng với các tài liệu về mỗi dịch vụ, bao gồm mục đích của dịch vụ, các thông tin vào/ ra... Thư mục này được sử dụng cùng với những hiểu biết về ngữ nghĩa của ứng dụng để xác định các điểm tích hợp trong tất cả các hệ thống của miền vấn đề.

**Bước 5: Hiểu tất cả các nguồn và đích thông tin hiện có trong miền vấn đề**

Bước này xác định các giao diện xử lý các thông tin đơn giản. Chúng có thể thực hiện một trong 2 vai trò: sử dụng thông tin (đích) hoặc cung cấp thông tin (nguồn).

Chúng ta cần phải hiểu rõ các khía cạnh sau:

Vị trí của chúng

Cấu trúc của luồng thông tin vào/ ra

Các ràng buộc tích hợp

Các phụ thuộc (các nguồn và đích khác, cũng có thể là các dịch vụ)

Các vấn đề bảo mật

**Bước 6: Hiểu tất cả các quy trình**

Chúng ta cần xác định và liệt kê tất cả các quy trình nghiệp vụ tồn tài trong miền vấn đề, có thể là tự động hóa hoặc không phải tự động hóa. Việc này rất quan trọng vì chúng ta đã biết các dịch vụ và nguồn/ đích nào hiện có, chúng ta cần phải xác định các cơ chế tương tác cao hơn, bao gồm tất cả các quy trình ở mức mức cao, mức trung bình và mức thấp. Trong nhiều trường hợp, những quy trình này vẫn chưa được tự động hóa hoặc chỉ có một phần được tự động hóa. Ví dụ, nếu một kiến trúc sư tích hợp ứng dụng cần hiểu tất cả các quy trình hiện có trong một ứng dụng kiểm kê, anh ta sẽ hoặc là đọc tài liệu hoặc là đọc mã nguồn để xác định quy trình nào đang được thực hiện. Sau đó, anh ta sẽ đưa quy trình nghiệp vụ vào phân loại và xác định mục đích của quy trình, ai là người sở hữu nó, nó chính xác là gì, và công nghệ để thực hiện nó (Java hoặc C++ ...). Những quy trình này sau đó được gắn với các quy trình mới để đáp ứng được yêu cầu nghiệp vụ. Chúng ta cần phải xem xét khái niệm quy trình chia sẻ và quy trình riêng. Một số quy trình là quy trình riêng, và do đó, chúng không chia sẻ với các thực thể bên ngoài (trong một số trường hợp, chúng thậm chí còn không chia sẻ với các phần khác của tổ chức). Các quy trình chia sẻ và quy trình riêng có thể tồn tại trong cùng một không gian quy trình với công nghệ tích hợp quy trình quản lý bảo mật giữa các người dùng.

Một thông tin có thể được bảo trì trong phân loại, đó là thông tin bao gồm các biến được sử dụng trong các quy trình, các lược đồ đối tượng, các yêu cầu bảo mật, và/ hoặc các đặc điểm hiệu suất. Mỗi phân loại quy trình phải duy trì tập thuộc tính riêng của nó, được xây dựng tùy biến cho mỗi yêu cầu tích hợp ứng dụng cụ thể.

**Bước 7: Xác định và phân loại tất cả các giao diện bên ngoài miền cần thiết cho việc xây dựng ứng dụng**

Chúng ta cần xác định tất cả các giao diện bên ngoài mà các hệ thống trong miền vấn đề của chúng ta có tương tác với, hoặc cần tương tác với để đem lại giá trị tối đa. Điều quan trọng ở đây là phải chắc chắn rằng tất cả các giao diện cần thiết đều được xác định, bao gồm khả năng thể hiện các dịch vụ của miền vấn đề ra bên ngoài cho các đối tác, cũng như khả năng nhận biết và thúc đẩy dịch vụ của họ. Các hệ thống của đối tác và của chúng ta cần hoạt động cùng nhau để hỗ trợ các quy trình chia sẻ chung.

**Bước 8: Xác định các dịch vụ mới, các dịch vụ phức hợp và thông tin ràng buộc đối với các dịch vụ đó**

Chúng ta cần phải xác định tất cả các dịch vụ tạo thành SOA; những dịch vụ này được chia thành 3 loại.

**Bước 9: Xác định các quy trình mới, cũng như các dịch vụ và thông tin ràng buộc đối với các quy trình đó**

Đến bước này, chúng ta cần hiểu phần lớn những thành phần cần thiết để xác định các quy trình mới, cũng như liên kết chúng với các quy trình hiện có, tự động hóa các quy trình mà trước chưa được tự động hóa.

**Bước 10: Lựa chọn tập công nghệ**

Có rất nhiều các công nghệ để lựa chọn, gồm các máy chủ ứng dụng, các đối tượng phân tán, và các máy chủ tích hợp. Sự lựa chọn công nghệ sẽ giống như một sự tổng hợp các sản phẩm và nhà cung cấp đưa ra để đáp ứng được yêu cầu cho SOA. Rất hiếm có trường hợp một nhà cung cấp duy nhất có khả năng giải quyết được tất cả các vấn đề.

Lựa chọn công nghệ là một công việc khó khăn yêu cầu một lượng thời gian và công sức đáng kể. Việc tạo ra tiêu chuẩn cho công nghệ và sản phẩm, việc hiểu rõ các giải pháp được đưa ra, và sau đó nối các tiêu chuẩn với các sản phẩm đó là việc không dễ dàng. Để thành công, việc kết nối tiêu chuẩn với sản phẩm thường đòi hỏi một dự án thử nghiệm để chứng minh rằng nó sẽ hoạt động. Thời gian cần thiết để lựa chọn các công nghệ phù hợp có thể dài bằng thời gian phát triển SOA, nhưng nếu nản chí, có thể sẽ dẫn tới việc lựa chọn các công nghệ không phù hợp dẫn đến phá hỏng hệ thống.

**Bước 11: Triển khai công nghệ SOA**

Đến bước này, chúng ta đã hiểu tất cả những gì cần phải hiểu, đã xác định được các dịch vụ và quy trình mới, đã chọn lựa được tập công nghệ thích hợp, và bây giờ sẽ là thời gian để xây dựng hệ thống.

**Bước 12: Kiểm thử và đánh giá**

Để đảm bảo cho việc kiểm thử, cần phải xây dựng kế hoạch kiểm thử.

Cần phải hiểu ràng 12 bước trên không phải là quy trình bắt buộc để xây dựng một dự án SOA thành công. Trong một số trường hợp, chúng ta cần thêm vào hoặc xóa bỏ đi một số bước để phù hợp với từng yêu cầu cụ thể.

Kiến trúc hướng dịch vụ được đưa ra nhằm loại bỏ sự trùng lặp và dư thừa qua việc tái sử dụng và tích hợp. Cách đơn giản nhất để bắt đầu với SOA là thử với một dịch vụ mà chúng ta biết rằng có nhiều cài đặt trong các ứng dụng khác nhau, sau đó bắt đầu xây dựng kế hoạch và chiến lược để loại bỏ các dịch vụ dư thừa. Kế hoạch này được gia tăng dần để loại bỏ các bản sao dư thừa. Quy trình cài đặt này sẽ giúp chúng ta đáp ứng được các yêu cầu cơ bản của tổ chức ẩn sau bước chuyển đổi thành công sang SOA. Khi chúng ta đã thực hiện được quy trình này, chúng ta sẽ có tất cả các công cụ và hiểu biết để mở rộng quy mô áp dụng SOA trong tổ chức của mình.

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

### Giới thiệu

Web Service 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 [bpel\_tk1] (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 1.13 thể hiện một tiến trình BPEL trong thực tế.

****

Hình 1.13 Ví dụ về một tiến trình BPEL

### 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 service 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

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

<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>

Hình 1.14 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.

#### Các thành phần và ký hiệu trong BPEL

Một tiến trình BPEL được thể hiện qua các Activity, các Activity trong BPEL được thực hiện tuần tự theo cấu trúc được khai báo trong tiến trình. Trong BPEL 2.0 thì các Activity được chia làm ba nhóm cơ bản như sau:

**Basic Activity:** là các Activity đơn thể, nó không thể chứa được bất kỳ các Actyvity nào khác bên trong nó nữa.

**Structrer Activity:** là các Activity có cấu trúc, nó có thể chứa được các Activity khác bên trong nó.

**Faul Handle Activity:** các Activity này được sử dụng để thụ lý lỗi và các ngoại lệ xảy ra trong quá trình hoạt động của một tiến trình

Tùy theo nhu cầu và trong các trường hợp cụ thể mà ta có thể chọn và sử dụng các Activity khác nhau. Bảng sau mô tả chi tiết về các Activity trong BPEL 2.0:

|  |  |  |
| --- | --- | --- |
| Tên Activity | Ký hiệu | Các trường hợp sử dụng |
| Các Activity cơ bản (Basic Activity) | | |
| Empty | D:\Study\Exam\BPELProject\Ver6.0\BPEL\src\bpel\images\icons20\empty.png | Là một activity đặc biệt, không làm gì hết khi được gọi. Activity này được dùng khi cần có một activity nhưng không thật sự cần một hành động nào xảy ra. |
| Invoke | D:\Study\Exam\BPELProject\Ver6.0\BPEL\src\bpel\images\icons20\invoke.png | Gọi một web servirse để thực hiện một tác vụ nào đó |
| Receive | D:\Study\Exam\BPELProject\Ver6.0\BPEL\src\bpel\images\icons20\receive.png | Nhận một thông điệp từ một service partner. Thông thường đây là activity bắt đầu một tiến trình mới |
| Reply | D:\Study\Exam\BPELProject\Ver6.0\BPEL\src\bpel\images\icons20\reply.png | Gửi trả một thông điệp cho một đối tượng bên ngoài tiến trình |
| Opaque Activity | D:\Study\Exam\BPELProject\Ver6.0\BPEL\src\bpel\images\icons20\opaqueactivity.png | Là một Activity dạng dẫn xuất |
| Assign | D:\Study\Exam\BPELProject\Ver6.0\BPEL\src\bpel\images\icons20\assign.png | Dùng để khởi tạo hoặc gán giá trị cho các biến trong tiến trình BPEL |
| Validate | D:\Study\Exam\BPELProject\Ver6.0\BPEL\src\bpel\images\icons20\validate.png | Kiểm tra tính hợp lệ của các biến và các biểu thức dựa trên định nghĩa của nó (chẳng hạn định nghĩa dựa trên XML Schema, hay WSDL…) |
| Các Activity điều khiển và có cấu trúc (Structrer Activity) | | |
| If/Else | D:\Study\Exam\BPELProject\Ver6.0\BPEL\src\bpel\images\icons20\if.png | Định nghĩa cấu trúc điều kiện |
| Pick | D:\Study\Exam\BPELProject\Ver6.0\BPEL\src\bpel\images\icons20\pick.png | Định nghĩa cách lựa chọn phương án hành động (hành động nào sẽ được thực hiện khi sự kiện tương ứng mà nó quy định xảy ra, nếu không có sự kiện nào xảy ra trong một thời gian chỉ định trước thì hành động nào sẽ được thực hiện…) |
| While | D:\Study\Exam\BPELProject\Ver6.0\BPEL\src\bpel\images\icons20\while.png | Lặp lại một tiến trình con nào đó trong process ở dạng while |
| RepeatUntil | D:\Study\Exam\BPELProject\Ver6.0\BPEL\src\bpel\images\icons20\repeatuntil.png | Lặp lại một tiến trình con nào đó trong process ở dạng do..while |
| Foreach | D:\Study\Exam\BPELProject\Ver6.0\BPEL\src\bpelproject\icons\foreach.png | Định nghĩa vòng lặp để duyệt qua một tập hợp |
| Wait | D:\Study\Exam\BPELProject\Ver6.0\BPEL\src\bpelproject\icons\wait.png | Dừng tiến trình trong một khoản thời gian được thiết lập trước |
| Sequence | D:\Study\Exam\BPELProject\Ver6.0\BPEL\src\bpelproject\icons\sequence.png | Dùng để thiết lập tuần tự hoạt động của các Activity |
| Scope | D:\Study\Exam\BPELProject\Ver6.0\BPEL\src\bpelproject\icons\scope.png | Dùng để chia nhỏ tiến trình thành các activity có các nhiệm vụ liên quan với nhau (khi tiến trình trở nên phức tạp). |
| Các Activity dùng để quản lý lỗi và ngoại lệ (Faul Handle Activity) | | |
| Exit | D:\Study\Exam\BPELProject\Ver6.0\BPEL\src\bpelproject\icons\exit.png | Dừng tiến trình hiện tại |
| Throw | D:\Study\Exam\BPELProject\Ver6.0\BPEL\src\bpelproject\icons\throw.png | Ném ra một ngoại lệ |
| Rethrow | C:\Users\Blind Kight\Desktop\images\rethrow.bmp | Ném ra thông báo lỗi sau khi lỗi này đã được thụ lý trước đó |
| Compensate Scope | D:\Study\Exam\BPELProject\Ver6.0\BPEL\src\bpelproject\icons\compensatescope.png | Là activity dùng để thụ lý lỗi. Khi có lỗi xảy ra thì activity này sẽ được dùng để xử lý lỗi trên phạm vi được chỉ ra |
| Compensate | D:\Study\Exam\BPELProject\Ver6.0\BPEL\src\bpelproject\icons\compensatescope.png | Có chức năng cũng giống như activity Compensate Scope nhưng được dùng để thụ lý lỗi trên phạm vi tất cả các phạm vi liên quan |

Hình 1.15 Các Activity trong BPEL

Trong quá trình thực thi thì các tiến trình sẽ có thể xảy ra các ngoại lệ, chẳng hạng như gọi đến một web service đã chết hoặc lỗi do cú pháp và các biểu thức mà người dùng nhập vào.Để thụ lý các lỗi này thì BPEL dùng các Activity như Throw hay Rethrow để quăng các lỗi ra cho một Activity xử lý lỗi bên ngoài là Compensate Scope hay Compensate để xử lý các lỗi này.

Có hai loại sự kiện trong BPEL đó là: Message Event: Sự kiện xảy ra khi có nhận được một thông điệp. Alarm Event: Sự kiện xảy ra khi hết một thời gian chỉ định và thường lặp đi lặp lại. Khi một sự kiện xảy ra, thông thường tiến trình BPEL sẽ gọi một Reply activity để trả lại thông điệp nhận được, hoặc ngừng thể hiện của tiến trình hiện tại mà không xử lý lỗi hoặc ném ra một lỗi để gọi activity xử lý lỗi tương ứng.

## Tổng 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 service 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 service 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

## 2.1 Nền tảng Eclipse

### 2.1.1 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.

Download, cài đặt Eclipse: [*http://www.Eclipse.org/downloads/*](http://www.eclipse.org/downloads/)

### 2.1.2 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.

*EclipseSDK* bao 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.

## 2.2 Kiến trúc plug-in của Eclipse

### 2.2.1 Giới thiệu về plug-in và extension point

Plug-in còn được gọi là trình cắm, hay phần bổ trợ là một thành phần giúp phần mềm ứng dụng thêm những tính năng cụ thể. Nếu được hỗ trợ, Plug-in cho phép tùy biến các chức năng của một ứng dụng. Plug-in có thể chứa mã nguồn, tài nguyên hay cả hai.

Extension point là 1 cơ chế cho phép 1 plug-in có thể thêm các chức năng từ 1 plug-in khác.

Extension là một chức năng

Mỗi Plug-in có một hay nhiều extension point và có thể tùy ý khai báo một điểm mở rộng mới (extension poit). Mỗi plug-in phụ thuộc vào một hoặc nhiều plug-in khác.



Hình 2.2 Plug-in và extension point trong eclipse

Tất cả các plug-in được tích hợp vào Eclipse sẽ nằm trong thư mục plug-ins của Eclipse, tên các thư mục con tương ứng với tên các plug-in và version của nó.

Plug-in với tên org.eclipse.ant.core được tích hợp trong Eclipse như là một Ant Builder. Trong mỗi thư mục chứa plug-in đều có 1 file plug-in.xml, file này là 1 file manifest dùng để mô tả plug-in như: tên, version. Nó cũng liệt kê tất cả các thư viện yêu cầu và các extension point được sử dụng cũng như được định nghĩa trong plug-in. Những file thường thấy trong thư mục plug-in:

plug-in.xml – Mô tả plug in

plug-in.properties – Chứa đựng các thông số hay thuộc tính được tham chiếu bởi plug-in.xml

about.html – Thông tin về bản quyền

\*.jar – Code của plug-in

lib – Thư mục chứa các file thư viện .jar

icons – Thư mục chứa các hình ảnh

Trong Eclipse, plug-in phải sử dụng một trong số plug-in hệ thống (Plug-in system) như nguồn lưu trữ (projects, folders và các files), thành phần giao diện người dùng để thực hiện công việc (Workbench user interface components), các plug-in có sẵn, trình gỡ lỗi (debugging), công cụ phát triển Java (JDT), môi trường phát triển thành phần cắm thêm (PDE) [4]

PDE cho phép các nhà phát triển dễ cài đặt plug-in trong Eclipse hơn là thông qua một công cụ bên ngoài. PDE chính là một plug-in và dựa vào sự hỗ trợ cùng một cốt lõi để thực hiện nhiệm vụ của mình như mỗi plug-in khác. Eclipse extensions được chia làm 3 loại:

* Plug-in
* Fragment
* Feature

Plug-in là một công cụ bổ sung thêm chức năng để Eclipse sử dụng các điểm mở rộng. Chu trình sống của plug-in được điều khiển bởi Workbench. Plug-in được thêm vào Eclipse khi Eclipse khởi động mà không được load ngay lập tức, Plug-in chỉ được load khi cần thiết. Trong suốt quá trình load, Elipse sẽ gọi phương thức startup() và trước khi unload thì gọi shutdown(). Hai phương thức load và unload có thể được coi như là hàm khởi tạo init() và hàm hủy destroy().

Fragment được hình thành từ một phần của plug-in. Fragment có ích khi muốn sử dụng một phần của plug-in độc lập với phần còn lại của plug-in. Fragment được dùng để thêm các đặc điểm nào đó vào một plug-in đang có mà không cần build lại plug-in đó hoặc được dùng để cung cấp chức năng cho một nền tảng nào đó. Fragment gần giống với plug-in thông thường, chỉ khác ở một vài đặc điểm sau:

Thông tin mô tả plug-in được lưu trong tập tin fragment.xml thay vì plug-in.xml

Trong tập tin fragment.xml, nút gốc là <fragment> và nút này có 2 thuộc tính là plug-in-id và plug-in-version dùng để chỉ ra định danh và phiên bản của plug-in cha.

Fragment sẽ tự động kế thừa các nút <requires> của plug-in cha và có thể thêm các nút <requires> khác nếu cần thiết.

Feature là việc đóng gói một nhóm các plug-in có liên quan lại thành một sản phẩm tích hợp. Feature không có chứa code. Ví dụ: Java Development Tooling (JDT) là một feature project được tạo thành bởi các plug-in như: Java editor, debugger, và console. Tập tin đặc tả feature project là feature.xml, tập tin này chứa tham chiếu đến các plug-in và các tài nguyên khác của feature project, đồng thời chứa các thông tin về việc update, copyright và license.

Về mặt kỹ thuật, một plug-in là một Java Archive (JAR) độc lập và tự khởi tạo. Plug-in độc lập vì plug-in chứa mã và tài nguyên các plug-in khác cần để chạy. Plug-in tự khởi tạo bởi vì nó bao gồm các thông tin của plug-in, tài nguyên plug-in, công dụng của nó. Trong một plug-in, hai tập tin mô tả là MANIFEST.MF và Plug-in.xml.

### 2.2.2 Tập tin cấu hình (Manifest) của Plug-in

Thông tin mô tả chi tiết plug-in nằm trong tập tin manifest (plug-in.xml). Eclipse sử dụng tập tin manifest để tích hợp plug-in vào framework.

Tập tin manifest chứa những thông tin chung về plug-in bao gồm: tên plug-in, phiên bản, tên lớp, tên file JAR. Ngoài ra, tập tin manifest còn chứa danh sách những plug-in mà plug-in hiện tại sử dụng.



Hình 2.3 Tập tin plug-in manifest

### 2.2.3 Vòng đời của plug-in

Khi khởi động Eclipse sẽ xem trong thư mục plug-ins và tạo 1 danh sách các plug-in, danh sách này được gọi là plug-in registry (danh sách được tạo dựa trên các file manifest). Plug-in được load khi chúng ta gọi nó và chỉ unload khi Eclipse đóng.

## 2.3 Mô hình và Eclipse EMF

### 2.3.1 Mô hình dữ liệu

Mô hình dữ liệu (Data model) hay đôi khi còn được gọi là mô hình miền dữ liệu (Domain model), biểu diễn dữ liệu mà chúng ta làm việc trên đó. Ví dụ mô hình dữ liệu của một ứng dụng đặt vé máy bay trực tuyến phải có một số đối tượng như Hành khách (Person), Chuyến bay (Flight), Đặt vé (Booking)…

Phương pháp tốt nhất là mô hình hóa mô hình dữ liệu của một ứng dụng độc lập với logic của ứng dụng hay với giao diện người sử dụng. Điều này dẫn đến việc các lớp (class) sẽ không tồn tại logic và có nhiều thuộc tính. Ví dụ, đối tượng Person sẽ có các thuộc tính như firstName, lastName, Address…

### 2.3.2 Eclipse Modeling Framework (EMF)

Eclipse Modeling Framework (EMF) là một tập các plug-in có thể sử dụng để mô hình hóa dữ liệu và tạo ra mã nguồn hay xuất ra các định dạng khác dựa trên mô hình. EMF là có sự phân biệt giữa meta-model và các mô hình thực tế. Nó là một mô hình cụ thể thể hiện của meta-model.

EMF cho phép các lập trình viên tạo ra meta-model bằng nhiều phương tiện khác nhau, ví dụ như XMI, Java annotations, UML hay XML Scheme. Nó cũng cho phép duy trì mô hình dữ liệu bằng cách sử dụng định dạng dữ liệu XML Metadata Interchange.

### 2.3.3 Tạo dữ liệu từ mô hình EMF

Thông tin lưu trữ trong mô hình EMF có thể được sử dụng để tạo ra các đầu ra mong muốn.

Trường hợp sử dụng thông thường của EMF đó là từ các meta-data (siêu dữ liệu) biểu diễn mô hình dữ liệu của ứng dụng và có thể sử dụng các chức năng của EMF để tạo ra các class Java tương ứng từ mô hình đó.

Lưu ý: EMF framework không chỉ giới hạn ở việc xuất ra các class Java, một số định dạng khác có thể được hỗ trợ và hoặc tự định nghĩa bởi người sử dụng.

Nói cách khác, mô hình EMF có thể sử dụng để tạo ra output, hoặc có thể biên dịch tại thời điểm chạy chương trình.

### 2.3.4 Meta-Models – Ecore và Genmodel

Như thảo luận trước đó, EMF là một meta-model. Mô hình này gồm hai phần chính: phần ecore và genmodel.

Phần ecore chứa thông tin về các class được định nghĩa

Phần genmodel chứa các thông tin thêm để tạo mã nguồn, như đường dẫn và file thông tin. Genmodel cũng chứa một số tham số điều khiển đến việc tạo ra mã nguồn

### 2.3.5 Mô tả file Ecore

Ecore cho phép định nghĩa các thành phần sau:

EClass: biểu diễn lớp, với không hay nhiều thuộc tính và không hay nhiều tham chiếu

EAttribute : biểu diễn một thuộc tính với tên và kiểu

EReference: biểu diễn liên kết giữa hai lớp. Nó có cờ chỉ thị để biểu diễn ràng buộc và lớp tham chiếu mà nó trỏ đến.

EDataType: biểu diễn kiểu của thuộc tính, ví dụ: int, float, hay java.util.Date…

Mô hình Ecore biểu diễn các đối tượng gốc của mô hình với các thành phần con biểu diễn các gói (packages), class và các thuộc tính.

### 2.3.6 Ưu điểm của EMF

EMF định nghĩa mô hình dữ liệu một cách rõ ràng. Chức năng tạo mã nguồn cũng có thể được điều chỉnh để phù hợp với sự thay đổi của mô hình. EMF cung cấp giao diện và phương tiện để tạo đối tượng, do đó nó khiến ứng dụng gọn gàng hơn.

### 2.3.7 UML

EMF cho phép tạo các lược đồ UML. The Unified Modeling Language (UML) là một ngôn ngữ đồ họa trong thiết kế phần mềm. Các sơ đồ khối cho UML là ở dạng lược đồ. UML chia lược đồ thành hai loại: lược đồ cấu trúc và lược đồ hành vi.

Phiên bản mới nhất của UML là UML 2 bổ sung thêm chức năng cho phép bắt được hành vi của hệ thống, hỗ trợ kiến trúc MDA (Model driving architecture). MDA có mục tiêu là tự động tạo ra chương trình ứng dụng từ một số mô hình.

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

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

## 3.2 Điều phối dịch vụ Web

## 3.3 Xây dựng Bus dịch vụ

## 3.4 Xây dựng kiến trúc “plug-and-play” dựa trên SOA

# Phần kết luận và hướng phát triển

[bpel\_tk1] OASIS, *Web Services Business Process Execution Language Version 2.0*, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html, 2007.

[4] Carlos Valcarcel (2005), Eclipse KickStart (Ver 3.0)

1. Extensible Markup Language, http://www.w3.org/XML/ [↑](#footnote-ref-1)