

**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**HỒ BẢO THANH 0112030
NGUYỄN HOÀNG LONG 0112141**

**NGHIÊN CỨU
KIẾN TRÚC HƯỚNG DỊCH VỤ
(SERVICE-ORIENTED ARCHITECTURE)
VÀ ỨNG DỤNG**

LUẬN VĂN CỬ NHÂN TIN HỌC

GIÁO VIÊN HƯỚNG DẪN

TH.S TRẦN MINH TRIẾT

Thành phố Hồ Chí Minh - 2005

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

HỒ BẢO THANH 0112030
NGUYỄN HOÀNG LONG 0112141

NGHIÊN CỨU
KIẾN TRÚC HƯỚNG DỊCH VỤ
(SERVICE-ORIENTED ARCHITECTURE)
VÀ ỨNG DỤNG

Chuyên ngành: CÔNG NGHỆ PHẦN MỀM

LUẬN VĂN CỬ NHÂN TIN HỌC

GIÁO VIÊN HƯỚNG DẪN:
TH.S TRẦN MINH TRIẾT

Thành phố Hồ Chí Minh - 2005

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

Khoa CNTT - ĐHKHTN TP.HCM

Khoa CNTT - ĐHKHTN TP.HCM

NHẬN XÉT CỦA GIÁO VIÊN PHẢN BIỆN

Lời cảm ơn

Chúng em chân thành cảm ơn Khoa Công Nghệ Thông Tin, trường Đại Học Khoa Học Tự Nhiên, Đại học Quốc gia Tp. Hồ Chí Minh đã tạo điều kiện thuận lợi cho chúng em trong quá trình học tập và thực hiện đề tài tốt nghiệp.

Chúng em xin nói lên lòng biết ơn sâu sắc đối với Th.S Trần Minh Triết. Chúng em xin chân thành cảm ơn Thầy đã luôn quan tâm, tận tình hướng dẫn chúng em trong quá trình học tập, nghiên cứu và thực hiện đề tài.

Chúng em xin chân thành cảm ơn quý Thầy Cô trong Khoa Công Nghệ Thông Tin đã tận tình giảng dạy, trang bị cho em những kiến thức quý báu trong suốt quá trình học tập và thực hiện đề tài. Chúng em cũng xin gửi lòng biết ơn đến thầy cô và bạn bè trong lớp đã giúp đỡ, động viên tinh thần chúng em rất nhiều trong suốt quá trình thực hiện luận văn này.

Chúng em nhớ mãi công ơn gia đình đã chăm sóc, động viên và tạo mọi điều kiện thuận lợi cho chúng em hoàn thành tốt luận văn này.

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 cảm ơn và mong luôn nhận được những tình cảm chân thành của tất cả mọi người.

Tp. Hồ Chí Minh, tháng 7 năm 2005

Hồ Bảo Thanh - Nguyễn Hoàng Long

Mục lục

Chương 1 TỔNG QUAN.....	1
1.1. Thực trạng hiện tại	1
1.2. Phân tích, đánh giá một số mô hình kiến trúc phân tán hiện tại	3
1.3. Các vấn đề phát sinh, nguyên nhân và biện pháp khắc phục.....	6
Chương 2 GIỚI THIỆU VỀ KIẾN TRÚC HƯỚNG DỊCH VỤ (SERVICE-ORIENTED ARCHITECTURE).....	10
2.1. Kiến trúc hướng dịch vụ là gì ?	10
2.2. Bốn nguyên tắc chính của hệ thống SOA.....	11
2.2.1. Sự phân định ranh giới rạch ròi giữa các dịch vụ	11
2.2.2. Các dịch vụ tự hoạt động.....	12
2.2.3. Các dịch vụ chia sẻ lược đồ.....	12
2.2.4. Tính tương thích của dịch vụ dựa trên chính sách.....	12
2.3. Các tính chất của một hệ thống SOA	12
2.3.1. Loose coupling.....	12
2.3.2. Sử dụng lại dịch vụ	14
2.3.3. Sử dụng dịch vụ bất đồng bộ.....	14
2.3.4. Quản lý các chính sách	14
2.3.5. Coarse granularity.....	15
2.3.6. Khả năng cộng tác.....	17
2.3.7. Tự động dò tìm và ràng buộc động	17
2.3.8. Tự hồi phục	18
2.4. Lợi ích của SOA.....	19
2.5. Một số mô hình triển khai SOA.....	23
2.6. Kiến trúc phân tầng chi tiết của SOA.....	26
2.6.1. Tầng kết nối	26
2.6.2. Tầng orchestration	27
2.6.3. Tầng ứng dụng tổng hợp.....	28
Chương 3 XÂY DỰNG HỆ THỐNG SOA.....	31
3.1. Những thách thức khi xây dựng hệ thống SOA	31
3.2. Xây dựng hệ thống SOA.....	34
3.2.1. Giới thiệu bài toán	34
3.2.2. Một số khái niệm	35
3.2.3. Các bước xây dựng hệ thống SOA.....	38
3.3. Triển khai SOA trong thực tế.....	46
3.3.1. Các đặc trưng chính về kinh doanh.....	47
3.3.2. Các đặc trưng chính về công nghệ	48
3.3.3. Các chuẩn mở.....	50
3.3.4. Kiến trúc hướng dịch vụ và Thương mại điện tử theo yêu cầu.....	50

Chương 4 SOA VÀ VẤN ĐỀ BẢO MẬT.....	52
4.1. Các thách thức về bảo mật trong hệ thống SOA.....	52
4.1.1. Đặt vấn đề.....	52
4.1.2. Các vấn đề bảo mật liên quan cần quan tâm.....	53
4.2. Giới thiệu về kiến trúc bảo mật hướng dịch vụ.....	55
4.2.1. Một số yêu cầu đặt ra của kiến trúc.....	55
4.2.2. Khái niệm về kiến trúc bảo mật hướng dịch vụ SOSA (service-oriented security architecture).....	58
4.2.3. Kiến trúc bảo mật hướng dịch vụ SOSA.....	60
4.3. Giới thiệu một số chuẩn về bảo mật trong XML.....	65
4.3.1. WS-Security.....	66
4.3.2. XML-Signature.....	67
4.3.3. XML-Encryption.....	67
4.3.4. XML Key Management Specification.....	67
4.3.5. Security Assertion Markup Language (SAML).....	67
4.4. Khai thác tính năng bảo mật web service của bộ thư viện WSE (Web Services Enhancements).....	68
4.4.1. Những tính năng chính của bộ thư viện WSE.....	68
4.4.2. Kiến trúc của WSE.....	71
Chương 5 SOA VÀ VẤN ĐỀ TÍCH HỢP.....	73
5.1. Giới thiệu về Enterprise Application Integration.....	73
5.1.1. Hiện trạng.....	73
5.1.2. Một số lý do khiến các tổ chức doanh nghiệp phải quan tâm đến vấn đề tích hợp (xét về mặt nghiệp vụ).....	74
5.1.3. Các vấn đề kỹ thuật gặp phải trong tích hợp hệ thống.....	75
5.1.4. Các yêu cầu cho một giải pháp tích hợp.....	76
5.1.5. Việc tích hợp có thể được áp dụng ở nhiều tầng khác nhau.....	76
5.2. Phân tích một số kỹ thuật tích hợp sử dụng Middleware.....	78
5.2.1. Khái niệm middleware.....	78
5.2.2. Các sản phẩm Middleware sử dụng trong tích hợp hệ thống.....	78
5.3. SOA và web service giải quyết vấn đề tích hợp như thế nào.....	82
5.3.1. Công nghệ XML và web service.....	82
5.3.2. Web services integration (WSI) và Service-oriented integration (SOI).....	84
5.4. Ứng dụng SOA và web service để tích hợp các hệ thống được xây dựng trên .NET và J2EE.....	87
5.5. Ứng dụng SOA và web service trong việc tích hợp các hệ thống cũ.....	90
Chương 6 SOA VÀ QUẢN LÝ TIẾN TRÌNH NGHIỆP VỤ.....	95
6.1. Một số khái niệm cơ bản về Quản lý tiến trình nghiệp vụ.....	95
6.1.1. Tiến trình nghiệp vụ.....	95
6.1.2. Quản lý tiến trình.....	96
6.1.3. Hệ quản lý tiến trình:.....	97
6.2. Quản lý tiến trình, SOA và Web Service.....	98
6.2.1. Quản lý tiến trình, SOA và Web Service được kết hợp thế nào.....	99
6.2.2. Phân tích một ví dụ kết hợp Quản lý tiến trình, SOA và web service.....	102

6.3	Thiết kế tiến trình	108
6.3.1	Orchestration và Choreography.....	108
6.3.2	Các yêu cầu kỹ thuật khi thiết kế tiến trình.....	110
6.3.3	Giới thiệu một số ngôn ngữ đặc tả tiến trình.....	112
Chương 7	ỨNG DỤNG “SOA SUITE”	125
7.1	Giới thiệu.....	125
7.1.1	Ứng dụng “SOA Suite”	125
7.1.2	Các thành phần của SOA Suite	126
7.2	ServiceBus.....	126
7.2.1	Vai trò chức năng của ServiceBus	126
7.2.2	ServiceBus và cơ sở tri thức.....	129
7.2.3	Các thành phần của ServiceBus:.....	130
7.2.4	Cơ chế hoạt động của ServiceBus.....	134
7.2.5	ServiceBus tích hợp với IIS.....	136
7.3	BpelEngine.....	136
7.3.1	Kiến trúc của BpelEngine.....	136
7.3.2	Các bước triển khai một business process trong BpelEngine.....	144
Chương 8	THÀNH PHẦN BPEL DESIGNER CỦA SOA SUITE	145
8.1	Giới thiệu.....	145
8.2	Chức năng.....	145
8.2.1	Tạo mới, chỉnh sửa, thiết kế một tiến trình	145
8.2.2	Chức năng kết xuất tiến trình ra file ảnh.....	145
8.2.3	Chức năng triển khai một tiến trình mới lên server	146
8.3	Thiết kế cài đặt	146
8.3.1	Cấu trúc chương trình.....	146
8.3.2	Giao diện chương trình.....	147
8.4	Hướng dẫn sử dụng.....	164
8.4.1	Thiết kế một tiến trình	164
8.4.2	Triển khai một tiến trình.....	169
Chương 9	ỨNG DỤNG SOA ĐỂ THIẾT KẾ MỘT SỐ TIẾN TRÌNH	170
9.1	Tiến trình dịch tự động đa ngôn ngữ	170
9.1.1	Mô tả	170
9.1.2	Sơ đồ	171
9.1.3	Mô tả luồng xử lý.....	172
9.2	Tiến trình thu thập thông tin từ bên ngoài.....	172
9.2.1	Mô tả	172
9.2.2	Sơ đồ	173
9.2.3	Mô tả luồng xử lý.....	173
9.3	Tiến trình chấm thi tự động qua mạng	174
9.3.1	Mô tả	174
9.3.2	Sơ đồ	175
9.3.3	Mô tả luồng xử lý.....	176

Chương 10 Kết luận.....	177
10.1 Một số kết quả đạt được.....	177
10.2 Hướng phát triển	178
Tài liệu tham khảo.....	180
Phụ lục A ĐẶC TẢ NGÔN NGỮ BPEL V1.1	182
A.1 Định nghĩa một tiến trình nghiệp vụ (business process).....	182
A.1.1 Cấu trúc của một tiến trình nghiệp vụ:.....	182
A.1.2 Chu kỳ sống của một tiến trình nghiệp vụ	187
A.2 Partner, Partner Link Type, và Partner Link	189
A.2.1 Partner	189
A.2.2 Partner Link Type	190
A.2.3 Partner Link.....	190
A.3 Xử lý dữ liệu.....	191
A.3.1 Biểu thức	191
A.3.2 Variable (biến)	193
A.4 Phép gán	195
A.5 Các xử lý cơ bản.....	197
A.5.1 Các thuộc tính cơ sở của mỗi xử lý.....	197
A.5.2 Các thành phần cơ sở của mỗi xử lý	198
A.5.3 Gọi một phương thức của Web Service (Invoke).....	198
A.5.4 Cung cấp các phương thức của Web Service.....	200
A.5.5 Cập nhật nội dung của biến	201
A.5.6 Báo lỗi (signaling fault).....	201
A.5.7 Waiting.....	202
A.5.8 Không làm gì cả.....	202
A.6 Các xử lý có cấu trúc (structure activity)	203
A.6.1 Sequence	203
A.6.2 Switch.....	204
A.6.3 While.....	205
A.6.4 Pick.....	206
A.6.5 Flow.....	207
A.7 Scope	215
A.7.1 Xử lý dữ liệu data và Partner Link	216
A.7.2 Xử lý lỗi trong tiến trình nghiệp vụ.....	216
A.7.3 Compensation handler	217
A.7.4 Xử lý lỗi	219
A.7.5 Xử lý sự kiện.....	221
Phụ lục B SOA VÀ WEB SERVICES	225
B.1 Kiến trúc Web services	225
B.2 Các đặt trưng của Web services.....	227
B.2.1 Loosely coupled.....	227
B.2.2 Tính đóng gói.....	227

B.2.3	Contracted	227
B.2.4	Giao thức chuẩn	227
B.2.5	Tự định nghĩa	228
B.2.6	Tìm kiếm và triệu gọi động	228
B.3	Lợi ích của Web services	228
B.4	SOAP	230
B.5	WSDL	231
B.6	UDDI	234
B.7	Một số chuẩn Web services mới	238
B.8	SOA và Web Service	238
Phụ lục C	ĐỊNH DẠNG CÁC FILE TRONG PROJECT BPEL DESIGNER	243

Danh sách hình

Hình 1-1 – Tích hợp dạng điểm nổi điểm.....	2
Hình 1-2 – Các thành phần của đối tượng CORBA.....	4
Hình 1-3 – Mô hình tương tác của đối tượng EJB.....	5
Hình 1-4 – Mô hình tương tác của các đối tượng DCOM.....	6
Hình 1-5 – Thực trạng cơ sở hạ tầng IT của hầu hết các tổ chức hiện nay.....	8
Hình 2-1 – Sơ đồ cộng tác trong SOA.....	10
Hình 2-2 - Tính chất loose-coupling.....	13
Hình 2-3 – Các đối tượng fine-grained.....	16
Hình 2-4 – Các đối tượng coarse-grained.....	16
Hình 2-5 – Các mức độ granularity.....	17
Hình 2-6 - Mô hình service registry.....	23
Hình 2-7 – Mô hình service broker.....	24
Hình 2-8 – Mô hình service bus.....	25
Hình 2-9 – Mô hình service bus phân tán.....	25
Hình 2-10 – Kiến trúc phân tầng của hệ thống SOA.....	26
Hình 2-11 – Một công thông tin cung cấp thông tin trong một vùng nhìn duy nhất.....	29
Hình 2-12 – Các portlet truy xuất dữ liệu từ nhiều nguồn khác nhau được cung cấp dưới dạng dịch vụ.....	29
Hình 3-1 - Các bước cần thực hiện khi triển khai một hệ thống SOA.....	35
Hình 3-2 – Phân rã domain thành một dãy các vùng chức năng liên quan.....	38
Hình 3-3 – Sơ đồ sử dụng của hệ thống bán hàng.....	39
Hình 3-4 – Sơ đồ các use case và quy trình nghiệp vụ.....	39
Hình 3-5 – Ví dụ về mô hình goal-service.....	41
Hình 3-6 – Các use case nghiệp vụ được “gắn” trên hệ thống con.....	43
Hình 3-7 – Phân bổ dịch vụ.....	44
Hình 3-8 – Mẫu đặc tả thành phần.....	45
Hình 3-9 – Chọn lựa một giải pháp thực thi thích hợp.....	46

Hình 3-10 – Sơ đồ tổng quan về thương mại điện tử theo yêu cầu.....	46
Hình 4-1 – Kiến trúc bảo mật hướng dịch vụ (SOSA).....	60
Hình 4-2 – Cấu trúc phân tầng của Standard-based Security Info Exchange Platform.....	61
Hình 4-3 – Security Token Service.....	63
Hình 4-4 – Các tầng kỹ thuật bên dưới của một hệ thống bảo mật.....	66
Hình 4-5 – Xác nhận số một thông điệp.....	69
Hình 4-6 – Mã hóa một thông điệp.....	69
Hình 4-7 – Điều phối thông điệp SOAP.....	70
Hình 4-8 – Cơ chế sử dụng bộ lọc của WSE.....	72
Hình 5-1 – Vai trò cơ bản của middleware.....	78
Hình 5-2 – Cơ chế hàng đợi.....	79
Hình 5-3 – Cơ chế Publish/Subscribe.....	80
Hình 5-4 - Remote Procedure Call.....	80
Hình 5-5 – Distributed Object Model.....	81
Hình 5-6 – Sử dụng web service trong vấn đề tích hợp.....	83
Hình 5-7 – Web services integration (WSI).....	85
Hình 5-8 – Service-oriented integration (SOI).....	86
Hình 5-9 – Sự khác nhau giữa kiến trúc .NET và J2EE.....	88
Hình 5-10 – Vai trò của WSDL trong liên kết các hệ thống.....	89
Hình 5-11 – WSDL mô tả cách các thông điệp SOAP được xử lý.....	89
Hình 5-12 – Dịch vụ hóa một CORBA server.....	92
Hình 6-1 – Minh họa một tiến trình nghiệp vụ.....	95
Hình 6-2 – Các thành phần của một hệ thống quản lý tiến trình nghiệp vụ.....	97
Hình 6-3 – Thực trạng “không đồng nhất” của các hệ thống doanh nghiệp.....	99
Hình 6-4 – Tầng dịch vụ dựa trên mô hình SOA với công nghệ Web Service.....	100
Hình 6-5 – Tầng tiến trình nghiệp vụ sử dụng tầng dịch vụ.....	100
Hình 6-6 – Các hệ thống BPM khi không có tầng services.....	101
Hình 6-7 – Ví dụ về các hệ thống cũ cần được service hóa.....	103
Hình 6-8 – Mô hình dữ liệu, dịch vụ và tiến trình.....	103

Hình 6-9 – Xây dựng các service đơn vị.....	105
Hình 6-10 – Xây dựng các dịch vụ tích hợp.....	106
Hình 6-11 – Xây dựng các tiến trình nghiệp vụ.....	107
Hình 6-12 – Cung cấp các tiến trình nghiệp vụ cho người dùng.....	108
Hình 6-13 – Sự khác nhau giữa orchestration và choreography.....	109
Hình 6-14 – Tiến trình cung cấp khả năng tái sử dụng.....	111
Hình 6-15 – Tiến trình được định nghĩa bằng WSFL.....	112
Hình 6-16 – Sơ đồ luồng trong WSFL.....	113
Hình 6-17 – Sơ đồ tổng thể trong WSFL.....	113
Hình 6-18 – Liên kết giữa các xử lý trong WSFL.....	114
Hình 6-19 – Một ví dụ về các luồng thông điệp trong tương tác giữa các service.....	115
Hình 6-20 – Minh họa Web Service Choreography Interface.....	116
Hình 6-21 – Một tiến trình được mô tả bằng WSCI.....	117
Hình 6-22 – Một tiến trình đặc tả bởi ngôn ngữ BPEL.....	119
Hình 6-23 – Mẫu xử lý WP2 và WP3.....	120
Hình 6-24 – Mẫu xử lý WP4 và WP5.....	121
Hình 6-25 – Mẫu xử lý WP8.....	122
Hình 6-26 – Mẫu giao tiếp CP1 và CP2.....	123
Hình 6-27 – Mẫu giao tiếp CP4.....	124
Hình 7-1 – Các thành phần của SOASuite.....	126
Hình 7-2 – Môi trường trao đổi thông điệp SOAP của serviceBUS.....	127
Hình 7-3 – Liên kết giữa ServiceBus và WSE Messaging.....	127
Hình 7-4 – Quy trình xử lý thông điệp của serviceBUS.....	135
Hình 7-5 – Sơ đồ kiến trúc của BpelEngine.....	137
Hình 7-6 – Tạo các đối tượng activity implementation.....	139
Hình 7-7 – Sơ đồ điều phối thông điệp của engine.....	141
Hình 7-8 – Sơ đồ phân cấp của các xử lý.....	143
Hình 8-1 – Cấu trúc thư mục dùng triển khai một tiến trình BPEL.....	169
Hình 9-1 – Tiến trình dịch tự động.....	171

Hình 9-2 – Tiến trình thu thập thông tin và dịch tự động.....	173
Hình 9-3 – Tiến trình chấm thi tự động.....	175
Hình B-1 – Một SOAP Operation đơn giản.....	231
Hình B-2 – Cấu trúc thông điệp SOAP.....	231

Danh sách các thuật ngữ và khái niệm

- **Service Consumer** : người sử dụng dịch vụ ở đây có thể là một ứng dụng, một dịch vụ hoặc là các module phần mềm khác yêu cầu sử dụng dịch vụ. Đây là thực thể thực thi quá trình định vị dịch vụ thông qua service registry, liên kết với dịch vụ và thực thi các chức năng của dịch vụ. Người sử dụng dịch vụ thực thi chức năng dịch vụ bằng cách một gửi yêu cầu theo đúng định dạng được mô tả trong hợp đồng.
- **Service provider** : nhà cung cấp dịch vụ ở đây là một dịch vụ chấp nhận và xử lý những yêu cầu từ người sử dụng dịch vụ. Nó có thể là một hệ thống mainframe, một thành phần hoặc các dạng phần mềm khác xử lý yêu cầu dịch vụ. Nhà cung cấp gửi hợp đồng lên service registry để những người sử dụng dịch vụ có thể truy cập đến nó.
- **Service Registry** : service registry là “thư mục” trên mạng chứa tất cả các dịch vụ đăng ký. Service registry chấp nhận và lưu trữ các hợp đồng gửi đến từ nhà cung cấp dịch vụ và cung cấp các hợp đồng tùy theo yêu cầu của người sử dụng dịch vụ.
- **Service contract** : một hợp đồng (contract) là một đặc tả về cách thức bên sử dụng dịch vụ trao đổi liên lạc với bên cung cấp dịch vụ. Nó chỉ rõ ra định dạng yêu cầu và đáp trả của dịch vụ.
- **Distributed computing** : một dạng tính toán trong đó dữ liệu và ứng dụng được phân tán trên nhiều máy hoặc hệ thống tách biệt nhưng lại được liên kết và tích hợp thông qua các dịch vụ mạng và chuẩn tích hợp để mà chúng có thể thực thi chức năng như trong một môi trường thống nhất.
- **Enterprise application** : một sản phẩm phần mềm được thiết kế để tích hợp các hệ thống máy tính bên trong doanh nghiệp lại với nhau. Mục tiêu là để tích hợp các xử lý nghiệp vụ chính (ví dụ như bán lẻ, kiểm toán, tài chính, quản lý nhân sự, tồn kho và sản xuất). Enterprise application được dùng rộng rãi trong

bối cảnh cần liên hệ chặt chẽ với nhà cung cấp, với đối tác kinh doanh và với khách hàng.

- **Loose coupling:** đây là một khái niệm trong tích hợp ứng dụng. Hai thành phần trao đổi thông tin không kết nối trực tiếp với nhau mà qua một thành phần trung gian được đặc tả rõ ràng từ trước. Các thành phần tham gia phải đảm bảo một cơ chế ngữ nghĩa chung để các thông điệp chứa bên trong chúng một ngữ nghĩa đủ để tự mô tả chính mình.
- **Tight coupling :** ngược lại với loose coupling.
- **Interface :** thành phần giao tiếp.
- **Coarse-grained :** mô tả mức độ gom nhóm xử lý của một thành phần xử lý thông tin. Các thành phần có tính coarse-grained thường truyền nhận và xử lý theo từng khối dữ liệu có thông tin ngữ cảnh lớn và số lần trao đổi thông tin trong một giao tác là ít. Xem Hình 2-4 để hiểu rõ hơn.
- **Fine-grained :** ngược lại với coarse-grained. Các thành phần có tính fine-grained truyền nhận và xử lý theo từng đơn vị nhỏ và có ngữ cảnh ngầm định, cần nhiều lần trao đổi thông tin trong một giao tác dẫn đến tăng băng thông sử dụng và kéo dài thời gian hồi đáp. Xem hình Hình 2-3 để hiểu rõ hơn.
- **Legacy system :** các hệ thống ứng dụng được cài đặt từ trước nhưng vẫn còn được sử dụng. Thông thường đây là những hệ thống sử dụng công nghệ đã lỗi thời nhưng vẫn quan trọng và còn hoạt động tốt. Đây là thành phần nền tảng cung cấp xử lý cho các dịch vụ hoạt động ở cấp cao hơn.
- **Granularity :** khái niệm mô tả độ phức tạp của tiến trình hoặc dịch vụ. Granularity được chia làm hai loại là “fine-grained” và “coarse-grained”. Khái niệm granularity được hiểu một cách trừu tượng không phân biệt hai loại trên và tùy ngữ cảnh mà có cảnh hiểu khác nhau.
- **Interoperability :** khả năng cộng tác, trao đổi thông tin giữa các hệ thống phân tán

- **Orchestration** : sự kết hợp, điều phối hoạt động của những Web Service theo một quy trình mong muốn.
- **Choreography** : tương tự như orchestration nhưng chỉ tập trung vào quan hệ giữa các thành phần dịch vụ với nhau theo mô hình peer-to-peer.
- **Middleware** : Khái niệm middleware dùng để mô tả phần mềm liên kết các phần mềm khác lại với nhau.
- **BPEL** : Business Process Execution Language, một ngôn ngữ XML được thiết kế nhằm cho phép kết hợp các xử lý trên một môi trường tính toán phân tán lại với nhau theo một luồng xử lý hoặc quy trình nghiệp vụ có cấu trúc định trước.

MỞ ĐẦU

Ngày nay, công nghệ thông tin đang là nền công nghệ mũi nhọn trong chiến lược phát triển kinh tế, xây dựng đất nước của hầu hết các quốc gia. Các sản phẩm công nghệ thông tin đã và đang được ứng dụng rộng rãi trong mọi lĩnh vực của đời sống kinh tế, xã hội và hầu hết đều đem đến những giá trị thiết thực.

Đối tượng phục vụ chủ yếu của ngành công nghệ thông tin hiện nay chính là các tổ chức, các cơ sở doanh nghiệp,... Nhu cầu ứng dụng các sản phẩm của ngành công nghệ mũi nhọn này để hỗ trợ tin học hóa các qui trình nghiệp vụ, mà vốn từ trước đến nay chỉ được thực hiện một cách thủ công, đang ngày trở nên một nhu cầu cấp thiết. Các sản phẩm dạng này đều có chung các đặc điểm là độ phức tạp lớn, chi phí sản xuất và bảo trì cao, mức độ cụ thể còn tùy thuộc vào độ lớn của tổ chức cũng như là độ phức tạp của các qui trình nghiệp vụ cần xử lý.

Với sự phát triển của internet và với xu thế hội nhập chung của toàn thế giới, các tổ chức, các cơ sở doanh nghiệp cần bắt tay, phối hợp hoạt động và chia sẻ tài nguyên với nhau để nâng cao hiệu quả hoạt động. Lúc này các sản phẩm sẽ có độ phức tạp lớn hơn, từ đó 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 an ninh bảo mật, 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 các vấn đề trên, nhiều giải pháp đã được nghiên cứu và ứng dụng. Nhưng hầu hết các giải pháp này không giải quyết các khó khăn một cách triệt để và kết quả đạt được cũng không như mong đợi. Hiện nay, một giải pháp mới đang được cộng đồng công nghệ thông tin rất quan tâm, đó là “Kiến trúc hướng dịch vụ” (Service-oriented Architecture - SOA). Giải pháp này bước đầu đã được ứng dụng trong một số dự án và đều đem lại những kết quả khả quan. Và người ta tin rằng SOA có thể giải quyết tốt những thách thức đã nêu trên, và sẽ

là “xu thế trong tương lai”. Thế thì “Kiến trúc hướng dịch vụ” là gì? Cách giải quyết vấn đề cũng như là những lợi ích đạt được của kiến trúc này như thế nào?

Đây chính lý do để chúng em thực hiện đề tài

“NGHIÊN CỨU KIẾN TRÚC HƯỚNG DỊCH VỤ (SERVICE-ORIENTED ARCHITECTURE) VÀ ỨNG DỤNG”

Mục tiêu của đề tài

Đề tài sẽ tập trung vào tìm hiểu các vấn đề sau:

- Nghiên cứu các cơ sở lý thuyết của kiến trúc hướng dịch vụ (SOA) thông qua việc tìm hiểu khái niệm về “kiến trúc hướng dịch vụ”, các tính chất cùng với những lợi ích đạt được của hệ thống SOA.
- Tìm hiểu các vấn đề liên quan đến xây dựng hệ thống SOA, bao gồm những thách thức gặp phải, những nguyên tắc thiết kế và các bước cần thực hiện khi triển khai hệ thống SOA.
- Ứng dụng SOA trong xây dựng kiến trúc bảo mật hướng dịch vụ. Tìm hiểu một số chuẩn bảo mật trong XML và khai thác tính năng bảo mật web service của bộ thư viện lập trình WSE (Web Services Enhancements).
- Tìm hiểu về nhu cầu và các thách thức gặp phải trong việc tích hợp hệ thống. Từ đó, ứng dụng SOA và Web service để giải quyết vấn đề tích hợp.
- Tìm hiểu khái niệm về tiến trình nghiệp vụ, quản lý tiến trình, mối quan hệ của tiến trình nghiệp vụ trong hệ thống SOA. Xem xét các nguyên tắc thiết kế và khảo sát một số ngôn ngữ đặc tả tiến trình nghiệp vụ.
- Xây dựng ứng dụng **SOASuite** nhằm hỗ trợ trong việc thiết kế, xây dựng và triển khai hệ thống kiến trúc hướng dịch vụ. Ứng dụng cung cấp môi trường linh hoạt để quản lý các dịch vụ có trong hệ thống dựa trên cơ chế thông điệp. Ngoài ra, SOASuite còn cung cấp môi trường phát triển cho phép người dùng

có thể thiết kế, xây dựng và thực thi và quản lý các tiến trình nghiệp vụ từ những web services đã được xây dựng sẵn thông qua các thao tác kéo thả mà không cần viết mã lệnh xử lý.

Nội dung luận văn

Nội dung của luận văn được trình bày gồm:

- **Chương 1:** trình bày, phân tích về một số khó khăn của ngành công nghệ phần mềm hiện nay. Từ đó giới thiệu một số mô hình kiến trúc phân tán được xây dựng để giải quyết các khó khăn trên như là CORBA, EJB, DCOM.
- **Chương 2:** giới thiệu khái niệm về kiến trúc hướng dịch vụ SOA, các nguyên tắc cũng như là tính chất của hệ thống SOA; phân tích một số lợi ích đạt được và khảo sát một số mô hình của kiến trúc hướng dịch vụ. Chương này cũng trình bày về kiến trúc phân tầng của hệ thống SOA.
- **Chương 3:** trình bày các vấn đề liên quan đến việc xây dựng hệ hống SOA, bao gồm phân tích các thách thức gặp phải và xem xét qui trình các bước nên thực hiện khi triển khai hệ thống SOA.
- **Chương 4:** trình bày về các khó khăn gặp phải trong việc bảo vệ hệ thống SOA. Từ đó xem xét, phân tích một giải pháp đó là mô hình kiến trúc bảo mật hướng dịch vụ. Chương này cũng giới thiệu một số chuẩn bảo mật trong XML như WS-Security, XML-Signature, XML-Encryption, XML Key Management Specification, SAML và bộ thư viện WSE (Web Services Enhancements) hỗ trợ lập trình bảo mật web services.
- **Chương 5:** trình bày và phân tích về nhu cầu và một số khó khăn gây trở ngại trong vấn đề tích hợp hệ thống. Qua đó xem xét một số giải pháp được sử dụng trong tích hợp, bao gồm giải pháp sử dụng các sản phẩm middleware và giải pháp ứng dụng SOA và Web services: Web Service Integration (WSI) và Service-Oriented Integration (SOI). Sau đó, xem xét cụ thể giải pháp ứng dụng

SOA và Web services trong tích hợp các hệ thống xây dựng trên nền .NET và J2EE và trong tái sử dụng lại các hệ thống cũ.

- **Chương 6:** trình bày một số khái niệm liên quan về quản lý tiến trình. Phân tích mối quan hệ kết hợp giữa quản lý tiến trình, SOA và Web services. Xem xét các vấn đề liên quan đến thiết kế tiến trình nghiệp vụ. Ngoài ra, chương này cũng sẽ giới thiệu về một số ngôn ngữ đặc tả tiến trình hiện đang được sử dụng phổ biến, như là Web Service Flow Language (WSFL), XLANG, Web Service Choreography Interface (WSCI) và Business Process Execution Language For Web Service (BPEL4WS)
- **Chương 7:** giới thiệu tổng quát về ứng dụng SOASuite. Trình bày về hai thành phần ServiceBus và BpelEngine. ServiceBus cung cấp môi trường quản lý các dịch vụ dựa trên cơ chế thông điệp và BpelEngine cung cấp môi trường triển khai và thực thi cho các tiến trình nghiệp vụ.
- **Chương 8:** giới thiệu về thành phần thứ ba của SOASuite, bộ công cụ “BpelDesigner” cung cấp môi trường trực quan hỗ trợ người dùng xây dựng, thiết kế các tiến trình nghiệp vụ
- **Chương 9:** giới thiệu một số mẫu tiến trình được thiết kế bằng bộ công cụ BpelDesigner.
- **Chương 10:** trình bày một số kết luận và hướng phát triển của đề tài.

Chương 1

TỔNG QUAN

Nội dung của chương 1 trình bày về một số khó khăn của ngành công nghệ phần mềm hiện nay. Từ đó giới thiệu, phân tích các ưu khuyết điểm của một số mô hình kiến trúc phân tán được xây dựng để giải quyết các khó khăn trên như là CORBA, EJB, DCOM...

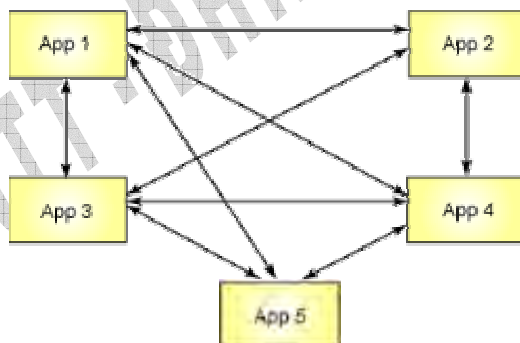
1.1 Thực trạng hiện tại

Phần mềm ngày nay đang ngày càng trở nên *phức tạp* và dường như đang vượt khỏi khả năng kiểm soát của các mô hình phát triển phần mềm hiện có. Albert Einstein đã nói :“Mọi việc nên thực hiện theo cách đơn giản đến mức có thể...” , một thực trạng đáng buồn là có rất nhiều hệ thống phần mềm được xây dựng với kiến trúc quá phức tạp, chi phí phát triển và bảo trì cao, đặc biệt là với các hệ thống phần mềm cao cấp. Hàng chục năm qua, nhiều kiến trúc phần mềm đã được xây dựng và triển khai nhằm giải quyết các vấn đề này. Thế nhưng độ phức tạp phần mềm vẫn cứ tiếp tục tăng và dường như đã trở nên vượt quá khả năng xử lý của các kiến trúc truyền thống.

Nguyên nhân khiến cho độ phức tạp của các hệ thống phần mềm không ngừng tăng cao như thế là do sự xuất hiện của *nhiều công nghệ mới* tạo nên môi trường *không đồng nhất*, trong khi nhu cầu về trao đổi, chia sẻ, tương tác giữa các hệ thống không thể đáp ứng được trong một môi trường như vậy. Làm sao có thể dung hòa được những cách biệt giữa cái cũ và cái mới? Các hệ thống cũ (legacy systems) cần được sử dụng lại thay vì phải gỡ bỏ và thay mới bởi vì chi phí thực hiện lại từ đầu chắc chắn sẽ cao hơn việc chi phí chuyển đổi cái cũ rất nhiều lần. Vấn đề này liên quan đến một khái niệm và cũng là một thách thức mà các tổ chức phải đối mặt, đó là “**tích hợp hệ thống**” (Enterprise Architecture Integration - EAI). Hiện các dự án dạng này đang được rất nhiều tổ chức quan tâm đến, với mức đầu tư về chi phí đang dẫn đầu so với các dạng dự án khác.

Một nguyên nhân khác cũng góp phần dẫn đến tình trạng khó khăn như thế chính là vấn đề *lập trình dư thừa và không thể tái sử dụng*. Hãy xét một ví dụ, một ngân hàng có nhiều chi nhánh khác nhau – mỗi chi nhánh có một hệ thống tách biệt và cần kết nối với các hệ thống khác của ngân hàng để phục vụ khách hàng được hiệu quả hơn. Giả sử rằng các hệ thống này đều được thiết kế rất tốt. Thế nhưng các hệ thống này được xây dựng trong những khoảng thời gian khác nhau, trong những dự án độc lập và khác nhau. Thông thường chức năng “lấy số liệu thống kê tài khoản” bị lặp lại trong mỗi hệ thống ATM, mỗi chi nhánh và trong hệ thống lưu trữ tài khoản, và ngay cả khi người dùng truy cập vào cùng một tài khoản trong cùng một cơ sở dữ liệu. Bây giờ nếu ngân hàng đó cần phát triển một hệ thống cung cấp dịch vụ gửi tiền hay cho vay tiền qua mạng để tăng chất lượng phục vụ cho các khách hàng. Hệ thống mới này sẽ được xây dựng thế nào? Nếu giải pháp chọn xây dựng lại hệ thống mới, thì lại tiếp tục mắc lại sai lầm trước đó: dư thừa, không đồng nhất ... Còn nếu chọn giải pháp là sử dụng lại các chức năng sẵn có, thì ta phải đối mặt với chuyện thiết lập các mối liên kết với toàn bộ các hệ thống trước.

Hầu như mọi tổ chức đều phải đối mặt với vấn đề tích hợp vì nhiều lý do, đặc biệt là trong thị trường ngày nay, sự thay đổi luôn diễn ra với tốc độ chóng mặt; có thể là mở rộng thêm chi nhánh, một hệ thống bán hàng mới, hoặc chỉ đơn giản là cần kết nối các hệ thống có sẵn. Nếu có n hệ thống ứng dụng cần được kết nối trực tiếp với nhau, thì cần $n*(n-1)$ kết nối, hoặc là interface.



Hình 1-1 – Tích hợp dạng điểm nối điểm

Tương tự, nếu có thêm một hệ thống ứng dụng thứ $(n+1)$ cần được tích hợp thêm vào hệ thống, thì nó đòi hỏi $2*n$ interface mới, bao gồm cả việc thu thập dữ liệu, kiểm thử, và bảo trì. Theo như Hình 1-1 trên thì 5 ứng dụng đòi hỏi 20 kết nối trực tiếp, một ứng dụng thứ 6 tích hợp thêm vào sẽ yêu cầu thêm 10 kết nối mới! Tệ hơn nữa, mã nguồn của các ứng dụng cũ phải được chỉnh sửa để thêm vào các kết nối, từ đó kéo theo chi phí kiểm thử, bảo trì.

Những vấn đề trước chưa giải quyết, mà nay các tổ chức lại phải đối mặt với những thách thức mới: đáp ứng nhanh chóng các sự thay đổi, giảm chi phí phát triển, tăng tính tương thích và khả năng tái sử dụng,... Tất cả đã tạo nên một áp lực nặng nề đối với các nhà phát triển phần mềm.

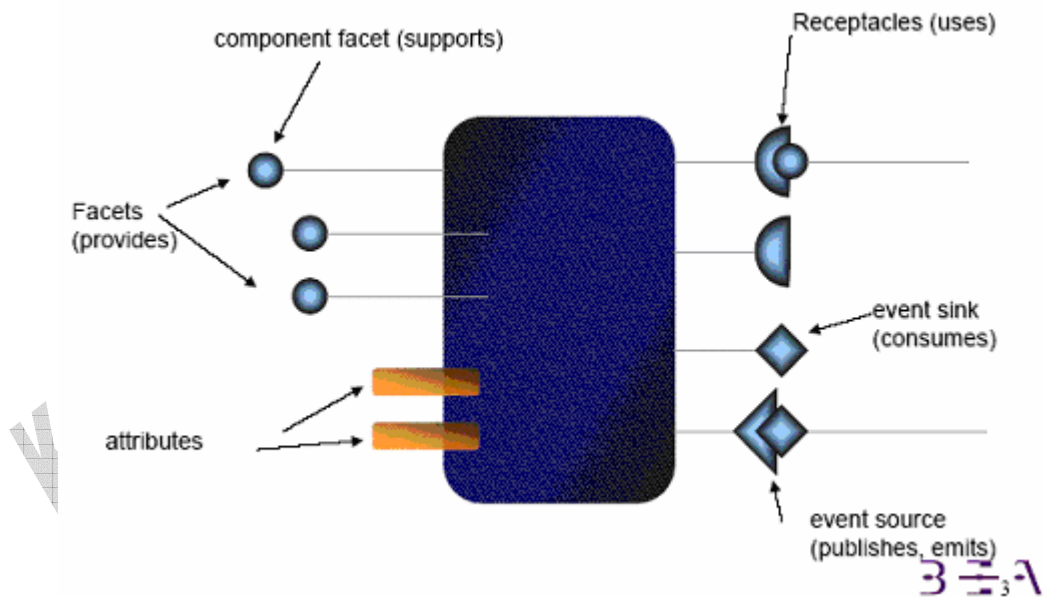
1.2 Phân tích, đánh giá một số mô hình kiến trúc phân tán hiện tại

Ba kiến trúc phân tán phổ biến nhất hiện nay là CORBA, DCOM và EJB. Các kiến trúc này là sự mở rộng của các hệ thống hướng đối tượng bằng cách cho phép phân tán các đối tượng trên mạng. Đối tượng đó có thể có không gian địa chỉ bên ngoài ứng dụng, hoặc ở một máy khác với máy chứa ứng dụng trong khi vẫn được tham chiếu sử dụng như một phần của ứng dụng.

- **CORBA – Common Object Request Broker Architecture:**

- ▶ CORBA được định nghĩa bởi Object Management Group (OMG), là một kiến trúc phân tán mở, độc lập nền tảng và độc lập ngôn ngữ.
- ▶ CORBA Component Model (CCM) là một cải tiến đáng kể nhằm định nghĩa các mô hình thành phần so với CORBA. Nó định nghĩa ra quy trình thiết kế, phát triển, đóng gói, triển khai và thực thi các thành phần phân tán. CCM định nghĩa khái niệm *Ports* cho các thành tố. Các port này được sử dụng để kết nối các thành phần có sẵn với nhau, tạo các hệ thống phân tán phức tạp hơn. Mỗi thành phần CCM có một đối tượng *Home* chịu trách nhiệm quản lý chu kỳ sống của đối tượng và được triển khai bên trong một trình chứa (container).

- Ưu điểm của CORBA là các lập trình viên có thể chọn bất kỳ ngôn ngữ, nền tảng phần cứng, giao thức mạng và công nghệ để phát triển mà vẫn thoả các tính chất của CORBA. Tuy nhiên CORBA sở một nhược điểm là nó là ngôn ngữ lập trình cấp thấp, rất phức tạp, khó học và cần một đội ngũ phát triển có kinh nghiệm. Ngoài ra các đối tượng CORBA cũng khó có thể tái sử dụng.



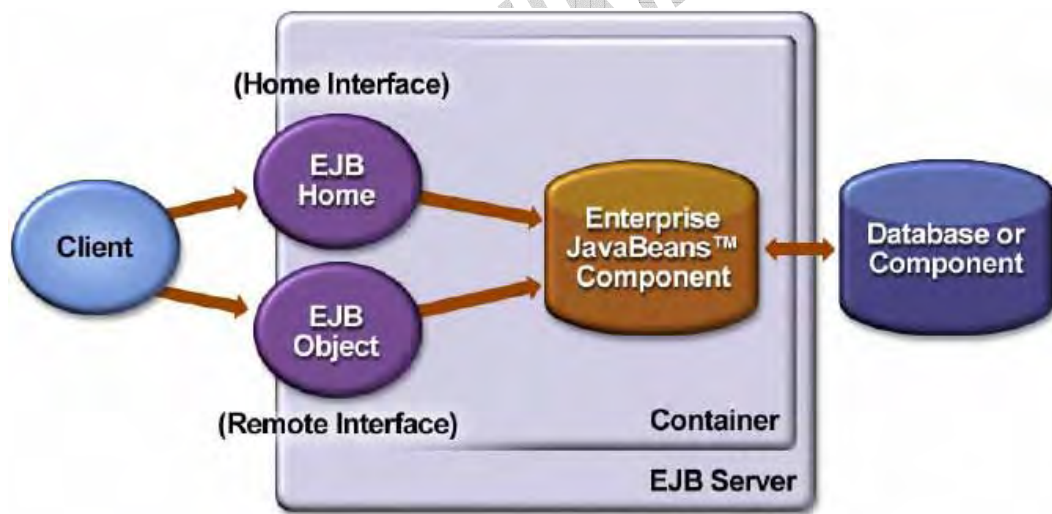
Hình 1-2 – Các thành phần của đối tượng CORBA

- **EJB - Enterprise Java Bean:**

- Kiến trúc EJB là một kiến trúc thành tố bên phía máy chủ dùng cho việc phát triển và triển khai các ứng dụng phân tán hướng đối tượng cỡ vừa và lớn.
- Kiến trúc EJB có 3 tầng với tầng đầu tiên là tầng trình diễn, tầng thứ hai là tầng xử lý nghiệp vụ, và tầng thứ ba là các tài nguyên như cơ sở dữ liệu máy chủ. Truyền thông giữa các đối tượng EJB thông qua Remote Method Invocation (RMI). Các client không bao giờ tương tác trực tiếp với các bean. Thay vì vậy chúng sẽ sử dụng các phương thức được định nghĩa trong các interface *Remote* và *Home*. Mỗi bean tồn tại bên trong trình chứa, chịu trách nhiệm việc tạo thể hiện mới, lưu trữ dữ liệu và các quản lý khác. Trình chứa sẽ triệu gọi các phương thức callback của mỗi thể hiện bean khi có sự kiện tương

ứng. Không giống như CCM, EJB không định nghĩa các port kết nối trực tiếp giữa các thành phần liên quan bởi vì mỗi bean bên trong trình chứa là một thực thể độc lập không có bất kỳ ràng buộc nào bên ngoài.

- EJB là một kiến trúc tốt cho việc tích hợp các hệ thống vì nó độc lập nền tảng nhưng nó cũng gặp vấn đề là nó không phải là một chuẩn mở, khả năng giao tiếp với các chuẩn khác vẫn còn hạn chế.

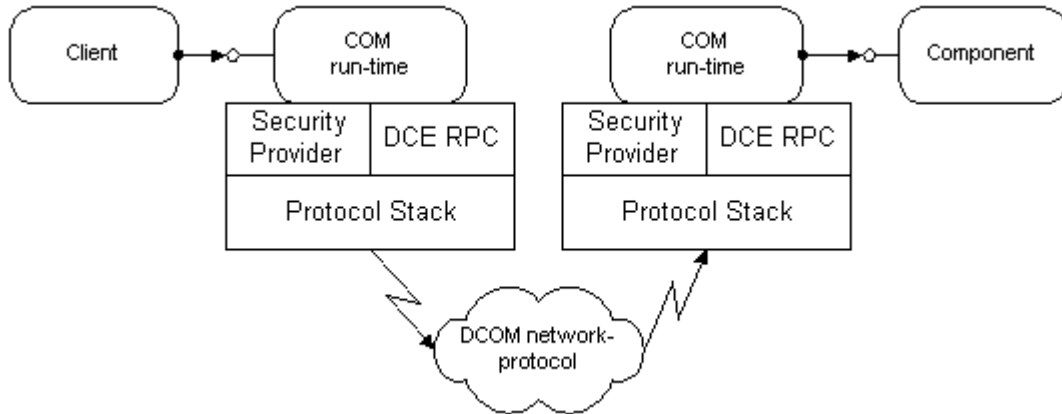


Hình 1-3 – Mô hình tương tác của đối tượng EJB

- **DCOM – Distributed Component Object Model:**

- DCOM là một mô hình phân tán dễ triển khai với chi phí thấp, hỗ trợ tight coupling giữa các ứng dụng và hệ điều hành. Mô hình Component Object Model (COM) định nghĩa cách thức các thành phần và client liên lạc trao đổi với nhau trên cùng một máy. DCOM mở rộng COM bằng cách sử dụng các giao thức mạng chuẩn khi cần trao đổi dữ liệu với máy khác trên mạng. DCOM hỗ trợ kết nối giữa các đối tượng và những kết nối này có thể được thay đổi lúc đang chạy. Các đối tượng DCOM được triển khai bên trong các gói nhị phân chứa các mã lệnh quản lý chu kỳ sống của đối tượng và việc đăng ký đối tượng.
- DCOM mang đến nhiều ưu điểm như tính ổn định, không phụ thuộc vị trí địa lý, quản lý kết nối hiệu quả và dễ dàng mở rộng, là một lựa chọn tốt cho các doanh nghiệp sử dụng công nghệ của Windows để chạy các ứng dụng có yêu

cầu cao về sự chính xác và ổn định. Tuy nhiên, các công nghệ của Microsoft có một nhược điểm lớn là chúng bị giới hạn trên nền tảng Windows.



Hình 1-4 – Mô hình tương tác của các đối tượng DCOM

Các kiến trúc trên đều hướng đến việc xây dựng một hệ thống “hướng dịch vụ” tuy nhiên chúng vẫn còn gặp phải một số vấn đề.

- Đầu tiên là chúng tightly coupled, nghĩa là kiến trúc triển khai cài đặt bên phía nhà cung cấp dịch vụ và phía sử dụng dịch vụ phải giống nhau. Điều này đồng nghĩa với khó khăn mỗi khi có sự thay đổi từ một trong 2 phía bởi vì mỗi thay đổi cần được đánh giá, lên kế hoạch và sửa chữa ở cả 2 phía.
- Tiếp đến những chuẩn trên đa phần là chuẩn đóng, chúng hầu như không thể kết hợp, hoạt động với chuẩn khác. Ví dụ như bất đối tượng Java trao đổi dữ liệu trực tiếp với một đối tượng DCOM là không thể. Cuối cùng các đối tượng của các mô hình trên là fine grained, nghĩa là lượng thông tin giữa trong mỗi lần thực hiện giao dịch là ít, và được thực hiện nhiều lần dẫn đến chiếm dụng băng thông sử dụng và tăng thời lượng đáp trả dữ liệu.

1.3 Các vấn đề phát sinh, nguyên nhân và biện pháp khắc phục

Ngày nay áp lực đặt lên các doanh nghiệp ngày càng lớn: giảm chi phí đầu tư cơ sở hạ tầng, khai thác có hiệu quả các công nghệ có sẵn, phải cố gắng phục vụ yêu cầu của khách hàng ngày càng tốt hơn, đáp ứng tốt các thay đổi nghiệp vụ, khả năng tích

hợp cao với các hệ thống bên ngoài... Nguyên nhân chính của mọi khó khăn trên đó là: sự không đồng nhất và sự thay đổi.

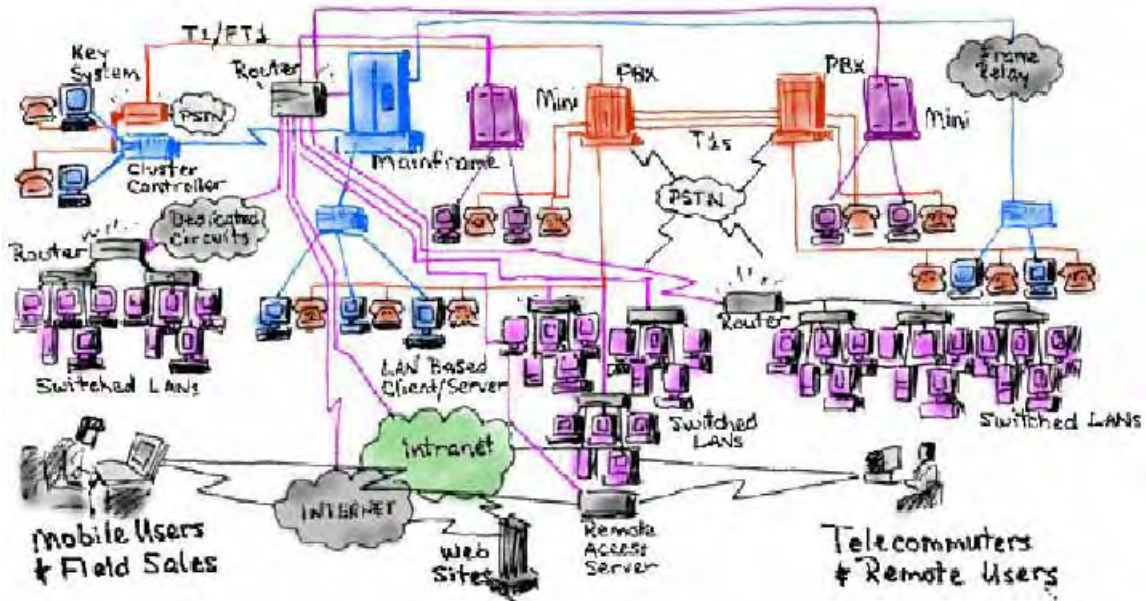
Hầu hết các doanh nghiệp ngày nay đều sở hữu nhiều hệ thống, ứng dụng, với những kiến trúc khác nhau, xây dựng vào những khoảng thời gian khác nhau và dựa trên những công nghệ khác nhau. Vào những năm 1990, các doanh nghiệp chọn giải pháp trọn gói, mua hẳn một vài gói phần mềm lớn với những module được tích hợp sẵn thay vì cố gắng “sửa và kết hợp” chúng với nhau, bởi vì lúc bấy giờ kết hợp các sản phẩm từ nhiều nhà cung cấp khác nhau thực sự là một cơn ác mộng. Ngày nay, các doanh nghiệp không thể chỉ trả như vậy, vì một giải pháp trọn gói thường không linh hoạt và có giá thành cao. Các doanh nghiệp quay lại tìm kiếm những giải pháp kết hợp những ứng dụng cũ sao cho thoả mãn nhu cầu, để những ứng dụng đó giải quyết phần việc của mình, sau đó chỉ việc tổng hợp thông tin trả về. Trong quá trình kết hợp chắc chắn sẽ gặp những khó khăn như:

- Không đủ khả năng quản lý quy trình nghiệp vụ
- Tồn chi phí tích hợp
- Số lượng lớn nhà cung cấp và khách hàng, đó là chưa kể các đối thủ cạnh tranh, các quy trình nghiệp vụ phức tạp
- Số lượng lớn các ứng dụng cần kết hợp và quản lý như Enterprise Resource Planning (ERP), Supply Chain Management (SCM), và Product Data Management(PDM)

- Quá nhiều định dạng dữ liệu
- Vấn đề bảo mật

Trong khi đó những thay đổi vẫn liên tục xảy ra

- Toàn cầu hoá dẫn đến tính cạnh tranh khốc liệt đòi hỏi phải rút ngắn quy trình sản phẩm để tăng ưu thế cạnh tranh với các đối thủ.
- Nhu cầu và yêu cầu khách hàng thường xuyên thay đổi nhanh chóng nhằm cho ra các sản phẩm có tính cạnh tranh liên tục xuất hiện trên thị trường.
- Cải tiến công nghệ dẫn đến thay đổi các thành phần liên quan



Hình 1-5 – Thực trạng cơ sở hạ tầng IT của hầu hết các tổ chức hiện nay.

Đa phần những khó khăn trên là bắt nguồn từ một trong ba nguyên nhân: phức tạp, không linh hoạt và không bền vững.

- **Phức tạp:** Ngày nay mỗi doanh nghiệp công nghệ thông tin có nhiều hệ thống đủ loại khác nhau và làm việc theo những cách khác nhau. Các công ty phát triển phần mềm phải thuê những nhóm nhân viên giàu kinh nghiệm, có khả năng trên nhiều lĩnh vực khác nhau để phát triển, triển khai và quản lý các ứng dụng và hệ thống mà bản thân chúng không thống nhất với nhau. Thêm vào đó là việc nâng cấp rối rắm, tích hợp cùng với nhu cầu về bảo mật ngày một tăng làm gia tăng tính phức tạp cho những vấn đề vốn đã khó giải quyết với các doanh nghiệp.
- **Không linh hoạt:** cùng với sự phức tạp là tính cứng nhắc trong chính sách, chiến lược phát triển, cũng như là cơ sở hạ tầng của các công ty. Hầu như công ty nào cũng có những ứng dụng có sẵn mà khó nâng cấp, khó kết hợp hoạt động hoặc tệ hơn, không thể thay thế. Vấn đề tích hợp vì thế trở nên tốn kém và khó khăn hơn.
- **Không bền vững:** trái ngược với sự cứng nhắc nói trên là sự không bền vững đi cùng với khả năng thất bại và những vấn đề khác đi kèm. Các phương pháp tiếp

cận truyền thống trong việc xây dựng các hệ thống phần mềm thường dẫn đến một “mớ hỗn độn” các giải pháp lắp ghép, tích hợp. Kết quả là mỗi khi có thay đổi về quy trình nghiệp vụ hoặc yêu cầu thì các công ty phải chấp nhận phát triển những dự án nâng cấp tốn kém hoặc là hủy và thay thế hẳn công nghệ không phù hợp. Rủi ro cùng lúc cũng tăng lên với sự phụ thuộc chồng chéo giữa các thành phần , hệ thống có sẵn.

Chính vì vậy các doanh nghiệp cần một cách tiếp cận mới để giải quyết vấn đề môi trường không đồng nhất và tốc độ chóng mặt của sự thay đổi trong khi phải xoay sở với nguồn ngân sách hạn hẹp và nền kinh tế khó khăn. May mắn thay, vẫn có một cách tiếp cận giải quyết khá toàn diện mọi khó khăn nêu trên và nó đã được triển khai trong thực tế. Cách tiếp cận đó gọi là “kiến trúc hướng dịch vụ” Service-oriented Architecture (SOA).

Chương 2

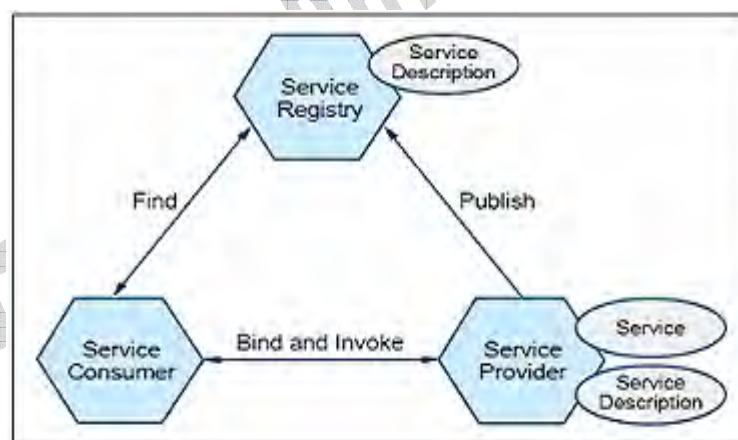
GIỚI THIỆU VỀ KIẾN TRÚC HƯỚNG DỊCH VỤ (SERVICE-ORIENTED ARCHITECTURE)

Nội dung của chương 2 trình bày về cơ sở lý thuyết của mô hình SOA, bao gồm: khái niệm về kiến trúc hướng dịch vụ (SOA), những đặc trưng và ích lợi đạt được của mô hình kiến trúc này. Ngoài ra, chương này cũng sẽ đi sâu vào tìm hiểu các tầng kiến trúc bên trong của mô hình SOA

2.1 Kiến trúc hướng dịch vụ là gì ?

Kiến trúc hướng dịch vụ (Service-oriented architecture) là một hướng tiếp cận với việc thiết kế và tích hợp các phần mềm, chức năng, hệ thống theo dạng module, trong đó mỗi module đóng vai trò là một “dịch vụ có tính loose coupling”, và có khả năng truy cập thông qua môi trường mạng. Hiểu một cách đơn giản thì một hệ thống SOA là một tập hợp các dịch vụ được chuẩn hoá trên mạng trao đổi với nhau trong ngữ cảnh một tiến trình nghiệp vụ.

Trong SOA có ba đối tượng chính, minh họa trong Hình 2-1



Hình 2-1 – Sơ đồ cộng tác trong SOA

Nhà cung cấp (service provider) dịch vụ cần cung cấp thông tin về dịch vụ của mình cho một dịch vụ lưu trữ thông tin dịch vụ (service registry). *Người sử dụng* (service consumer) thông qua *service registry* để tìm kiếm thông tin mô tả về dịch vụ cần tìm và sau đó là xây dựng kênh giao tiếp với phía nhà cung cấp.

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.

Thật ra, tư tưởng về một hệ thống SOA không phải là mới. *Common Object Request Broker Architecture* (CORBA) và mô hình *Distributed Component Object Model* (DCOM) của Microsoft hay như *Enterprise Java Bean* (EJB) của Java của đã cung cấp tính năng này từ lâu. Tuy nhiên những cách tiếp cận hướng dịch vụ này vẫn còn gặp phải một số vấn đề khó khăn (đã phân tích ở trên). SOA không chỉ là một cải tiến đáng kể giúp giải quyết những yếu điểm của các công nghệ trước mà còn đem đến nhiều ưu điểm nổi trội hơn (xem lợi ích của SOA mục 2.4).

2.2 Bốn nguyên tắc chính của hệ thống SOA

2.2.1 Sự phân định ranh giới rạch ròi giữa các dịch vụ

Các dịch vụ thực hiện quá trình tương tác chủ yếu thông qua thành phần giao tiếp. Thành phần giao tiếp này sẽ qui định về những định dạng thông điệp sử dụng 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 thông tin và chức năng của dịch vụ. Ta chỉ cần gửi các thông điệp theo các định dạng đã được định nghĩa trước mà không cần phải quan tâm đến cách xử lý của dịch vụ như thế nào (môi trường thực thi, ngôn ngữ lập trình...). Điều này đạt được do sự tách biệt giữa thành phần giao tiếp và thành phần xử lý trong kiến trúc của dịch vụ .

2.2.2 Các dịch vụ tự hoạt động

Các dịch vụ cần phải được triển khai và hoạt động như những thực thể độc lập mà không lệ thuộc vào một dịch vụ khác. Dịch vụ phải có tính bền vững cao, nghĩa là nó sẽ không bị sụp đổ khi có sự cố. Để thực hiện điều này, 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 một dịch vụ cộng tác bị hỏng; và để tránh các cuộc tấn công từ bên ngoài (như gửi thông điệp lỗi, hay gửi thông điệp ồ ạt) bằng cách sử dụng các kỹ thuật về an toàn, bảo mật...

Đây chính là ý nghĩa của khái niệm ‘loose coupling service’ mà ta đã đề cập trong định nghĩa SOA.

2.2.3 Các dịch vụ chia sẻ lược đồ

Các dịch vụ nên cung cấp thành phần giao tiếp của nó (interface) 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 (độc lập ngôn ngữ, độc lập hệ nền.). Như thế hệ thống của ta sẽ có tính liên kết và khả năng dễ mở rộng.

2.2.4 Tính tương thích của dịch vụ dựa trên chính sách

Điều này nghĩa là, một dịch vụ khi muốn tương tác với một dịch vụ khác thì phải thỏa mãn các chính sách (policy) và yêu cầu (requirements) của dịch vụ đó như là mã hóa, bảo mật... Để thực hiện điều này, mỗi dịch vụ cần phải cung cấp công khai các yêu cầu, chính sách đó.

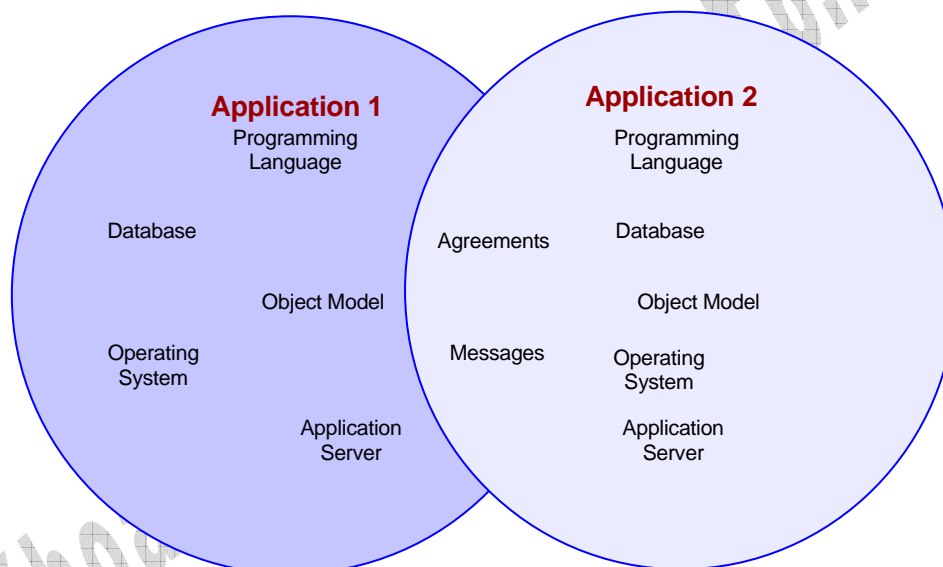
2.3 Các tính chất của một hệ thống SOA

2.3.1 Loose coupling

Vấn đề kết nối (coupling) ám chỉ đến một số ràng buộc giữa các module với nhau. Có hai loại coupling là rời (*loose*) và chặt (*tight*). Các module có tính loose coupling có một số ràng buộc được mô tả rõ ràng trong khi các module có tính tight coupling lại có nhiều ràng buộc không thể biết trước. Hầu như mọi kiến trúc phần mềm đều

hướng đến tính loose coupling giữa các module. Mức độ kết dính của mỗi hệ thống ảnh hưởng trực tiếp đến khả năng chỉnh sửa hệ thống của chính nó. Kết dính càng chặt bao nhiêu thì càng có nhiều thay đổi liên quan cần chỉnh sửa ở phía sử dụng dịch vụ mỗi khi có thay đổi nào đó xảy ra. Mức độ coupling tăng dần khi bên sử dụng dịch vụ càng 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 giữa hai bên càng chặt. 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 hai bên càng có tính loose coupling.

SOA hỗ trợ loose coupling thông qua việc sử dụng hợp đồng và liên kết (contract and binding). Một người sử dụng truy vấn đến nơi lưu trữ và cung cấp thông tin dịch vụ (registry) để lấy thông tin về loại dịch vụ cần sử dụng. Registry sẽ trả về tất cả những dịch vụ thỏa tiêu chuẩn tìm kiếm. Từ bây giờ người dùng chỉ việc chọn dịch vụ mà mình cần, và thực thi phương thức trên đó theo mô tả dịch vụ nhận được từ registry. Bên sử dụng dịch vụ không cần phụ thuộc trực tiếp vào cài đặt của dịch vụ mà chỉ dựa trên hợp đồng mà dịch vụ đó hỗ trợ.



Hình 2-2 - Tính chất loose-coupling

Tính loose coupling 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 hiệu suất, khả năng mở rộng

và khả năng đáp ứng cao. Những thay đổi cài đặt cũng được che dấu đi. Loose coupling đem đến sự độc lập giữa bên cung cấp và bên sử dụng nhưng nó đòi hỏi các interface phải theo chuẩn và cần một thành phần trung gian quản lý, trung chuyển yêu cầu giữa các hệ thống đầu cuối.

2.3.2 Sử dụng lại dịch vụ

Bởi vì các dịch vụ được cung cấp lên trên mạng và được đăng ký ở một nơi nhất định nên chúng dễ dàng được tìm thấy và tái sử dụng. Nếu một dịch vụ không có khả năng tái sử dụng, nó cũng không cần đến interface 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 độ vững chắc trong cài đặt, nó còn giúp đơn giản hoá 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 *shared infrastructure service*.

2.3.3 Sử dụng dịch vụ bất đồng bộ

Trong phương thức triệu gọi dịch vụ bất đồng bộ, bên gọi gửi một thông điệp với đầy đủ thông tin ngữ cảnh tới bên nhận. Bên nhận xử lý thông tin và trả kết quả về thông qua một “kênh thông điệp”, bên gọi không phải chờ cho đến khi thông điệp được xử lý xong. Khi sử dụng kết hợp thông điệp dạng coarse-grained với một dịch vụ chuyên thông điệp, các yêu cầu dịch vụ có thể được đưa vào hàng đợi và xử lý với tốc độ tối ưu. Do bên gọi không phải chờ cho đến khi yêu cầu được xử lý xong và trả về nên không bị ảnh hưởng bởi việc xử lý trễ và lỗi khi thực thi các dịch vụ bất đồng bộ. Trên lý thuyết một hệ thống SOA có thể hỗ trợ gửi và nhận cả thông điệp đồng bộ và bất đồng bộ.

2.3.4 Quản lý các chính sách

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 policy. Các policy cần được quản lý các áp dụng cho mỗi dịch vụ cả khi thiết kế lẫn khi trong thời gian thực thi.

Việc này tăng khả năng tạo ra các dịch vụ có đặc tính tái sử dụng. Bởi vì các policy 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 policy, 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 policy. Ngược lại, nếu sử dụng policy, những nhân viên phát triển phần mềm giờ 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.

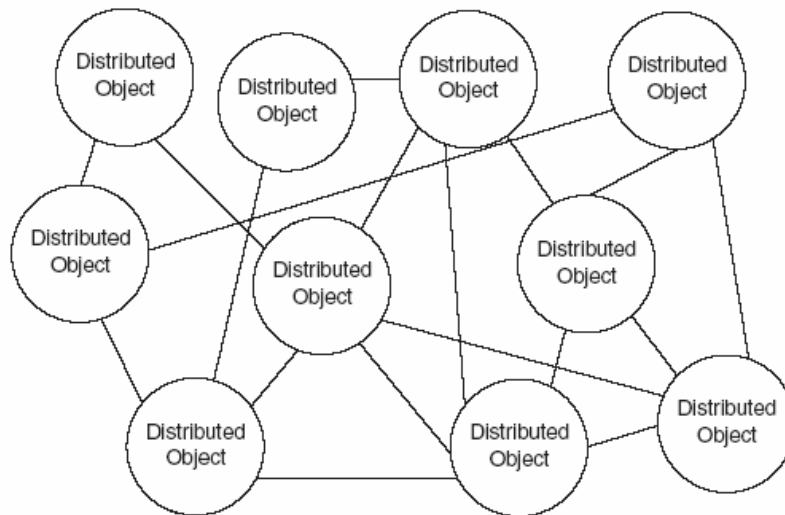
2.3.5 Coarse granularity

Khái niệm granularity trong dịch vụ có thể hiểu theo hai cách. Đầu tiên, nó được hiểu trong phạm vi toàn bộ kiến trúc cài đặt của dịch vụ. Thứ hai, nó được hiểu trong phạm vi từng phương thức của từng interface triển khai. Mức độ granularity cũng được hiểu ở mức tương đối. Ví dụ, nếu một dịch vụ cài đặt tất cả chức năng của một hệ thống ngân hàng, chúng ta xem nó là *coarse-grained*. Nếu nó hỗ trợ chỉ chức năng kiểm tra thể tính dụng, chúng ta lại xem nó là *fine-grained*.

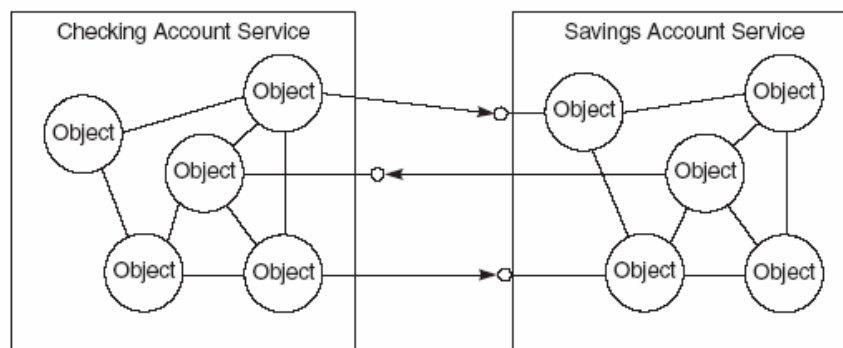
Trước khi có kiến trúc thành tổ và dịch vụ, các hệ thống phân tán chủ yếu dựa trên ý tưởng phân tán đối tượng. Những hệ thống phân tán đối tượng chứa bên trong nó nhiều đối tượng *fine-grained* trao đổi thông tin với nhau qua mạng. Mỗi đối tượng có những ràng buộc với nhiều đối tượng khác bên trong hệ thống. Do truy cập đến một đối tượng phải qua nhiều trung gian mà hiệu quả đạt được không cao nên khuyến khích thiết kế hệ thống phân tán đối tượng đang dần chuyển sang thiết kế các *coarser-grained interface*.

Hình 2-3 minh họa một hệ thống phân tán đối tượng với nhiều mối liên kết. Cùng với kích thước và độ phức tạp của hệ thống ngày càng tăng, những ràng buộc này trở nên ngày càng khó quản lý. Hiệu suất cũng giảm tương ứng số lượng các kết nối trung gian. Khả năng bảo trì cũng giảm khi số lượng ràng buộc giữa những đối tượng ngày một tăng. Khi một đối tượng cần được thay đổi ở interface, nó có thể ảnh hưởng đến một lượng lớn những đối tượng phân tán khác. Nhân viên phát triển phải biên dịch và triển khai lại toàn bộ đối tượng bị thay đổi và những đối tượng liên quan với chúng.

Một hệ thống dựa trên quản lý các truy cập đến đối tượng bên trong dịch vụ thông qua một số coarse-grained interface như Hình 2-4. Một dịch vụ có thể được cài đặt như một tập những đối tượng fine-grained nhưng bản thân những đối tượng đó lại được sử dụng trực tiếp qua mạng. Trong khi đó một service được cài đặt như những đối tượng có một hoặc nhiều đối tượng coarse-grained hoạt động như những facades phân tán thì những đối tượng này lại có thể được sử dụng qua mạng và cho phép truy cập đến các đối tượng sâu bên trong. Tuy nhiên các đối tượng bên trong service đó bây giờ sẽ trao đổi trực tiếp với nhau ở trong cùng một máy chứ không phải trên mạng.

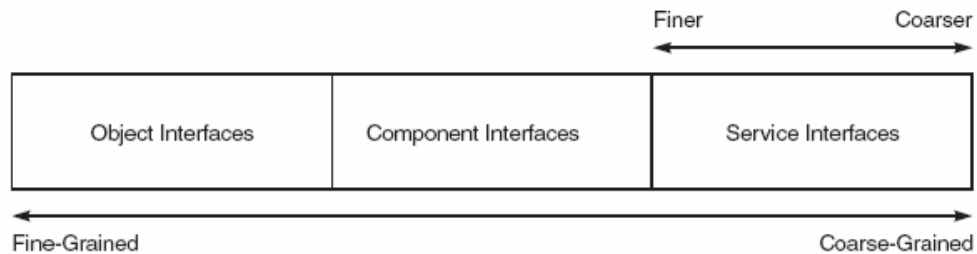


Hình 2-3 – Các đối tượng fine-grained



Hình 2-4 – Các đối tượng coarse-grained

Mặc dù những service nói chung hỗ trợ coarser-grained interface hơn các hệ thống phân tán đối tượng và các hệ thống hướng thành tố, độ “thô” vẫn hàm chứa bên trong nó chứa một mức độ granularity nào đó, như hình Hình 2-5.



Hình 2-5 – Các mức độ granularity

2.3.6 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 đổi đối tượng dạng Java thành SOAP.

2.3.7 Tự động dò tìm và ràng buộc động

SOA hỗ trợ khái niệm *truy tìm 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 dịch vụ nào thoả yêu cầu tìm kiếm. Ví dụ, một hệ thống chuyển khoảng (consumer) yêu cầu một registry tìm tất cả các dịch vụ có khả năng kiểm tra thẻ tín dụng. Registry trả về một tập các entry thoả yêu cầu. Các

entry chứa thông tin về dịch vụ, bao gồm cả 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 đó dựa trên thông tin registry entry 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 cung cấp 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. Mỗi ràng buộc duy nhất giữa bên cung cấp và bên sử dụng là bản hợp đồng được cung cấp bởi registry trung gian. Mỗi ràng buộc này là ràng buộc trong thời gian chạy chứ không phải ràng buộc trong lúc biên dịch. Tất cả thông tin cần thiết về dịch vụ được lấy về và sử dụng trong khi chạy.

Ví dụ trên cho thấy cách bên sử dụng triệu gọi dịch vụ một cách “động”. Đây là một thế mạnh của SOA. 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.

2.3.8 Tự hồi phục

Với kích cỡ và độ phức tạp của những ứng dụng phân tán ngày nay, khả năng phục hồi của một hệ thống sau khi bị lỗi trở thành một yếu tố quan trọng. Một hệ thống tự hồi phục (self-healing) là một hệ thống có khả năng tự hồi phục sau khi bị lỗi mà không cần sự can thiệp của con người.

Độ tin cậy (reliability) là mức độ đo khả năng một hệ thống xử lý tốt như thế nào trong tình trạng hỗn loạn. Trong kiến trúc hướng dịch vụ, các dịch vụ luôn có thể hoạt động hay ngừng bất kỳ lúc nào, nhất là đối với những ứng dụng tổng hợp từ những 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 ứng dụng được xây dựng. Một kiến trúc hỗ trợ kết nối và thực thi động khi chạy sẽ có khả năng tự phục hồi hơn một hệ thống không hỗ trợ những tính năng trên.

Thêm vào đó, bởi vì những hệ thống dựa trên dịch vụ yêu cầu sự tách biệt giữa interface và cài đặt, nên có thể có nhiều cài đặt khác nhau cho cùng một interface. 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 chỉ tương tác với interface của dịch vụ chứ không tương tác trực tiếp cài đặt của dịch vụ. Đây là một trong những tính chất cơ bản của các hệ thống hướng dịch vụ.

2.4 Lợi ích của SOA

Nói đến SOA là nói đến ‘tiết kiệm’ - cả tiết kiệm chi phí lẫn thu được giá trị nhiều hơn từ các hệ thống có sẵn. Hẳn cũng phải có lý do để hàng trăm tập đoàn đã chú ý đến SOA như một giải pháp tích hợp nhằm giảm giá thành của một đề án một cách rất ấn tượng.

➤ Sử dụng lại những thành phần có sẵn

Một trong những lợi ích rõ ràng nhất của SOA là nó giúp các công ty thu được giá trị nhiều hơn bằng cách sử dụng lại những tài nguyên sẵn có; kết quả là giảm chi phí cho phần kiến trúc và tích hợp. Ngoài ra nó còn giúp giảm chi phí mua phần mềm mới. Thời gian viết chương trình lấy dữ liệu từ máy chủ trước đây được tính bằng tháng thì bây giờ chỉ còn tính bằng phút ! Lợi ích của việc sử dụng lại có thể chia làm 2 phần :

- Lợi ích từ việc sử dụng lại những thành phần nhằm giảm tính dư thừa.
- Lợi ích từ việc sử dụng lại những thành phần có sẵn khi thiết kế cung cấp một chức năng mới.

Đầu tiên như đã nói, nhiều phần mềm doanh nghiệp đã phát triển thành những nhóm phần mềm tách biệt (functional silos), thông thường là tương ứng với mỗi đơn vị kinh doanh. Ví dụ một công ty bán lẻ có thể có một nhóm phần mềm cho hệ thống phân phối, một nhóm phần mềm cho hệ thống lưu kho và một nhóm phần mềm cho những chức năng liên kết. Thông thường những nhóm phần mềm này được phát triển trên nhiều nền tảng khác nhau, sử dụng nhiều ngôn ngữ lập trình khác nhau và

thường có nhiều tính năng lặp lại giữa chúng. Một hệ thống SOA cho phép các công ty tránh tình trạng lặp dư thừa, tạo ra những đơn thể dịch vụ chia sẻ giữa các ứng dụng.

Trong một hệ thống SOA, chỉ cần thay đổi duy nhất một phiên bản của dịch vụ cần được thay đổi và chỉ cần kiểm thử một lần, sử dụng những kỹ năng tương ứng với những kỹ năng đã dùng để phát triển dịch vụ. Lợi ích rõ ràng nhất là giảm chi phí bảo trì phần mềm. Ngoài ra điều này còn giúp doanh nghiệp chịu trách nhiệm nhiều hơn với những thay đổi về mặt nghiệp vụ và cho phép doanh nghiệp cập nhật những tính năng của nó nhanh hơn.

Bằng cách phân rã một ứng dụng thành những đơn thể dịch vụ nghiệp vụ, sau đó cho những dịch vụ này liên kết lại với nhau, các hệ thống bây giờ có thể sử dụng lại được các thành phần có sẵn, giảm chi phí phát triển từng phần độc lập cho mỗi tính năng mới chưa có. Thay vì phải “thay đổi”, với SOA ta chỉ cần tạo ra các “cầu nối” liên hệ giữa những hệ thống và ứng dụng khác nhau, thay vì chỉnh sửa hoặc xây dựng lại từ đầu.

Bởi vì có đa phần các dịch vụ mới sử dụng lại những dịch vụ sẵn có nên chi phí phát triển các thành phần mới được giảm đến mức tối thiểu. Nghĩa là :

- Các công ty có thể triển khai những tiến trình xử lý mới nhanh hơn rất nhiều.
- Chi phí dành cho phát triển và kiểm thử giảm đáng kể
- Giảm rủi ro khi dịch vụ tạm ngưng hoạt động

Lợi ích cuối cùng của việc tái sử dụng thường khó nhận thấy, đó là sử dụng những thành phần có sẵn trước đó mỗi khi có thể thì mỗi khi có lỗi xảy ra ta có thể giới hạn vào khu vực có những thành phần đang được phát triển. Nhờ đó rủi ro về lỗi phần mềm giảm đi và tăng chất lượng dịch vụ.

Giảm chi phí phát triển và kiểm thử, và tránh công việc trùng lặp là một trong những lợi ích mà SOA mang lại. Nhưng quan trọng vẫn là lợi ích từ việc tăng khả năng kinh doanh. Nếu những nhân tố này được đánh giá đúng mức thì lợi ích mang lại là rất đáng kể.

➤ **Giải pháp ứng dụng tổng hợp cho doanh nghiệp**

Cũng với ví dụ trên, SOA mang đến khả năng tổng hợp một lớp các ứng dụng mới bằng cách kết hợp chức năng từ những hệ thống có sẵn, cung cấp cho người cuối những chức năng liên kết. Ở đây một số tiến trình cũ có thể được kết hợp với nhau bên trong một cổng thông tin (portal) giúp cho người dùng cuối chỉ cần truy cập một lần mà vẫn có thông tin về hàng loạt sản phẩm của doanh nghiệp. Loại kết hợp này có thể khó khăn nếu không sử dụng SOA vì nó đòi hỏi việc tích hợp phức tạp, nỗ lực lập trình và kiểm thử. Nhưng với SOA, một ứng dụng tổng hợp có thể được tổng hợp dễ dàng, bất kể sự khác nhau về địa lý hoặc công nghệ phát triển các dịch vụ đó. Điều này cho phép doanh nghiệp phản ứng nhanh theo yêu cầu, giảm chi phí đến mức tối thiểu và tăng sức mạnh thoả mãn yêu cầu của người dùng cuối hiệu quả hơn.

➤ **Tính loose coupling giúp tăng tính linh hoạt và khả năng triển khai cài đặt**

Lợi ích kế tiếp đến từ tính loose coupling của SOA, trong đó phía triệu gọi dịch vụ không cần quan tâm đến địa chỉ hoặc công nghệ nền tảng của service. Nó mang đến khả năng linh hoạt cao và nhiều lợi ích khác.

Trong một hệ thống SOA ta triệu gọi dịch vụ thông qua các interface theo một dạng thức chuẩn nên giúp lập trình viên tránh được việc phải lặp lại công việc tạo mới các service có khả năng hiểu tất cả những công nghệ được sử dụng bởi từng dịch vụ trong hệ thống.

Thứ hai, trong trường hợp cần kết nối với các đối tác thương mại thì những dịch vụ có tính loose-coupling, những interface chuẩn càng đem lại nhiều lợi ích hơn. Với một hệ thống SOA, thật dễ dàng khi cung cấp một loạt những dịch vụ ra bên ngoài cho một đối tác nào đó sử dụng. Nhờ tính độc lập địa chỉ và công nghệ của SOA, đối tác kia không cần quan tâm đến dịch vụ được cài đặt như thế nào, và nhờ các dịch vụ đã theo chuẩn giao tiếp nên đối tác đó chỉ cần một lượng thông tin nhỏ vừa đủ để sử dụng dịch vụ. Tương tự cho điều ngược lại, nếu đối tác đã xây dựng một hệ thống SOA thì việc đem sử dụng chức năng một số dịch vụ của họ vào sử dụng bên trong hệ thống của mình cũng trở nên dễ dàng và nhanh chóng. Đặc tính này của SOA hứa hẹn tăng hiệu suất và tự động hoá.

Cuối cùng một lợi ích mà tính loose coupling mang lại là tăng khả năng triển khai. Như đã phân tích ở trên, những thành phần có tính loose coupling có thể được triệu gọi mà không cần biết chúng được cài đặt như thế nào mà chỉ cần biết cách thức triệu gọi chúng thông qua một interface chuẩn. Vì vậy chỉ cần bọc những thành phần sử dụng interface ứng dụng thành dạng dịch vụ, ta đã có một đơn thể thành phần được sử dụng trong hệ thống SOA như những dịch vụ bình thường khác.

➤ **Thích ứng với những thay đổi trong tương lai**

Các phương pháp tiếp cận truyền thống trong quy trình phát triển phần mềm có thể mô tả ngắn gọn là người dùng mô tả họ cần gì – công ty phát triển phần mềm – triển khai hệ thống theo yêu cầu. Quy trình này đôi khi gặp khó khăn khi gặp những tình huống thay đổi không định trước. Với SOA, công ty phát triển phần mềm có thể tạo nên những quy trình nghiệp vụ uyển chuyển, phức tạp biến đổi tùy “*theo yêu cầu*” và theo “*thời gian thực*”.

➤ **Hỗ trợ đa thiết bị và đa nền tảng.**

SOA cung cấp một tầng giao tiếp trừu tượng từ các nền tảng bên dưới. Điều này cho phép hỗ trợ nhiều loại thiết bị đầu cuối khác nhau bao gồm cả những trình duyệt và thiết bị di động như pager, điện thoại di động, PDA và các thiết bị chuyên dụng khác sử dụng cùng một chức năng mà vẫn có thông tin trả về tùy theo dạng thiết bị. Tính độc lập công nghệ này giúp cho các công ty tiết kiệm chi phí rất nhiều nhất là khi phải xử lý với vô số công nghệ hiện đang được sử dụng.

➤ **Tăng khả năng mở rộng và khả năng sẵn sàng cung cấp.**

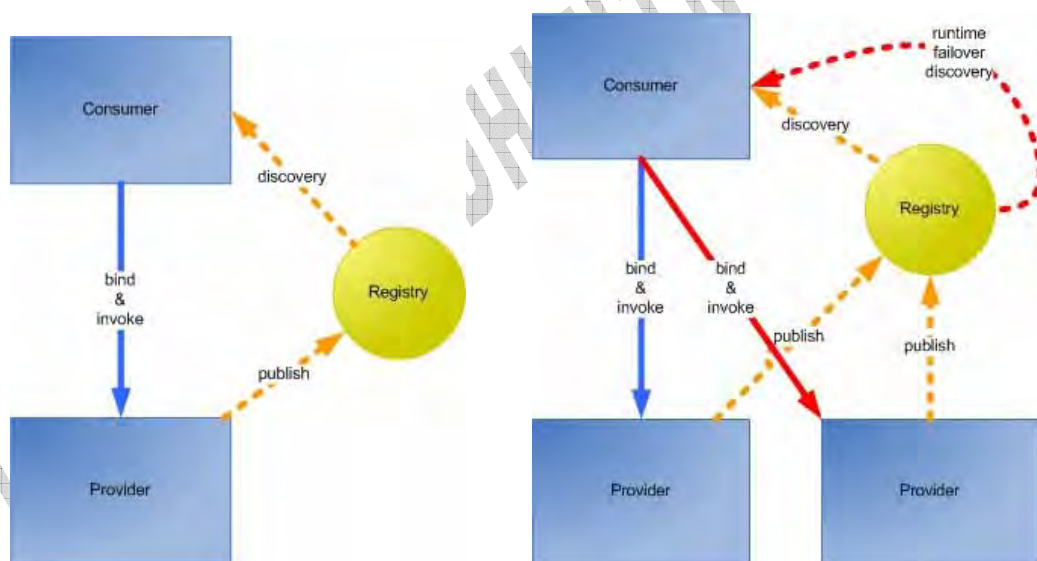
Nhờ tính độc lập địa chỉ của SOA, ta có thể tăng khả năng mở rộng bằng cách thêm nhiều thể hiện (instance) của một service. Công nghệ chia tải (load-balancing) sẽ tự động tìm và định tuyến yêu cầu đến thể hiện service thích hợp. Tương tự, SOA có thể chuyển tiếp nội dung yêu cầu đến một thể hiện khác khi cần, nhờ đó tăng khả năng sẵn sàng phục vụ.

Cần nhấn mạnh là lợi ích mà SOA mang lại không phải là ít mà là rất ấn tượng. Thực tế giá trị kinh tế mà SOA mang lại lớn đến nỗi các tập đoàn trên thế giới đang suy xét xem làm thế nào để chuyển toàn bộ các kiến trúc phần mềm có sẵn của họ thành SOA.

2.5 Một số mô hình triển khai SOA

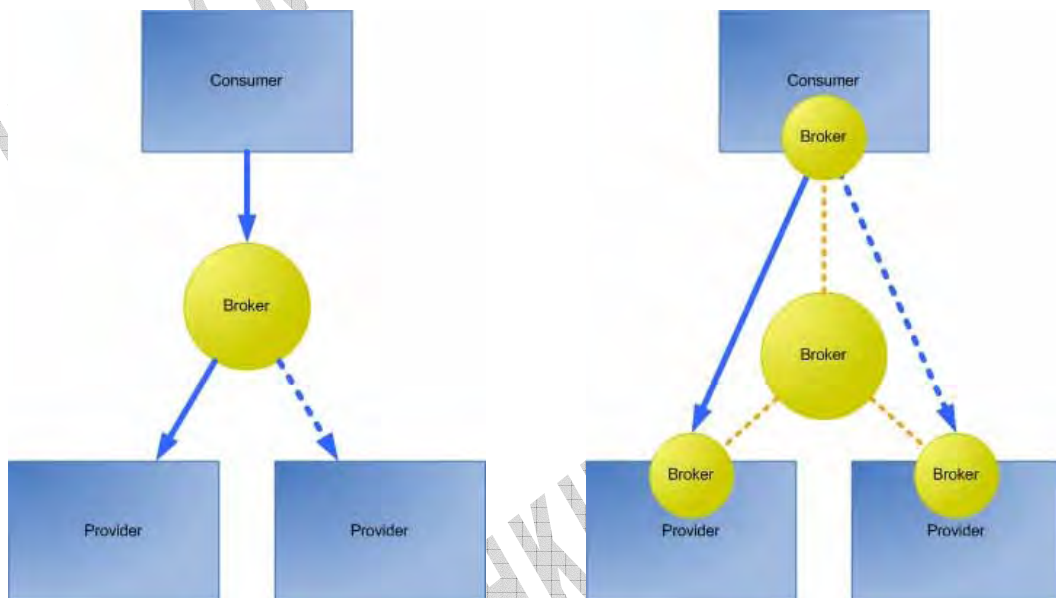
Chúng ta sẽ thảo luận về ba mô hình triển khai chính của SOA là : service registry, service broker và service bus.

Service registry : đây là mô hình truyền thống để định vị và liên kết các dịch vụ trong một hệ thống SOA. (xem hình Hình 2-6) . Mô hình service registry về cơ bản chỉ cần các chuẩn Web services thông thường là SOAP, WSD và UDDI. Vấn đề lớn nhất của mô hình này là các liên kết dịch vụ là kết nối tĩnh và phải định nghĩa trong thiết kế, điều này làm cho mô hình trở nên cứng nhắc. Có một cách cải tiến làm cho mô hình này linh hoạt hơn là tìm kiếm, định vị các dịch vụ khi chạy. UDDI hỗ trợ nhiều cấu hình khác nhau cho cùng một dịch vụ cung cấp bởi nhiều nhà cung cấp dịch vụ khác nhau. Điều này cho phép chia tải và tăng tính tin cậy bởi vì service directory có thể tìm kiếm một dịch vụ nào đó trên tất cả các nhà cung cấp dịch vụ hiện có .



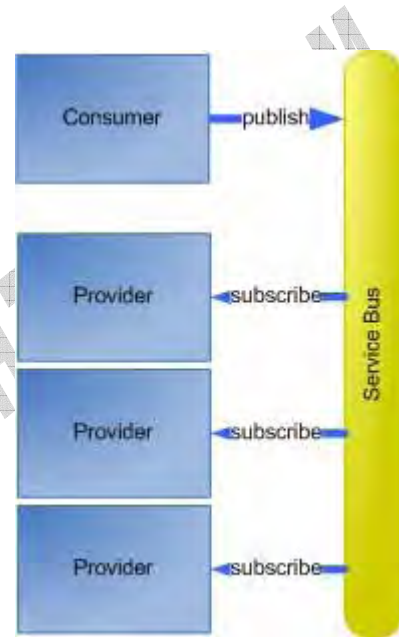
Hình 2-6 - Mô hình service registry

Service broker : Một bộ trung gian làm việc giữa dịch vụ cung cấp và dịch vụ tiêu thụ. Trong mô hình cơ bản, tất cả những thông điệp đều được trung chuyển qua service broker. Dịch vụ này có thể làm nhiều chức năng như định tuyến dựa trên dữ liệu thông điệp, xử lý lỗi, chuyển đổi thông điệp, chia tải và lọc thông tin. Nó cũng có thể cung cấp dịch vụ bảo mật, chuyển đổi giao thức, lưu vết và các dịch vụ hữu ích khác. Tuy nhiên, service broker là nơi có thể xảy ra hiện tượng nghẽn cổ chai và là điểm dễ bị hỏng hóc. Mô hình broker phân tán là một bước cải tiến mới, ở đó mỗi nền tảng dịch vụ có một broker cục bộ cho phép giao tiếp với một service broker trung tâm và giao tiếp trực tiếp với các service broker cùng cấp ở các nền tảng dịch vụ khác.

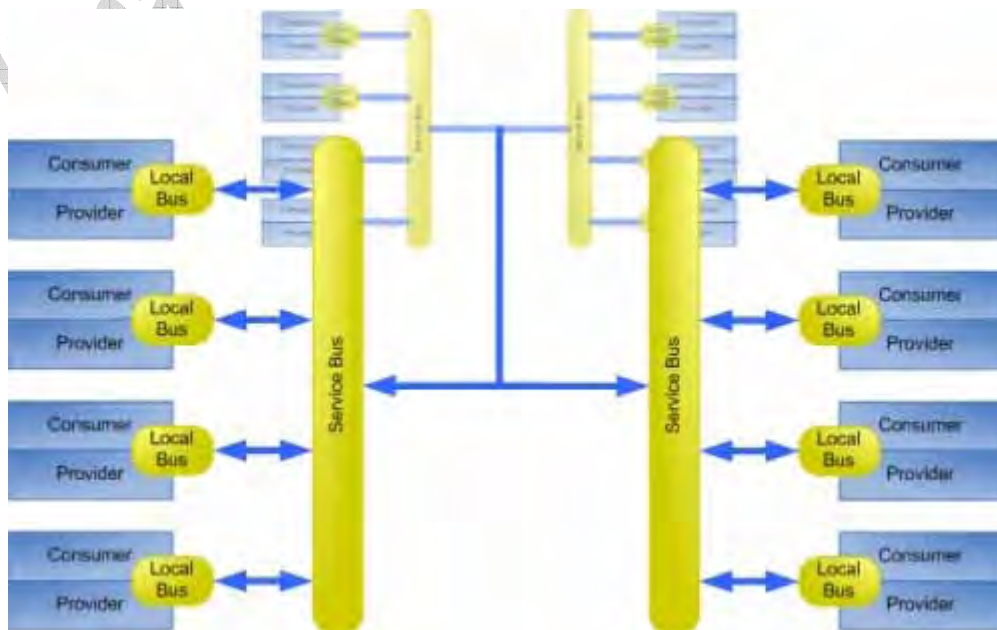


Hình 2-7 – Mô hình service broker

Service bus : đây là mô hình ra đời sau nhất trong 3 mô hình nhưng nó đã được sử dụng trong các sản phẩm thương mại large-scale (như IBM, BEA). Service bus cũng là mô hình có tính loose coupling nhất trong các mô hình, trong đó các dịch vụ không kết nối trực tiếp với nhau. Đôi khi các service bus kết nối với nhau thành một mạng các service bus.



Hình 2-8 – Mô hình service bus



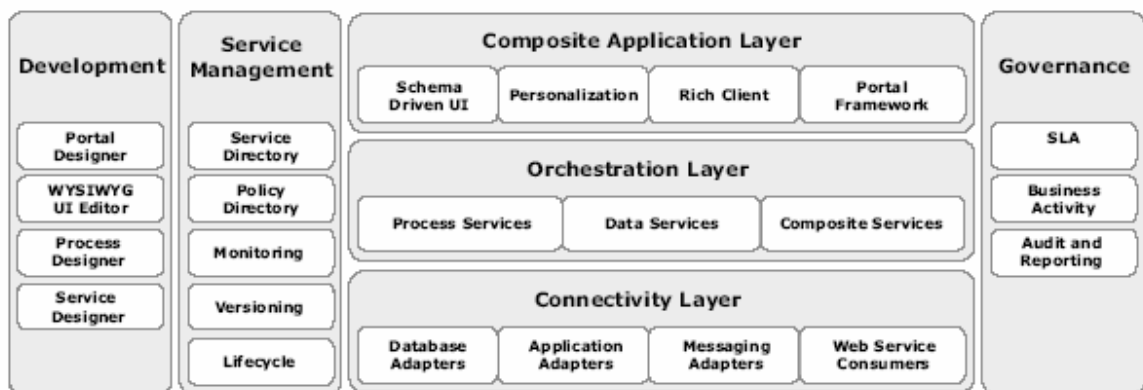
Hình 2-9 – Mô hình service bus phân tán

2.6 Kiến trúc phân tầng chi tiết của SOA

Ở tầng thấp nhất, tầng kết nối (*connectivity*), những dịch vụ được mô hình hoá dựa trên những ứng dụng enterprise bên dưới. Tầng này chứa các dịch vụ như “lấy thông tin chi tiết sản phẩm” hoặc “cập nhật thông tin khách hàng”, chúng tương tác trực tiếp với các hệ thống phi dịch vụ bên dưới. Các dịch vụ này là đặc trưng cho mỗi ứng dụng enterprise.

Phía bên trên tầng kết nối là một số dịch vụ orchestration được thêm vào để tạo ra các dịch vụ thật sự xử lý những chức năng nghiệp vụ độc lập dựa trên những ứng dụng enterprise bên dưới. Những dịch vụ này còn gọi là những dịch vụ tổng hợp (*composite service*).

Trên cùng của tầng service orchestration là các ứng dụng tổng hợp sử dụng các service and cung cấp giao diện cụ thể cho người sử dụng.



Hình 2-10 – Kiến trúc phân tầng của hệ thống SOA

2.6.1 Tầng kết nối

Mục đích của tầng kết nối là kết nối đến các ứng dụng enterprise hoặc tài nguyên bên dưới và cung cấp chúng thành dạng những dịch vụ. Tầng này là tầng chuyên giao tiếp với các nhà cung cấp, hoạt động như một bộ chuyển đổi (*adapter*) giữa các ứng dụng phi dịch vụ và mạng các dịch vụ khác. Tùy vào ứng dụng cụ thể nào mà bộ chuyển đổi tương ứng được sử dụng.

Về cơ bản, tầng này có thể dùng để thực hiện kết nối đến các hệ cơ sở dữ liệu. Những ngôn ngữ lập trình hiện đại cung cấp những tập hàm API cho phép truy cập đến hầu hết cơ sở dữ liệu thông dụng. Với các ứng dụng enterprise thì lại khác, vì mỗi nhà cung cấp cung cấp tập hàm API khác nhau.

2.6.2 Tầng orchestration

Tầng orchestration chứa các thành phần đóng vai trò vừa là những dịch vụ sử dụng vừa là những dịch vụ cung cấp. Những dịch vụ này sử dụng những dịch vụ của tầng kết nối và các dịch vụ orchestration khác để kết hợp những chức năng cấp thấp hơn thành những dịch vụ hoạt động ở cấp cao hơn, có hành vi gần với những chức năng nghiệp vụ thực hơn.

Simple composite service : là những dịch vụ đơn thuần kết hợp những lời triệu gọi đến các dịch vụ bên dưới, hoạt động như mẫu mặt tiền. Chúng giúp đơn giản hoá qua trình tương tác với các dịch vụ cấp thấp và che dấu tính phức tạp tới người sử dụng các dịch vụ ở tầng cao.

Process service: là những dịch vụ định ra một tiến trình kết nối những dịch vụ cấp thấp hơn. Điều này rất hữu dụng khi thiết kế các dịch vụ kết nối đến nhiều hệ thống enterprise bên dưới sau đó thực thi như một tiến trình. Ví dụ dịch vụ “Submit New Order” có thể được xem như một process service thực hiện tương tác với cơ sở dữ liệu khách hàng để lấy thông tin chi tiết về khách hàng, quyết định dựa trên thông tin tài khoản của khách hàng, tương tác hệ thống lưu kho và hệ thống tài chính để hoàn tất yêu cầu đặt hàng.

Bởi vì process service có những đặc tính gần với quy trình nghiệp vụ của doanh nghiệp nên hiện có rất nhiều nỗ lực để chuẩn hoá cách thức định nghĩa ra chúng. WS-BPEL (Web Service Business Process Execution Language) là tên được tổ chức OASIS chọn cho chuẩn này. Cho đến tháng 5 năm 2004, chuẩn này vẫn được liên tục cập nhật để có thể mô tả nhiều process đa dạng với mức độ phức tạp ngày càng cao.

Data service : là những dịch vụ cung cấp dữ liệu thu thập từ nhiều nguồn dữ liệu tách biệt khác nhau. Trong nhiều trường hợp dữ liệu cùng tồn tại trên nhiều ứng dụng và cơ sở dữ liệu khác nhau. Ví dụ thường thấy là thông tin về khách hàng. Doanh nghiệp thường lưu trữ thông tin khách hàng trong hệ thống đặt hàng, hệ thống tài chính và hệ thống CRM. Mỗi hệ thống chỉ chứa một phần dữ liệu trong toàn bộ dữ liệu về khách hàng.

Data service thường không được xem như một dịch vụ orchestration mà nó chịu trách nhiệm tương tác trực tiếp với những cơ sở dữ liệu bên dưới thông qua những phương thức truy cập phi dịch vụ (non-service oriented accesss) như JDBC hoặc J2CA. Chúng cung cấp dữ liệu từ những ứng dụng độc lập và kết hợp dữ liệu từ nhiều nguồn khác nhau.

Data service thường sử dụng một số ngôn ngữ truy vấn và cơ chế mô tả khác để xác định quan hệ giữa những lược đồ dữ liệu (data schema). Các công nghệ phổ biến hiện nay là SQL, XSLT và Xquery trong đó XQuery và XSLT là hai công nghệ thuần về việc truy vấn dữ liệu trong bối cảnh Web Service vì chúng xử lý và kết xuất dữ liệu dạng XML.

2.6.3 Tầng ứng dụng tổng hợp

Dữ liệu truyền qua lại giữa những dịch vụ cuối cùng cũng định hướng đến người sử dụng theo nhiều dạng giao diện khác nhau. Tầng này được xem là tầng tích hợp cuối cùng của quá trình tích hợp.



Hình 2-11 – Một công thông tin cung cấp thông tin trong một vùng nhìn duy nhất



Hình 2-12 – Các portlet truy xuất dữ liệu từ nhiều nguồn khác nhau được cung cấp dưới dạng dịch vụ

Tầng ứng dụng tổng hợp là tầng đơn thuần sử dụng các dịch vụ, nó cung cấp các ứng dụng cho người dùng cuối. Nhờ tính linh hoạt của SOA và đặc tính của các dịch vụ được tổng hợp từ tầng orchestration, các ứng dụng tổng hợp có khả năng biểu diễn mọi loại thông tin từ mọi nguồn thông tin, thậm chí còn cho phép người sử dụng gửi thông tin tổng hợp mà thông tin đó sẽ được phân phối lại cho các hệ thống bên dưới.

Bản chất của giao diện là khó xây dựng. Chúng cũng khó được chia thành từng thành phần logic tương tác với nhau theo dạng chuẩn hoá. Đôi khi cố gắng phân rã một thành phần giao diện thành những phần nhỏ lại làm mất bố cục chặt chẽ và khó sử dụng hơn là ứng dụng gốc.

Tầng ứng dụng tổng hợp chia làm hai tầng nhỏ hơn là Portal và tầng Portlet. Một Portlet được định nghĩa như một ứng dụng chạy trong một cửa sổ thành phần trong một ngữ cảnh lớn hơn với sự tách biệt rõ ràng giữa Portlet và ngữ cảnh của nó. Portlet là thành phần cung cấp và sử dụng dịch vụ. Có điều là dịch vụ chúng cung cấp là một dạng dịch vụ giao diện đặc biệt được thiết kế đặc biệt để được sử dụng bởi một bộ *UI Framework consumer* (một Portal). Mỗi portlet sử dụng một số dịch vụ liên quan của tầng orchestration bên dưới và cho phép người sử dụng gửi thông tin bổ sung. Công nghệ web hiện nay như Java Server Faces (JSF) và ASP.NET đều hỗ trợ xây dựng portlet. Portal là một bộ khung tích hợp sử dụng các Portlet, trang bị cho chúng một vẻ ngoài thống nhất và thể hiện thành một giao diện hoàn chỉnh cho người dùng cuối.

✍ *Kiến trúc hướng dịch vụ thật sự đem đến những lợi ích to lớn. Thế nhưng để đạt được những lợi ích này ta phải xây dựng và triển khai thành công hệ thống SOA. Chương 3 sẽ trình bày về các vấn đề này.*

Chương 3

XÂY DỰNG HỆ THỐNG SOA

Chương 3 sẽ trình bày các vấn đề liên quan đến việc xây dựng và triển khai hệ thống SOA trong thực tế một cách hiệu quả, bao gồm phân tích các thách thức gặp phải và xem xét qui trình các bước nên thực hiện khi xây dựng hệ thống SOA.

3.1 Những thách thức khi xây dựng hệ thống SOA

Những lợi ích đạt được của một hệ thống SOA đã quá rõ ràng. Thế nhưng việc triển khai một hệ thống SOA không phải là điều dễ dàng. Trong thời gian qua, chúng ta đã chứng kiến các sự thay đổi các mô hình tính toán trong các tổ chức. Từ mô hình tính toán tập trung (mainframe) sang các mô hình phân tán client/server, rồi sau đó là các kiến trúc dựa trên nền tảng Web bởi phát triển lớn mạnh của Internet. Và nay, quá trình này vẫn còn tiếp tục. Chúng ta đang ở thời kỳ quá độ sang mô hình tính toán dựa trên dịch vụ và kiến trúc hướng dịch vụ (SOA). Với mọi sự chuyển đổi đều có những thảo luận, tranh cãi, kinh nghiệm...; có những người “dẫn đường” và những người “nổi bước”; có những người “chiến thắng” và những người “thất bại”; có những hệ thống vận hành hiệu quả và những hệ thống bị đổ vỡ... Lần này cũng sẽ không có gì khác.

Ta sẽ xem xét các vấn đề mà sẽ gây nhiều trở ngại cho trong quá trình triển khai các hệ thống SOA. Phần này sẽ trình bày thách thức mà một tổ chức sẽ phải đối mặt tương ứng với các pha trong một dự án triển khai thực tế.

- Xác định dịch vụ
 - ▶ Dịch vụ là gì? Chức năng nghiệp vụ nào cần được cung cấp bởi một dịch vụ? Độ “mịn” (granularity) của dịch vụ thế nào là tốt?
 - ▶ Việc xác định dịch vụ và quyết định đối tượng cung cấp dịch vụ một cách thích hợp, hiệu quả là giai đoạn quan trọng và đầu tiên trong một giải pháp

hướng dịch vụ. Trong thực tế, nhiều chức năng nghiệp vụ tương tự nhau có thể được cung cấp bởi nhiều hệ thống trong một tổ chức.

- Phân bổ dịch vụ

- ▶ Ta nên đặt dịch vụ ở vị trí nào trong hệ thống?
- ▶ Các dịch vụ thường hoạt động dựa trên các thực thể nghiệp vụ. Các đối tượng này được lưu và quản lý trong hệ thống. Vị trí của các thực thể này cũng là vị trí tốt nhất để đặt dịch vụ. Tuy nhiên, bởi đặc tính của một hệ thống phân tán nên các đối tượng này được phân bố rải rác ở nhiều vị trí và có thể có tình trạng một đối tượng được quản lý ở nhiều nơi. Vì thế, đồng bộ dữ liệu giữa các hệ thống trở nên là một yêu cầu quan trọng. Trong môi trường như thế thì dịch vụ sẽ được đặt ở đâu?

- Xác định miền dịch vụ

- ▶ Làm sao gom nhóm các dịch vụ thành các miền luận lý (logic domain)?
- ▶ Việc phân loại, gom nhóm các dịch vụ thành các miền luận lý sẽ đơn giản hóa kiến trúc hệ thống bởi sẽ giảm được số lượng các thành phần cần xây dựng. Ngoài ra, việc định nghĩa các miền luận lý như thế cũng ảnh hưởng đến nhiều khía cạnh khác của kiến trúc hệ thống như cân bằng tải (load balancing), điều khiển truy cập (access control), sự phân chia theo chiều sâu hay chiều rộng các xử lý nghiệp vụ. Tuy nhiên, điều khó khăn nhất đó là làm sao đạt được đồng nhất quan điểm của các đơn vị, các bộ phận kỹ thuật về tính hợp lý của một miền tạo ra.

- Đóng gói dịch vụ

- ▶ Làm sao có thể bao bọc các chức năng sẵn có của hệ thống cũ vào trong một dịch vụ?
- ▶ Nếu hệ thống khi được thiết kế đã quan tâm và hỗ trợ vấn đề tích hợp với các hệ thống mới thì vấn đề này sẽ dễ dàng hơn. Tuy nhiên, khi các hệ thống cũ này trước đây được xây dựng như the mô hình kín, đóng gói (monolithic) trong đó chứa toàn bộ các thông tin về nguyên tắc và qui trình

xử lý thì nay, khi được tích hợp, các thông tin này cần được chia sẻ và phân bố trong nhiều ứng dụng khác nhau.

- Service orchestration:

- ▶ Làm sao để có thể tạo ra các dịch vụ tổng hợp (composite service)?
- ▶ Nhu cầu kết hợp nhiều dịch vụ để đáp ứng được nhiều yêu cầu phức tạp từ đối tượng sử dụng là có thực. Vấn đề là làm sao kết hợp các dịch vụ này một cách có hiệu quả, theo những qui trình với những ràng buộc phức tạp.

- Định tuyến dịch vụ

- ▶ Làm sao để chuyển một yêu cầu từ một đối tượng sử dụng dịch vụ đến dịch vụ hay miền dịch vụ thích hợp?
- ▶ Một hệ thống SOA phải có tính độc lập địa chỉ cho đối tượng sử dụng các service của hệ thống. Ngoài ra còn phải quan tâm đến vấn đề hiệu suất hoạt động của hệ thống vì việc định vị một dịch vụ là một quá trình cũng chiếm nhiều thời gian.

- Quản lý dịch vụ

- ▶ Vấn đề quản lý và bảo trì các dịch vụ, việc tạo xây, theo dõi và thay đổi các dịch vụ như thế nào cho có hiệu quả?

- Chuyển đổi thông điệp chuẩn dịch vụ

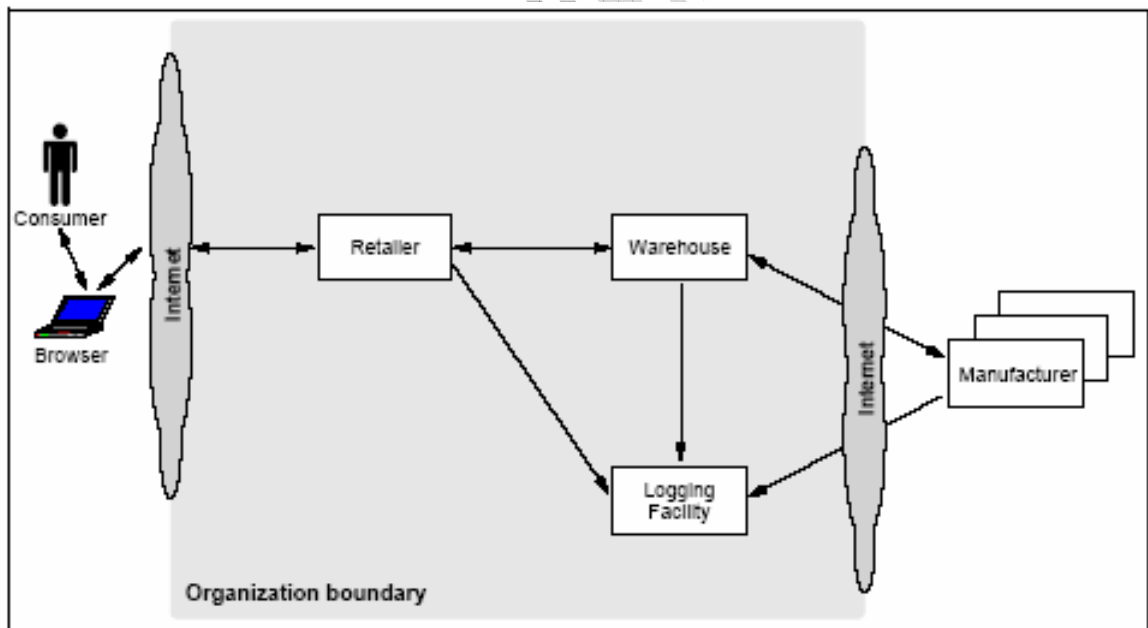
- ▶ Làm sao để lựa chọn một chuẩn định dạng thông điệp trao đổi giữa các chuẩn? Làm sao có thể xây dựng một chuẩn định dạng dữ liệu mà mọi hệ thống đều có khả năng hiểu và xử lý.

Ngoài các khó khăn trên mỗi tổ chức với mỗi đặc thù riêng của mình có thể sẽ phải đối diện với các vấn đề khác trong quá trình triển khai hệ thống

3.2 Xây dựng hệ thống SOA

3.2.1 Giới thiệu bài toán

Phần này sẽ giới thiệu các giai đoạn cần tiến hành để triển khai một hệ thống SOA đạt hiệu quả cao. Để trình bày vấn đề một cách trực quan hơn, ta sẽ thực hiện xây dựng một giải pháp tổng thể cho bài toán “Bán hàng qua mạng”



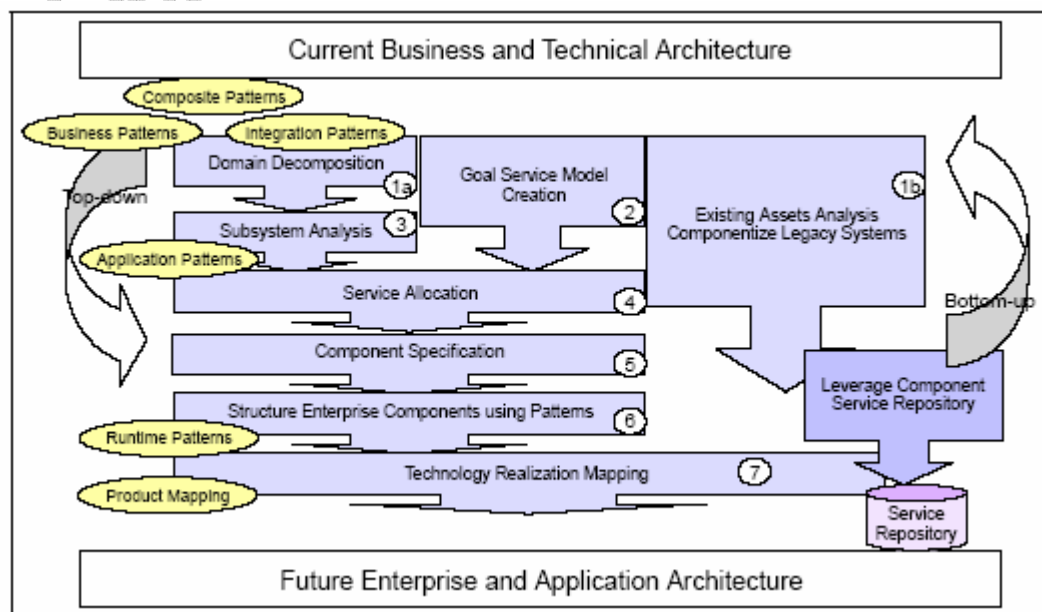
Mô tả bài toán:

- Khách hàng (customer) sẽ truy cập vào website của đại lý (retailer) để xem qua các mặt hàng và đặt hàng các sản phẩm điện tử như là TV, đầu DVD và video camera.
- Đại lý chuyển yêu cầu (order) đến cho bộ phận thủ kho (warehouse) để xử lý.
- Nếu nguồn hàng trong kho dưới mức cho phép thì yêu cầu bổ sung hàng (replenishment order) được gửi đến nhà sản xuất (manufacturer).
- Nhà sản xuất không giải quyết ngay yêu cầu bổ sung hàng, mà có thể sau một khoảng thời gian nào đó khi sản xuất đủ lượng hàng.

3.2.2 Một số khái niệm

Phần này sẽ giới thiệu các phương pháp để xác định các services theo phương pháp top-down (xuất phát từ các yêu cầu nghiệp vụ) và bottom-up (xuất phát từ thực trạng của hệ thống hiện có). Cụ thể hơn, sẽ trình bày một số kỹ thuật cũng như là mô tả các bước cần tiến hành để xây dựng hay chuyển đổi một hệ thống sẵn có theo mô hình SOA.

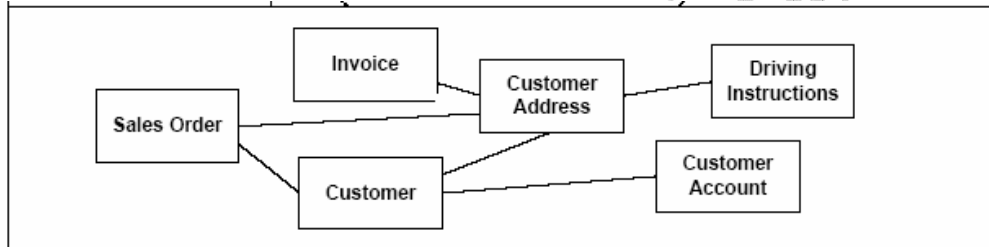
Phương pháp này bao gồm 7 bước chính được thể hiện ở Hình 3-1. Các bước này không nhất thiết phải được thực hiện một cách tuần tự và tuyến tính, mà quy trình có thể được điều chỉnh bằng cách quyết định các bước nào cần được thực hiện song song, và sự ràng buộc giữa các bước là như thế nào tùy vào các đặc trưng của hệ thống cụ thể cần triển khai.



Hình 3-1 - Các bước cần thực hiện khi triển khai một hệ thống SOA.

Chúng ta sẽ đi theo qui trình từ trên xuống (theo như trong Hình 3-1). Với các bước mà song song nhau thì trong thực tế có thể tiến hành cùng một lúc. Một số khái niệm cần quan tâm :

- Miền nghiệp vụ (Business domain): Là một miền hay môi trường trong đó diễn ra hoạt động tương tác (như trao đổi tài nguyên) giữa các tác nhân nghiệp vụ (economic agents), như là mua bán, sản xuất, dịch vụ ...



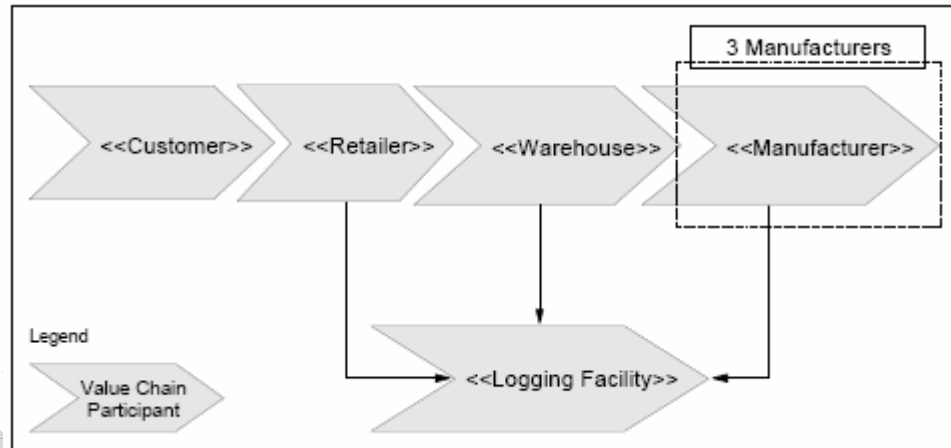
- Tiến trình nghiệp vụ: Là một hoạt động thực hiện trong quá trình quản lý nghiệp vụ một cách thủ công hay tự động. Mọi tiến trình đều cần dữ liệu đầu vào và cho ra một kết quả khi kết thúc. Tùy thuộc vào độ phức tạp mà một tiến trình có thể là một tác vụ đơn giản hay là một thủ tục phức tạp
- Tiến trình con: Là một tiến trình được thực hiện như một phần của tiến trình khác. Tiến trình con có thể được “trừu tượng hóa” để che dấu chi tiết bên trong và “cụ thể hóa” để thể hiện đầy đủ quy trình thực hiện bên trong.
- Sơ đồ nghiệp vụ sử dụng: Tập trung hơn vào mô tả các qui trình xử lý, còn các vấn đề liên quan đến kỹ thuật, cách cài đặt... chỉ được mô tả một cách tổng quát, trừu tượng, như là chỉ nêu lên những dự định (intentions).
- Sơ đồ hệ thống: Trong sơ đồ hệ thống mô tả rõ ràng những quyết định (decisions) liên quan đến cài đặt, triển khai, kỹ thuật, ví dụ như trong khi mô tả sẽ sử dụng các thuật ngữ như system, report, screen...)
- Hệ thống con: Một hệ thống sẽ được chia thành nhiều hệ thống con chịu trách nhiệm về một hay một nhóm chức năng của hệ thống. Hệ thống con sẽ được xây dựng như những thành phần độc lập để có thể sử dụng lại.
- Thành tố: Một hệ thống gồm nhiều thành phần, và một thành phần đảm nhiệm việc thực thi một nhóm chức năng có liên quan của hệ thống. Một thành phần sẽ chứa nhiều dịch vụ và cung cấp các dịch vụ này ra bên ngoài như là thành phần giao tiếp.

- Thành phần nghiệp vụ: Thành phần nghiệp vụ là một thành phần cài đặt của một chức năng hay qui trình nghiệp vụ. Thành phần nghiệp vụ được xây dựng như những thành đơn vị độc lập và có khả năng tái sử dụng của hệ thống. Một vài ví dụ của thành phần nghiệp vụ như: quản lý danh mục hàng hóa, quản lý thanh toán, quản lý đặt hàng và tính thuế...
- Thành phần kỹ thuật: thành phần kỹ thuật thực thi các chức năng kỹ thuật trong cơ sở hạ tầng của hệ thống (như security component , scheduling component), độc lập với các chức năng nghiệp vụ.
- Value-chain : Là khái niệm nhằm chỉ các loại hoạt động nghiệp vụ được thực hiện với mục đích nâng cao lợi nhuận của doanh nghiệp, bao gồm các hoạt động như thu thập nguyên vật liệu, sản xuất, phân phối sản phẩm, marketing, chăm sóc khách hàng....
- Vùng chức năng: Là khái niệm dùng để mô tả một bộ phận chịu trách nhiệm cho một nhóm các chức năng có liên quan như là về khách hàng, sản xuất, phân phối sản phẩm, lưu trữ kho...
- Top-down : trong xây dựng một hệ thống SOA, thì phương pháp top-down là phương pháp mà điểm xuất phát của nó sẽ là từ các yêu cầu nghiệp vụ để xác định các yêu cầu chức năng, các tiến trình và tiến trình con, các trường hợp sử dụng (use cases) để tiến tới việc xác định các thành phần hệ thống (components), các dịch vụ...
- Bottom-up : phương pháp này sẽ dựa trên việc phân tích tình trạng, các tài nguyên của hệ thống hiện có và sẽ tái sử dụng lại những thành phần này trong việc xây dựng các dịch vụ mới.

3.2.3 Các bước xây dựng hệ thống SOA

3.2.3.1 Phân rã domain

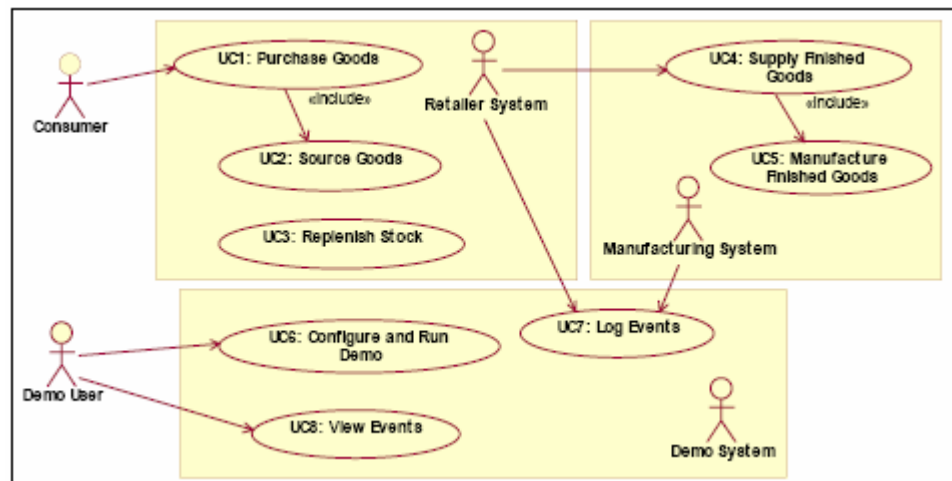
Ở giai đoạn này ta có thể sử dụng kỹ thuật top-down để phân rã domain (toàn bộ quy trình nghiệp vụ) thành các quy trình nghiệp vụ, tiến trình con và sơ đồ sử dụng. Nếu xem xét ở khía cạnh nghiệp vụ thì một domain bao gồm nhiều vùng chức năng.



Hình 3-2 – Phân rã domain thành một dãy các vùng chức năng liên quan

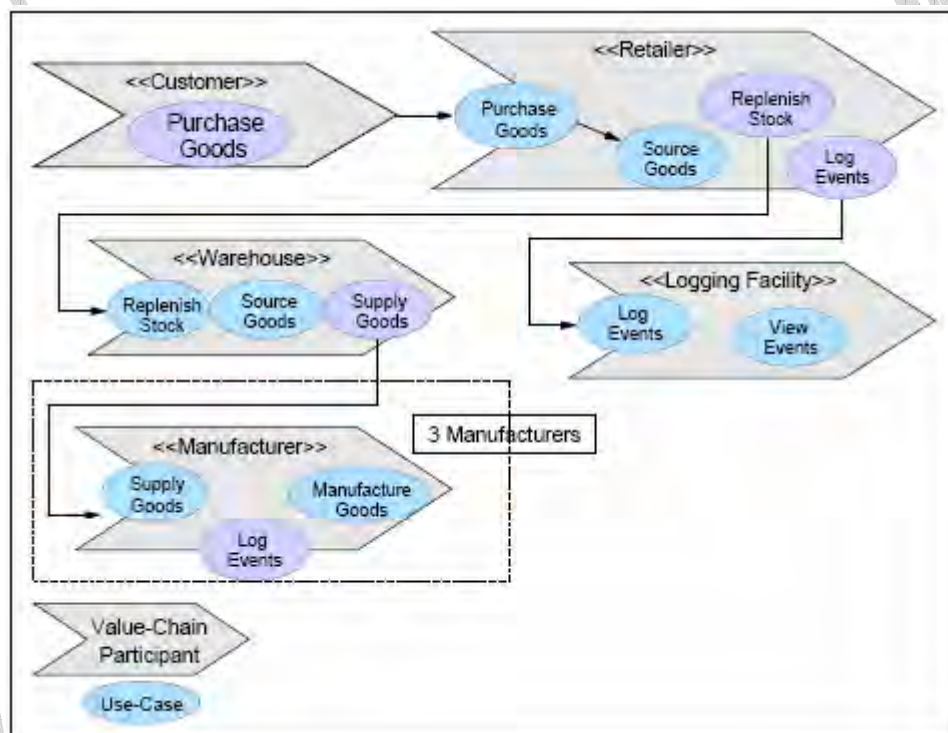
Sau khi phân rã domain thành một dãy các vùng chức năng liên quan, ta tiếp tục phân tích từng vùng chức năng để xác định các sơ đồ sử dụng.

- Mô hình use case:
 - ▶ UC1: Purchase Goods (khách hàng chọn và mua hàng từ danh sách nhà bán lẻ cung cấp)
 - ▶ UC2: Source Goods (nhà bán lẻ lấy hàng từ kho để giao cho khách hàng)
 - ▶ UC3: Replenish Stock (yêu cầu bổ sung hàng khi lượng hàng trong kho dưới mức sàn).
 - ▶ UC4: Supply Finished Goods (xử lý yêu cầu bổ sung hàng)
 - ▶ UC5: Manufacture Finishes Goods (sản xuất thêm hàng để đáp ứng yêu cầu khi lượng hàng hiện có không đủ cung cấp)
 - ▶ UC6: Configure and Run Demo (chạy demo ứng dụng).
 - ▶ UC7: Log Events (theo dõi và lưu lại các hoạt động được thực hiện bởi các hệ thống)
 - ▶ UC8: View Events (xem lại thông tin hoạt động đã được lưu lại trước đó.)



Hình 3-3 – Sơ đồ sử dụng của hệ thống bán hàng

- Các use case xác định được sẽ là những “ứng cử viên” cho các dịch vụ mà cuối cùng sẽ được cung cấp như những Web service trong các thành phần của hệ thống. Điều này giúp chúng ta đạt được mục tiêu đó là “tiến trình nghiệp vụ tương ứng với hệ thống thông tin”.



Hình 3-4 – Sơ đồ các use case và quy trình nghiệp vụ

Phân tích các use case để xác định các use case nghiệp vụ và quy trình nghiệp vụ: Với mỗi use case nghiệp vụ, ta cần xác định dữ liệu vào và dữ liệu ra. Các thông tin dữ liệu tại thời điểm này có thể còn ở mức trừu tượng, tuy thế ta vẫn đảm bảo tính đầy đủ của nó vì các thông tin này sẽ được tinh chế trong các giai đoạn sau khi thiết kế các dịch vụ và thành phần.

Chúng ta đang ở trong giai đoạn phân tích, và khi chuyển sang giai đoạn thiết kế thì mỗi vùng chức năng sẽ được ánh xạ thành một hay nhiều các hệ thống con và các use case nghiệp vụ sẽ được ánh xạ thành các use case hệ thống.

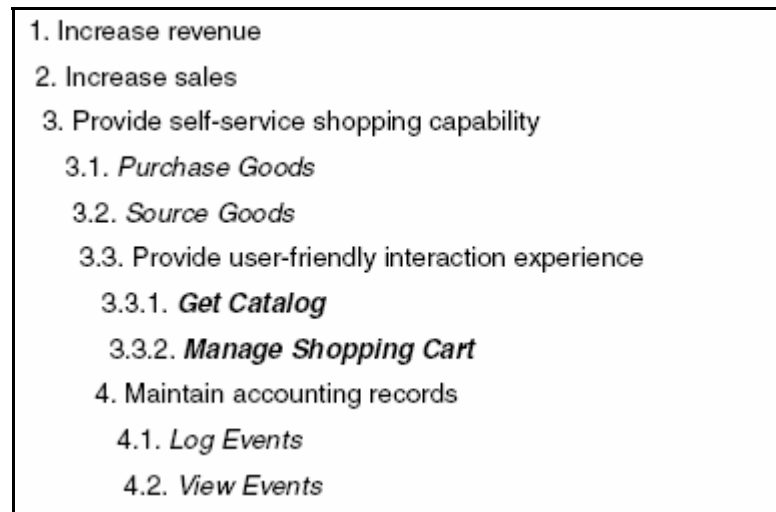
3.2.3.2 Xây dựng mô hình Goal-service

Trong giai đoạn phân rã domain ta đã xác định được các use case nghiệp vụ, và đây sẽ là cơ sở chính để ta xác định các dịch vụ. Trong giai đoạn này, chúng ta sẽ thiết kế mô hình goal-service để kiểm tra “các ứng cử viên” trên có thật sự là tốt không?

Thông qua việc phỏng vấn các đối tượng quản lý nghiệp vụ (business owners) ta sẽ hiểu các mục tiêu trong công việc của họ là gì. Ta sẽ xuất phát từ mục tiêu chính (goal) và từng bước phân tích để xác định các công việc cần làm để đạt được mục tiêu chính đó là gì (sub-goal). Quá trình phân rã như thế (goal thành các sub-goal) sẽ được thực hiện cho tới khi nào ta thấy rằng “mục tiêu này có thể được thực hiện bởi một dịch vụ nào đó”.

Như vậy các dịch vụ sẽ gắn liền với các sub-goal nào mà nó cần thực thi trong mô hình goal-service. Điều này sẽ làm cho các dịch vụ có thể truy vết tới business goal (mục tiêu chính). Đây là một đặc điểm quan trọng để đảm bảo rằng toàn bộ các dịch vụ nghiệp vụ là có thể xác định được trong thời gian sớm nhất.

Có rất nhiều cách để thể hiện mô hình goal-service. Ở đây, ta sử dụng một cách đơn giản đó là dùng danh sách phân cấp lồng nhau để biểu diễn các goal, sub-goal và dịch vụ. Dưới đây là một ví dụ về mô hình goal-service của nghiệp vụ bán lẻ



Hình 3-5 – Ví dụ về mô hình goal-service

- Goal: được thể hiện bằng font chữ bình thường
- Dịch vụ: được thể hiện bằng font chữ in nghiêng
- Giá trị gia tăng (đây là những dịch vụ cần thiết, nhưng chưa được xác định trong giai đoạn domain decomposition): thể hiện bằng font chữ in đậm.
- Giải thích ý nghĩa mô hình:
 - ▶ Bắt đầu với mục tiêu chính (business goal) đó là increasing revenue (tăng doanh thu). Điều này đạt được nếu ta increasing sales (tăng số lượng bán). Để tăng số lượng bán, ta cung cấp self-service shopping (hỗ trợ bán hàng tự động), hai use case “Purchase Goods” và “Source Goods” (đã được xác định trong bước phân rã domain) xử lý yêu cầu này.
 - ▶ Nếu xét kỹ càng hơn về goal self-service shopping, thì ta thấy rằng sẽ tốt hơn nếu ta có những hỗ trợ tính tiện dụng và thân thiện cho người dùng (provide user-friendly interaction experience). Để giải quyết vấn đề này, thì ta cần cung cấp cho người dùng *shopping catalog* để xem qua các sản phẩm, đồng thời hỗ trợ *shopping card* để thực hiện thanh toán qua mạng.
- Trong quá trình xây dựng mô hình goal-service này, ta sẽ xác định thêm các dịch vụ cần thiết.

3.2.3.3 Phân tích hệ thống con

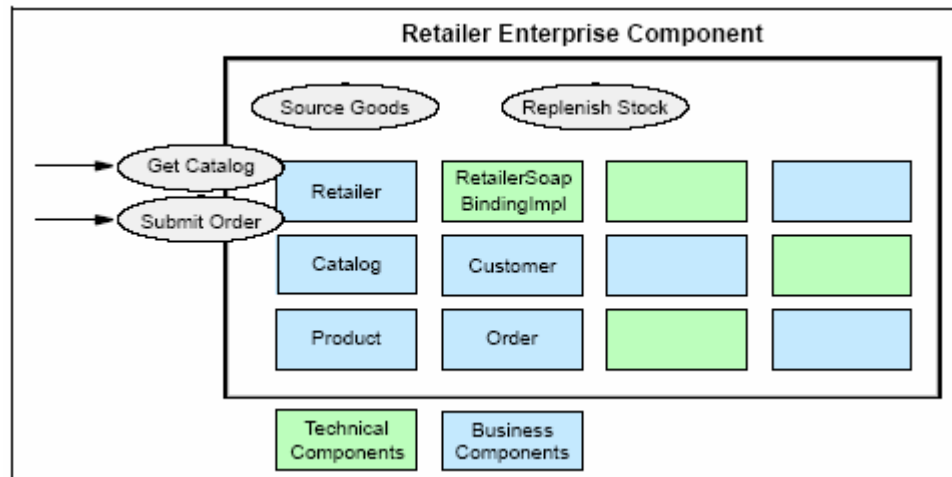
Trong giai đoạn này, ta sẽ đi sâu hơn trong việc thiết kế và xây dựng kiến trúc hệ thống. Các use case nghiệp vụ sẽ là cơ sở để thiết kế các use case hệ thống.

Hệ thống con bao gồm các thành phần nghiệp vụ (như là Customer, Order và Product) và các thành phần kỹ thuật (như là messaging, security và logging). Trong suốt giai đoạn phân tích hệ thống con, các thành phần nghiệp vụ và thành phần kỹ thuật sẽ được xác định như sau:

- Phân tích luồng xử lý bên trong của hệ thống con (thường là một chuỗi các use case) để tìm ra các “ứng cử viên” cho thành phần nghiệp vụ.
- Phân tích các yêu cầu phi chức năng để tìm ra các thành phần kỹ thuật.
- Xác định các chức năng được yêu cầu cho mỗi thành phần nghiệp vụ, nghĩa là, xác định các use case hệ thống mà các thành phần này phải hỗ trợ.

Các use case nghiệp vụ được xác định trong giai đoạn phân rã domain thường là những “ứng cử viên” tốt để đưa vào tầng giao tiếp (interface) của các thành phần của hệ thống con, nhằm cung cấp các dịch vụ bên trong của thành phần. Các use case nghiệp vụ này sẽ kết hợp với nhau để hỗ trợ cho một quy trình nghiệp vụ.

Trong bài toán của ta, bốn hệ thống con là Retailer, Warehouse, Manufacturer và Logging Facility. Mỗi hệ thống con cung cấp một tập các service nghiệp vụ. Hình 3-6 minh họa hệ thống con Retailer. Sau khi tạo mô hình goal-service xong, ta phân tích use case nghiệp vụ “Purchase Goods” thành hai dịch vụ là “Get Catalog” và “Submit Order”.



Hình 3-6 – Các use case nghiệp vụ được “gắn” trên hệ thống con

Mỗi use case nghiệp vụ tương ứng với một tập các use case hệ thống được đóng gói trong hệ thống con. Hệ thống con sẽ sử dụng các thành phần nghiệp vụ và thành phần kỹ thuật để thực thi các use case hệ thống và hỗ trợ cho dịch vụ nghiệp vụ đã được cung cấp ra ngoài (như là Submit Order).

Mô hình thể hiện ở Hình 3-6 chỉ nhằm mục đích minh họa. Trong thực tế, các kỹ thuật mô hình hóa sử dụng chuẩn UML sẽ được sử dụng. Sau khi kết thúc giai đoạn phân tích hệ thống con, ta sẽ có được các thành phần được xây dựng như là các dịch vụ.

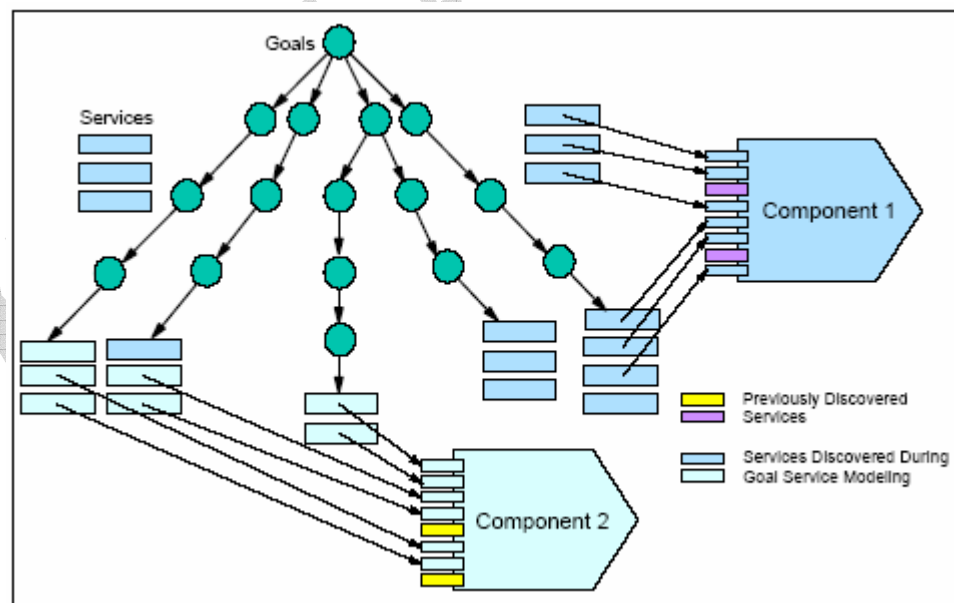
- **Retailer** dịch vụ cung cấp chức năng truy cập vào danh sách hàng hóa và đặt hàng.
- **Warehouse** dịch vụ hỗ trợ gửi hàng đã đặt và cập nhật tồn kho khi xuất nhập hàng. Mỗi khi lượng hàng tồn kho thấp hơn mức sàn, thì nó sẽ gửi “Purchase Order” (PO) đến cho nhà sản xuất để xử lý.
- **Warehouse Callback** dịch vụ nhận thông tin phản hồi từ phía nhà sản xuất về kết quả xử lý PO rằng có thành công hay không.
- **Manufacturer:** dịch vụ nhận PO và bắt đầu quá trình sản xuất.
- **Login:** dịch vụ theo dõi thông tin diễn biến của các hoạt động đã xảy ra và hỗ trợ cho người dùng cuối xem lại thông tin này.

Tại thời điểm này, các dịch vụ trên đã có thể kết hợp (orchestration) để tạo nên các dịch vụ tổng hợp hỗ trợ quy trình nghiệp vụ.

3.2.3.4 Phân bổ dịch vụ

Ta đã xác định được tất cả các dịch vụ cần thiết qua hai giai đoạn là phân rã domain và xây dựng mô hình goal-service. Trong giai đoạn này, chúng ta sẽ thực hiện “phân bổ” các dịch vụ này vào các thành phần.

Phân bổ dịch vụ sẽ xác định xem thành phần nào sẽ cung cấp phần thực thi và quản lý cho mỗi dịch vụ. Phân bổ dịch vụ sẽ thể hiện tính năng truy vết giữa các dịch vụ và các thành phần chịu trách nhiệm thực thi và quản lý chúng.



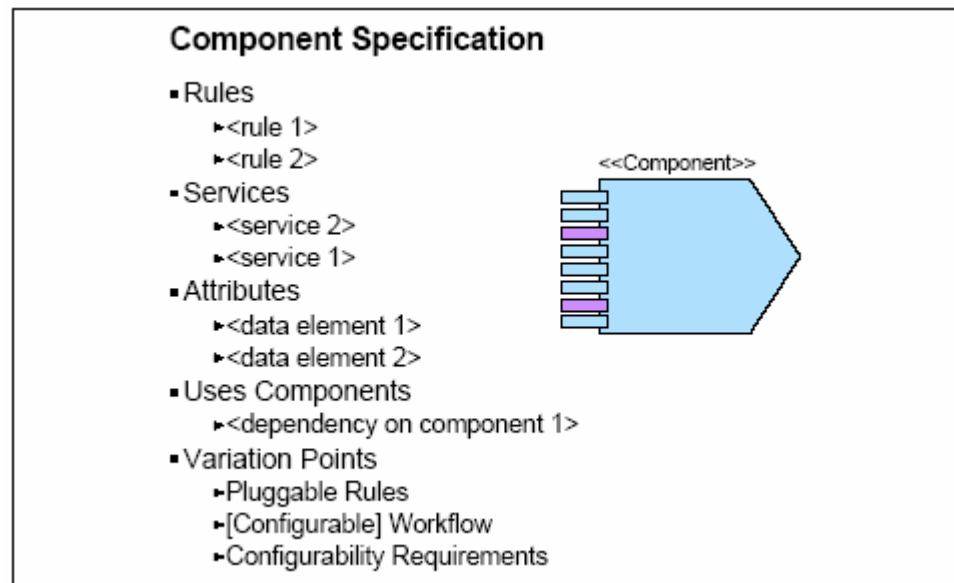
Hình 3-7 – Phân bổ dịch vụ

Trong bài toán của ta thì giai đoạn này tương đối đơn giản vì số lượng các dịch vụ không nhiều.

3.2.3.5 Đặc tả thành tố

Sau giai đoạn phân tích hệ thống con, ta đã xác định được interface của các hệ thống con, các use case hệ thống, thành phần nghiệp vụ và kỹ thuật. Và ta cũng đã thực hiện gán các dịch vụ vào trong các thành phần.

Trong giai đoạn này sẽ tiến hành xây dựng các đặc tả cho từng thành phần. Mẫu của đặc tả này được thể hiện ở Hình 3-8



Hình 3-8 – Mẫu đặc tả thành phần

3.2.3.6 Cấu trúc thành phần và dịch vụ

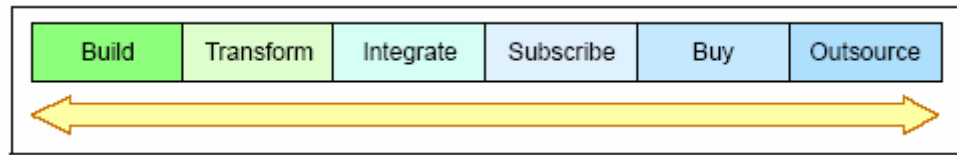
Ta đã xác định mối liên kết giữa thành phần và dịch vụ, và cũng đã xây dựng đặc tả của các thành phần. Trong giai đoạn này ta sẽ thực hiện phân bố các dịch vụ và các thành phần vào các tầng của SOA. Khi thực hiện giai đoạn này ta cần cân nhắc kỹ càng trước khi đưa ra quyết định, vì kết quả của giai đoạn này không chỉ quyết định kiến trúc của hệ thống mà còn ảnh hưởng đến các kỹ thuật, công nghệ đã được thiết kế và sử dụng để triển khai hệ thống.

3.2.3.7 Lựa chọn giải pháp thực thi

Khi các đặc tả chức năng của dịch vụ và thành phần đã được xác định một cách chi tiết, thì giai đoạn tiếp theo là xác định cơ chế, môi trường để thực thi các đặc tả đó. Quyết định này cũng đóng một vai trò rất quan trọng.

Ta có thể thực hiện một trong các lựa chọn sau:

- Xây dựng các thành phần mới hoàn toàn
- Chuyển đổi các hệ thống cũ để tái sử dụng lại các chức năng đã được dịch vụ hóa.
- Thực hiện tích hợp hệ thống cũ vào các hệ thống mới
- Mua các sản phẩm của hãng khác và tích hợp vào hệ thống của mình
- Đăng ký và thuê (outsource) một số phần chức năng thông qua web service



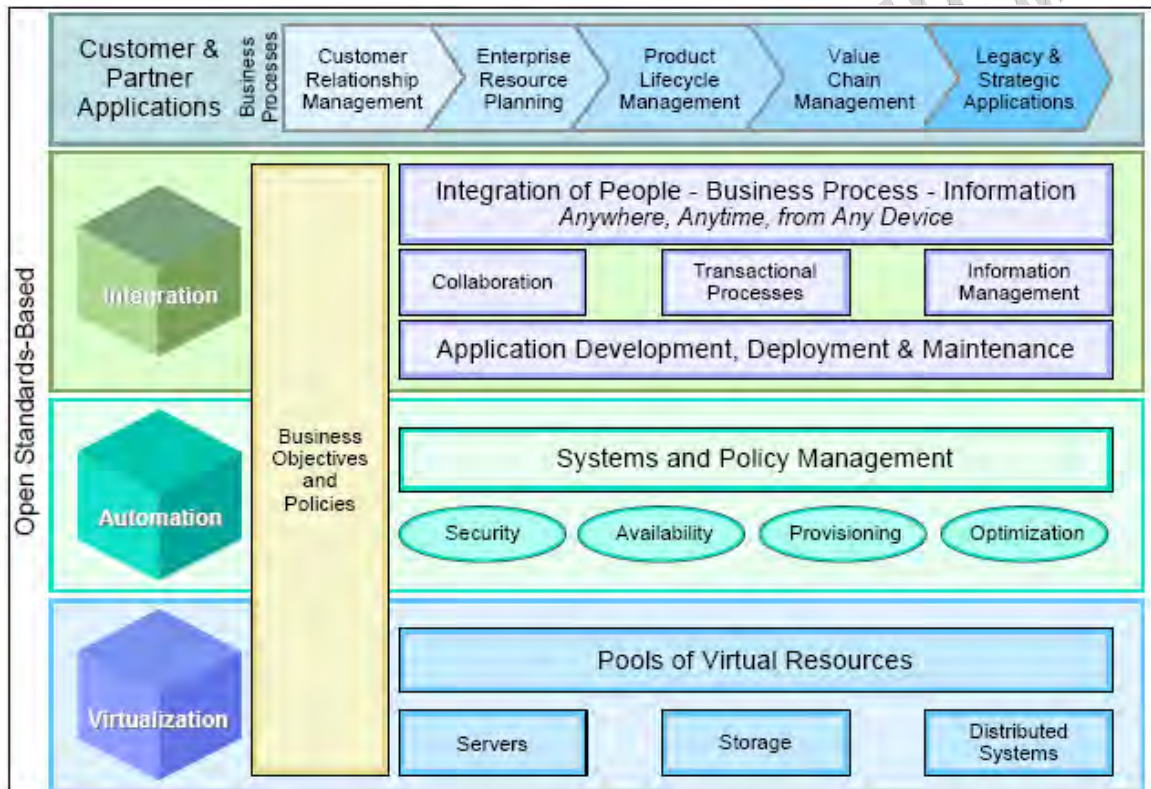
Hình 3-9 – Chọn lựa một giải pháp thực thi thích hợp.

Phản xử lý của một dịch vụ có thể sử dụng lại các hệ thống cũ với một trong các cách sau:

- Bao bọc (wrapping) hệ thống cũ lại bằng một dịch vụ xử lý thông điệp theo hàng đợi hay một Web service. Nhưng đôi khi giải pháp này không thật sự hiệu quả vì phải thể hiện toàn bộ chức năng của hệ thống ra ngoài.

Thành phần hóa những phần nào của hệ thống cũ để chỉ cung cấp các chức năng cần thiết ra bên ngoài. Quá trình này còn được gọi là chuyển đổi (transformation). Cần quan tâm đến vấn đề “giới hạn” (scope) khi thực hiện quá trình này, tránh chuyển đổi cả những phần không cần thiết.

3.3 Triển khai SOA trong thực tế



Hình 3-10 – Sơ đồ tổng quan về thương mại điện tử theo yêu cầu

Doanh nghiệp muốn tập trung vào hoạt động kinh doanh chính, giảm chi phí và sử dụng lại những thông tin có sẵn theo những cách mới mà không phải đại tu lại toàn bộ kiến trúc có sẵn của họ. Luôn có một áp lực đòi hỏi phải cân bằng giữa nhu cầu tính linh hoạt, tiết kiệm chi phí và hiệu quả. Phần sau sẽ phân tích sơ lược các đặc trưng kinh doanh và công nghệ nền tảng. Hình 2-10 minh họa các thành phần chính của “thương mại điện tử theo yêu cầu” (*e-business on demand*)

3.3.1 Các đặc trưng chính về kinh doanh

Ở góc nhìn của doanh nghiệp, thương mại điện tử theo yêu cầu là cung cấp một cách thức cho những công ty cân đối lại hoạt động kinh doanh và môi trường công nghệ phù hợp với yêu cầu tái sử dụng lại những chức năng nghiệp vụ. Các đặc trưng thể được tóm gọn trong những ý sau :

➤ Tập trung

Cho phép doanh nghiệp tập trung vào các hoạt động kinh doanh chính, điều khiến họ thành công và nổi trội. Các chiến lược cộng tác cũng được hình thành từ đây để nhằm huy động vốn từ những nguồn này.

➤ Trách nhiệm

Là khả năng ứng phó nhanh với các yêu cầu từ phía khách hàng, những mối hiểm nguy từ bên ngoài và nắm bắt cơ hội thị trường.

➤ Đa dạng

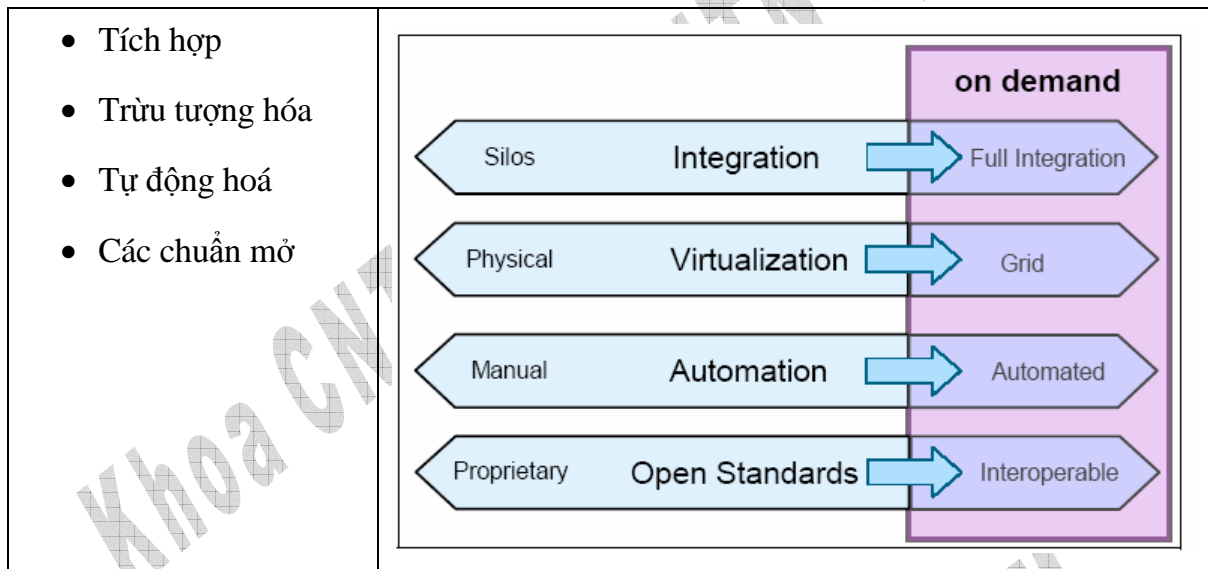
Nhằm đạt được tính linh hoạt trong hoạt động và quy trình nghiệp vụ. Nhằm thích ứng với điều kiện giá cả khác nhau, tăng năng suất hoạt động.

➤ Vững vàng

Tính vững vàng giúp doanh nghiệp đáp ứng lại những thay đổi cả ở trong kinh doanh lẫn môi trường công nghệ. Điều này còn giúp quản lý thay đổi, quản lý rủi ro và ước lượng doanh thu.

3.3.2 Các đặc trưng chính về công nghệ

Thương mại điện tử theo yêu cầu cần được hỗ trợ bởi một kiến trúc công nghệ được định nghĩa, mô tả chi tiết rõ ràng. Các đặc trưng về công nghệ này mang lại khả năng linh hoạt, trách nhiệm và hiệu quả mà doanh nghiệp cần có:



3.3.2.1 Tích hợp

Thành phần cơ bản của cấu trúc theo yêu cầu (on demand infrastructure) là *tích hợp*. Năm 2002, Sam Palmisano, Chief Executive Officer của IBM định nghĩa on demand như sau : “Một hoạt động kinh doanh theo yêu cầu (on demand business) là một tổ chức có quy trình nghiệp vụ, tích hợp với các đối tác chính, các nhà cung cấp và khách hàng mà vẫn có thể đáp ứng nhanh mọi yêu cầu của khách hàng, mỗi hiểm nguy từ bên ngoài và nắm bắt được cơ hội trên thị trường” .

Quá trình tích hợp có thể diễn ra ở nhiều mức độ khác nhau :

Con người

Quy trình

Ứng dụng

Hệ thống

Dữ liệu

3.3.2.2 Trừu tượng hóa

Nhiều công nghệ trong đời sống thường ngày của chúng ta khai thác khái niệm trừu tượng hóa như điện thoại di động, PDA, kết nối mạng không dây, máy in, vân vân.... Các khía cạnh của trừu tượng hóa còn gây tác động ảnh hưởng sâu rộng đến các quan niệm kiến trúc như thiết kế và phát triển hướng đối tượng, Web Service và XML.

3.3.2.3 Tự động hoá

Tính toán tự động giúp giải quyết nhu cầu của một tổ chức muốn giới hạn thời gian và chi phí xuất phát từ những nguyên nhân sau:

- Chi phí cao cho ứng dụng mới và nhân công chất lượng cao
- Thời gian tiêu phí cho những nền tảng công nghệ khác hẳn nhau thậm chí ở bên trong cùng một tổ chức.
- Ngân sách IT chỉ cho bảo trì chứ không phải cho giải pháp giải quyết vấn đề.
- Tính phức tạp giữa những hệ thống không đồng nhất.

Vậy thì làm cách nào một tổ chức xác định được các vấn đề liên quan sử dụng môi trường thực thi theo yêu cầu (on demand Operating Enviroment) ? Lúc này là lúc cần đến khả năng tính toán động (autonomic computing). Khả năng tính toán động có thể được tóm gọn trong 4 ý chính :

➤ **Tự hồi phục**

Là khả năng duy trì chức năng hoạt động của một hệ thống bằng cách dò tìm, ngăn ngừa và phục hồi sau khi bị lỗi mà chỉ cần một sự can thiệp tối thiểu hoặc không cần can thiệp từ phía con người. Yêu cầu này tỉ lệ với môi trường càng ngày càng nhiều với kiến trúc công nghệ. Nhu cầu tự hồi phục tăng khi yêu cầu về khả năng đáp ứng của tổ chức tăng.

➤ **Tự cấu hình**

Khả năng thích ứng động với sự thay đổi của môi trường, thêm hoặc bớt thành phần từ các hệ thống và thay đổi môi trường để thích nghi với yêu cầu công việc khác nhau.

➤ **Tự tối ưu hoá**

Tự động chọn cấu hình đạt hiệu quả làm việc tối ưu bao gồm việc tối ưu tài nguyên và quản lý công việc. Điều này giúp giảm bớt gánh nặng khan hiếm tài nguyên phục vụ cho các yêu cầu thường xuyên. Mục tiêu là điều chỉnh hệ thống nhằm đáp ứng được với những thay đổi trong công việc. Các hệ thống phải kiểm tra và tự điều chỉnh liên tục để có thể đáp ứng và thay đổi phù hợp với môi trường xung quanh.

➤ **Tự bảo vệ**

Bảo mật là một trong những yêu cầu cần được quan tâm trước khi các tổ chức chuẩn bị chia sẻ dữ liệu ra bên ngoài. Khả năng tự bảo vệ (self-protecting) yêu cầu hệ thống phải cung cấp một cách thức an toàn để bảo vệ thông tin và dữ liệu. Qua trình tự bảo vệ làm việc theo quy tắc dự đoán, dò tìm, nhận dạng và bảo vệ hệ thống khỏi những hiểm nguy từ bên ngoài lẫn bên trong.

3.3.3 Các chuẩn mở

Các chuẩn mở ảnh hưởng đến môi trường thực thi theo yêu cầu giữa các tầng như khả năng tự động hoá, tích hợp và trừu tượng hóa. Các chuẩn mở là nhân tố quan trọng ảnh hưởng đến tính linh hoạt và khả năng cộng tác giữa những hệ thống không đồng nhất.

Tính phổ biến toàn cầu của đặc tả chuẩn cho phép các hệ thống tách biệt tương tác lẫn nhau. Các nền tảng bên dưới có thể khác hẳn và độc lập với nhau nhưng các chuẩn mở cho phép xây dựng những tiến trình bất kể sự khác biệt đó.

Các chuẩn mở cung cấp cho môi trường thực thi thương mại điện tử theo yêu cầu (e-business on demand Operating Environment) một cơ chế mở và đúng chuẩn để có thể triệu gọi những service hệ thống.

3.3.4 Kiến trúc hướng dịch vụ và Thương mại điện tử theo yêu cầu

SOA được định nghĩa là một cách tiếp cận trong việc định ra những kiến trúc tích hợp dựa trên khái niệm dịch vụ. Các chức năng nghiệp vụ và kiến trúc được cung cấp

dưới dạng những dịch vụ. Đây là khái niệm nền tảng của hệ thống. SOA sử dụng Web Service như một tập các chuẩn linh hoạt và có khả năng cộng tác cho các hệ thống phân tán. Có sự bổ sung qua lại tự nhiên giữa SOA và Web Service, SOA nhấn mạnh đến 4 thành phần của thương mại điện tử theo yêu cầu theo cách sau :

➤ **Các chuẩn mở**

- SOA cung cấp một cách thức triệu gọi chuẩn đến các Web Service cho các tổ chức chia sẻ sử dụng qua mạng.
- Web Service sử dụng các chuẩn mở cho phép kết nối đa doanh nghiệp qua mạng và internet
 - ▶ Giao thức thông điệp (SOAP)
 - ▶ Giao thức truyền tải (HTTP, HTTPS, JMS)
 - ▶ Xử lý bảo mật ở tầng truyền tải (HTTPS) lẫn tầng giao thức (WS-Security)
- WSDL cho phép Web Service tự mô tả về mình

➤ **Tích hợp**

- Cung cấp các interface nhằm bao bọc các điểm cuối dịch vụ hướng đến xây dựng một kiến trúc độc lập hệ thống.
- SOA có thể cung cấp tính năng truy tìm và kết nối dịch vụ động (dynamic service discovery and binding), nghĩa là có thể tích hợp dịch vụ *theo yêu cầu*.

➤ **Virtualization**

- Một đặc tính quan trọng của SOA là các dịch vụ được triệu gọi mà không cần biết chi tiết cài đặt của service kể cả địa chỉ, nền tảng của service đó.

➤ **Tự động hoá**

Công nghệ Grid đang được áp dụng vào trong SOA để triển khai kiến trúc dịch vụ nhằm cung cấp một cách tiếp cận mới mẽ tăng tính tự động hoá.

✍ *Trong môi trường mở và phân tán như SOA, việc thiết lập một môi trường thật sự an toàn trong quá trình tương tác giữa các dịch vụ thật sự là một khó khăn. Vấn đề liên quan đến bảo mật an toàn hệ thống này sẽ được trình bày trong chương 4.*

Chương 4

SOA VÀ VẤN ĐỀ BẢO MẬT

✍ Chương này trình bày về các khó khăn gặp phải trong việc bảo vệ hệ thống SOA. Từ đó xem xét, phân tích một giải pháp đó là “kiến trúc bảo mật hướng dịch vụ”. Chương này cũng giới thiệu một số chuẩn bảo mật trong XML như WS-Security, XML-Signature, XML-Encryption, XML Key Management Specification, SAML và bộ thư viện WSE (Web Services Enhancements) hỗ trợ lập trình bảo mật web services.

4.1 Các thách thức về bảo mật trong hệ thống SOA

4.1.1 Đặt vấn đề

Với việc phát triển không ngừng của công nghệ web service đã tạo nên những ảnh hưởng nhất định trong việc xây dựng các mô hình tính toán phân tán. Các kiến trúc phân tán hướng đối tượng DOA (Distributed Object Architecture) sử dụng các công nghệ như là CORBA, DCOM, DCE, và Java RMI đang nhanh chóng chuyển sang các kiến trúc hướng dịch vụ (SOA) với những công nghệ mới như SOAP, HTTP và XML.

Việc thay đổi kiến trúc hệ thống như thế đã dẫn đến những thay đổi nhất định trong việc đưa ra những giải pháp cho vấn đề bảo mật của hệ thống. Hầu hết các giải pháp bảo mật đang được sử dụng ngày nay đều dựa trên thực trạng là cả hệ thống máy khách và máy chủ đều đặt tại cùng một mạng vật lý (ví dụ như là LAN) hay mạng logic (như VPN). Những giải pháp này đảm bảo an toàn cho hệ thống, thắt chặt vấn đề an ninh thông qua việc giám sát, điều khiển tất cả mọi ngõ vào và ngõ ra của mạng. Thế nhưng, một hệ thống SOA với các đặc tính mở của nó, đã thật sự làm cho các giải pháp này không còn thích hợp nữa.

- Nếu như trước đây khi người dùng muốn sử dụng các chức năng của hệ thống thì họ phải ngồi tại các thiết bị đầu cuối gắn liền với hệ thống mạng để truy cập, thì nay điều này không còn cần thiết nữa. Vì trong một hệ thống SOA, các chức năng này đã được “*dịch vụ hóa*” và cung cấp ra cho các đối tượng bên ngoài truy cập thông qua các nghi thức chuẩn của web service.
- Các dịch vụ về bản chất là không phụ thuộc vào vị trí. Địa chỉ của dịch vụ cũng có thể thay đổi vì một số lý do nào đó (nâng cấp hệ thống, hệ thống bị sự cố trong quá trình vận hành...). Điều này khiến cho việc kiểm soát các luồng tương tác giữa hệ thống và các đối tượng bên ngoài càng trở nên khó khăn hơn.
- Nhà cung cấp dịch vụ cũng như là phía đối tượng sử dụng các dịch vụ đó có thể thuộc về các mạng vật lý khác nhau của cùng một tổ chức, hay thậm chí là ở các tổ chức khác nhau. Thế cho nên sẽ dẫn đến những sự khác biệt về các chính sách an ninh, bảo mật, và quá trình kiểm soát sẽ gặp nhiều cản trở hơn.

4.1.2 Các vấn đề bảo mật liên quan cần quan tâm

4.1.2.1 Những hạn chế của tường lửa

Các hệ thống tường lửa thông thường đều không giám sát chặt chẽ các gói tin được truyền tải dựa trên giao thức HTTP, nghĩa là, các yêu cầu truy cập đến web service thông qua nghi thức HTTP đều được các hệ thống tường lửa cho phép qua. Sự thiếu sót này có thể khiến cho máy chủ có nguy cơ bị những cuộc tấn công mà không thể biết trước. Trong thực tế, đã có những cuộc tấn công bằng cách thiết kế các gói tin SOAP qua mặt được hệ thống tường lửa và máy chủ để gây nên lỗi “tràn vùng đệm” cho các ứng dụng bên trong.

Thời gian gần đây, rất nhiều những sản phẩm tường lửa giám sát những gói tin chứa dữ liệu dạng XML được xây dựng nhằm bảo vệ các web service trong việc kiểm soát các yêu cầu dạng SOAP. Tuy nhiên, tính hiệu quả của những sản phẩm này chưa đủ sức thuyết phục để các nhà quản trị chấp nhận sử dụng nó như một phần của kiến trúc an ninh hệ thống.

4.1.2.2 Cơ chế bảo mật chưa được định nghĩa một cách đầy đủ

Hầu hết những chuẩn về bảo mật hiện nay đều chỉ tập trung vào việc đưa ra các định dạng bảo vệ dữ liệu trong quá trình trao đổi, mà không quan tâm đến việc xác định các nghi thức mà các bên cần thực hiện khi tương tác, như là việc chứng thực và kiểm tra quyền...

4.1.2.3 Các giải pháp bảo mật khó tích hợp với nhau

Vì thiếu những chuẩn chung về việc chỉ ra *nghi thức giao tiếp* giữa những web service khiến cho các sản phẩm hỗ trợ cho vấn đề bảo mật của web service trên thị trường ngày nay không thể hoàn toàn tích hợp vào nhau, mặc dù các sản phẩm này đều được thiết kế dựa trên các chuẩn về bảo mật cho web service.

4.1.2.4 Bảo mật trong qui trình phối hợp hoạt động của các web service

Khi số lượng các web ngày càng gia tăng, thì nhu cầu tái sử dụng lại các dịch vụ này, kết hợp chúng theo những qui trình xử lý khác nhau để đạt được những kết quả khác nhau đang giành được rất nhiều sự quan tâm. Và lúc này, rõ ràng là ta phải giải quyết được vấn đề bảo mật trong mối quan hệ tương tác giữa các web service.

4.1.2.5 Bảo mật và vấn đề về hiệu suất hoạt động

Một hệ thống sử dụng cơ chế bảo mật khóa công cộng đòi hỏi phải thực hiện rất nhiều phép tính như là chứng nhận thông điệp, mã hóa và kiểm tra các giấy chứng nhận. Việc truyền gửi một thông điệp đã được chứng nhận sẽ chậm hơn rất nhiều lần so với việc truyền gửi một thông điệp thông thường. Và chắc chắn rằng, sẽ luôn luôn có một sự ràng buộc giữa mức độ bảo mật và hiệu suất hoạt động của hệ thống. Vấn đề đặt ra đó là, cần phải lên phương án, hoạch định kế hoạch chi tiết, kỹ càng, thận trọng những công nghệ tối ưu về hiệu quả nhằm đảm bảo rằng hệ thống SOA được an toàn, trong khi vẫn có hiệu suất hoạt động ở mức chấp nhận được.

4.2 Giới thiệu về kiến trúc bảo mật hướng dịch vụ

4.2.1 Một số yêu cầu đặt ra của kiến trúc

- Chứng thực
 - ▶ Hầu hết các nhà cung cấp dịch vụ đều yêu cầu các bên sử dụng dịch vụ phải được chứng thực trước khi yêu cầu sử dụng dịch vụ được chấp nhận.
 - ▶ Các đối tượng sử dụng dịch vụ cũng sẽ phải chứng thực nhà cung cấp dịch vụ khi nhận được các kết quả trả về.
 - ▶ Hệ thống nên hỗ trợ nhiều cơ chế chứng thực, và các cơ chế này phải đủ linh hoạt để có thể dễ dàng thay đổi theo các yêu cầu đặc trưng của dịch vụ.
- Phân quyền
 - ▶ Ngoài việc phải thỏa mãn các yêu cầu về chứng thực thì các đối tượng sử dụng dịch vụ cần phải có một quyền nhất định nào đó. Việc kiểm tra các quyền này thông qua các chính sách (ví dụ như, đối tượng nào được quyền sử dụng các dịch vụ, và trong điều kiện gì...).
 - ▶ Nên sử dụng nhiều mô hình phân quyền khác nhau:
 - Discretionary Access Control (DAC): mô hình này kiểm soát việc truy cập dựa trên ID của đối tượng muốn truy cập. ID này có thể là tên truy cập, mật khẩu, hay các dấu hiệu đặc trưng về phần mềm hay phần cứng...
 - Mandatory Access Control (MAC): mô hình này bảo vệ thông tin bằng cách gán cho mọi thông tin một giá trị “nhạy cảm” và sẽ so sánh giá trị này với giá trị “nhạy cảm” của người muốn truy cập. Đây sẽ là cơ sở để thực hiện cấp quyền truy cập thông tin. Mô hình này thích hợp cho các hệ thống đòi hỏi cao về độ an toàn.

- Role-Based Access Control (RBAC) : với mô hình này, quyết định cho phép truy cập dựa trên vai trò của từng cá nhân và trách nhiệm trong tổ chức của cá nhân đó.
- Độ tin cậy
 - ▶ Phải có cơ chế để bảo vệ môi trường truyền dữ liệu bên dưới cũng như là các thông điệp, và tài liệu được truyền trên môi trường đó sao cho chúng không bị truy cập bởi các đối tượng không có quyền.
- Toàn vẹn dữ liệu
 - ▶ Bảo vệ dữ liệu không bị xâm hại trong suốt quá trình truyền.
- Cơ chế định danh
 - ▶ Nhằm đảm bảo các đối tượng tham gia trong quá trình tương tác không thể phủ nhận vai trò của mình (người gửi không thể phủ nhận những gì mình đã gửi, và người nhận cũng không thể chối bỏ những gì mình đã nhận).
 - ▶ Yêu cầu này đặc biệt quan trọng trong môi trường thương mại ngày nay, khi mà các cuộc gặp gỡ tận mặt nhiều khi không thể thực hiện được.
- Có một cơ chế quản lý
 - ▶ Kiến trúc an ninh của hệ thống phải cung cấp cơ chế để quản lý các tính năng ở trên, bao gồm: quản lý người dùng, quản lý các chính sách bảo mật...
- Cơ chế ghi nhận
 - ▶ Thực hiện các tất cả các ghi nhận liên quan đến các quá trình tương tác của các đối tượng với hệ thống.
 - ▶ Tính năng này sẽ góp phần hỗ trợ cho yêu cầu về xây dựng một cơ chế định danh.
- Xử lý bảo mật liên miền
 - ▶ Kiến trúc phải cung cấp một mô hình đáng tin cậy nhằm bảo vệ quá trình tương tác giữa các web service trong những miền khác nhau

- Khả năng liên kết cao

- ▶ Khả năng dễ mở rộng, liên kết và tích hợp với các hệ thống khác là một đặc trưng nổi bật của hệ thống SOA. Vì thế, yêu cầu kiến trúc bảo mật khi được tích hợp vào sẽ vẫn duy trì tốt nhất đặc trưng này của SOA.
- ▶ Khả năng liên kết cao còn thể hiện ở việc cho phép kiến trúc của ta có thể tích hợp với những giải pháp, sản phẩm về an toàn dữ liệu khác mà không phải bỏ ra chi phí quá cao bởi lẽ kiến trúc ta xây dựng không hẳn là để thay thế hoàn toàn cho cơ sở kiến trúc hạ tầng hiện có.
- ▶ Tại những vị trí giao tiếp giữ vai trò quan trọng trong việc tích hợp, mở rộng hệ thống như: giữa nhà cung cấp và các đối tượng sử dụng dịch vụ, giữa nhà cung cấp dịch vụ và cơ sở hạ tầng của kiến trúc bảo mật bên dưới, và giữa những kiến trúc bảo mật của những miền khác nhau phải được thiết kế với các yêu cầu về tính ổn định, đồng nhất và hiệu quả dựa trên các chuẩn mở.

- Kiểm soát được những thay đổi về yêu cầu bảo mật

- ▶ Trong những giải pháp về bảo mật trước đây, mọi tài nguyên và dịch vụ đều dùng chung các chính sách về bảo vệ, an toàn. Giải pháp dùng chung như thế làm cho chi phí bỏ ra không cao, nhưng bù lại, hệ thống của ta sẽ quá lỏng lẻo hoặc quá cứng nhắc, không thích hợp với đặc trưng về tính đa dạng của các tài nguyên, dữ liệu, dịch vụ, chức năng...
- ▶ Trong một hệ thống SOA, các dịch vụ ở các tầng khác nhau sẽ đòi hỏi những mức độ bảo vệ khác nhau, nói cách khác là cần có những chính sách bảo mật khác nhau. Ví dụ như, một dịch vụ có thể yêu cầu chứng thực dựa trên X.509 certificate, trong khi một dịch vụ chỉ cần chứng thực theo tên/mật khẩu. Vì thế, các chính sách về an toàn của ta phải đầy đủ ngữ nghĩa, và đủ linh hoạt để đáp ứng được những sự thay đổi.

4.2.2 Khái niệm về kiến trúc bảo mật hướng dịch vụ SOSA (service-oriented security architecture)

Kiến trúc hướng dịch vụ (SOA) là một tập hợp các qui tắc cho việc thiết kế các dịch vụ có tính dễ mở rộng, khả năng kết hợp và tương tác cao. Các nguyên tắc này không chỉ có thể áp dụng cho các dịch vụ nghiệp vụ để hình thành các hệ thống nghiệp vụ SOA (service-oriented business architecture), mà có thể dùng cho các dịch vụ bảo mật để tạo nên các hệ thống bảo mật SOA (service-oriented security architecture), cho các dịch vụ quản lý để xây dựng các hệ thống quản lý SOA (service-oriented management architecture)...

Mô hình SOSA không hướng đến việc thay thế hoàn toàn các kiến trúc bảo mật hiện có, mà muốn đưa ra một giải pháp để liên kết các cơ sở hạ tầng có sẵn. Việc thay thế bằng các dịch vụ bảo mật mới hoàn toàn đòi hỏi rất nhiều chi phí. Hiện tại chúng ta cũng không có nhu cầu cho vấn đề này. Thay vào đó, chúng ta sẽ tái sử dụng lại (chứ không phải thay mới) những kỹ thuật, dịch vụ bảo mật hiện có dựa trên những nguyên tắc của SOA để tạo nên một kiến trúc bảo mật hướng dịch vụ mới. Và điều chúng ta thật sự muốn đó là tạo ra một tầng liên kết bên trên các dịch vụ, công nghệ bảo mật đó. Đây cũng là mục tiêu chính của các chuẩn mở về XML và web service mà đã và đang được phát triển: *“không thay thế những gì đã có, mà làm cho chúng liên kết với nhau trong một môi trường đồng nhất.”*

Yếu tố đầu tiên và quan trọng nhất mà ta cần phải quan tâm đến khi thiết kế tầng liên kết này đó là *“thiết lập được sự tin cậy”*.

- Theo định nghĩa của tổ chức OASIS (Organization for the Advancement of Structured Information Standards) thì **“sự tin cậy”** được định nghĩa như là một đặc điểm của một thực thể mà sẽ là cơ sở cho một thực thể khác dựa trên đó để thực hiện một số hành động và/hay xác nhận về đối tượng đó.

Để cho đối tượng sử dụng dịch vụ ủy quyền thực thi một tiến trình cho phía cung cấp dịch vụ thì một chính sách về sự tin cậy phải được thiết lập để làm cơ sở cho sự ủy quyền đó. Vì thế, mục tiêu cuối cùng của kiến trúc bảo mật hướng dịch vụ (SOSA) đó

là xây dựng được các sự tin cậy giữa các dịch vụ với nhau bằng cách thiết lập và thi hành các chính sách về bảo mật. Các chính sách này sẽ được xây dựng dựa trên “**mô hình ủy thác**”, trong đó sẽ định nghĩa các mối quan hệ tin tưởng lẫn nhau giữa các dịch vụ).

Các nghi thức, nguyên tắc, cơ chế vận hành trong SOSA không nên lệ thuộc vào một môi trường thực thi cụ thể nào, thay vào đó, nó nên dựa trên những chuẩn chung của “**mô hình ủy thác**”. Các nghi thức này bao gồm:

- Cơ chế định danh một đối tượng, như là đối tượng sử dụng dịch vụ, dịch vụ, tài nguyên, chính sách, nhà cung cấp dịch vụ. Có thể dùng cơ chế định danh URIs (Uniform Resource Identifier)
- Định nghĩa của những dấu hiệu mật và chính sách
 - ▶ Policy: là một tập hợp các cơ chế xác nhận policy
 - ▶ Cơ chế xác nhận (Policy Assertion): một policy assertion có thể xem như một quy tắc mà liên quan đến cách thức hoạt động của hệ thống (ví dụ như: loại security token nào cần được dùng, thông điệp trao đổi có cần được chứng thực, thời gian còn hiệu lực của thông điệp là bao lâu?...)
 - ▶ Security Token: Security Token có thể là dạng binary (X.509, Kerberos ticket) hay XML (SAML, XrML)
- Nghi thức tạo, trao đổi các token và policy:
 - ▶ Việc phát sinh và phân phối các chính sách nên được thực hiện cùng lúc với việc tạo và gửi thông tin định nghĩa về dịch vụ (WSDL, UDDI)
 - ▶ Việc tạo và phân phối các security token
 - ▶ Việc đặt các token trong các thông điệp trao đổi.
 - ▶ Việc kiểm tra xem các token có phù hợp với yêu cầu.

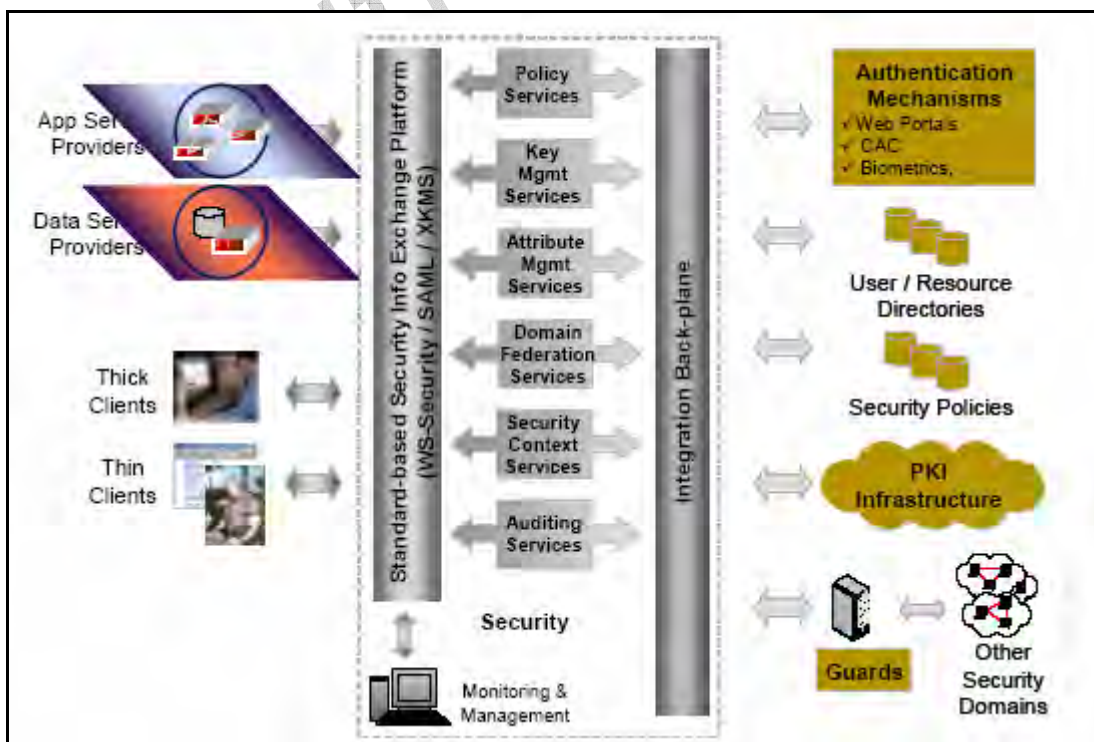
Nói cách khác, hệ thống SOSA sẽ phải xây dựng được:

- Các nghi thức chung dùng trong việc trao đổi các token của các đối tượng sử dụng dịch vụ.
- Các nguyên tắc chung về cách xử lý các token đó của phía cung cấp dịch vụ.

Nếu thiết lập được cơ chế quản lý các mối liên kết giữa các đối tượng sử dụng dịch vụ và nhà cung cấp dịch vụ thì hệ thống của ta sẽ vận hành một cách linh hoạt hơn, đặc biệt là trong bối cảnh các đối tượng trên phân tán trong những tổ chức, những miền, với những chính sách và cơ chế xử lý token đặc thù (ví dụ như Public Key Infrastructure, Active Directory, Kerberos), khi đó chỉ cần trao đổi các policy và token phát sinh giữa các miền với nhau.

4.2.3 Kiến trúc bảo mật hướng dịch vụ SOSA

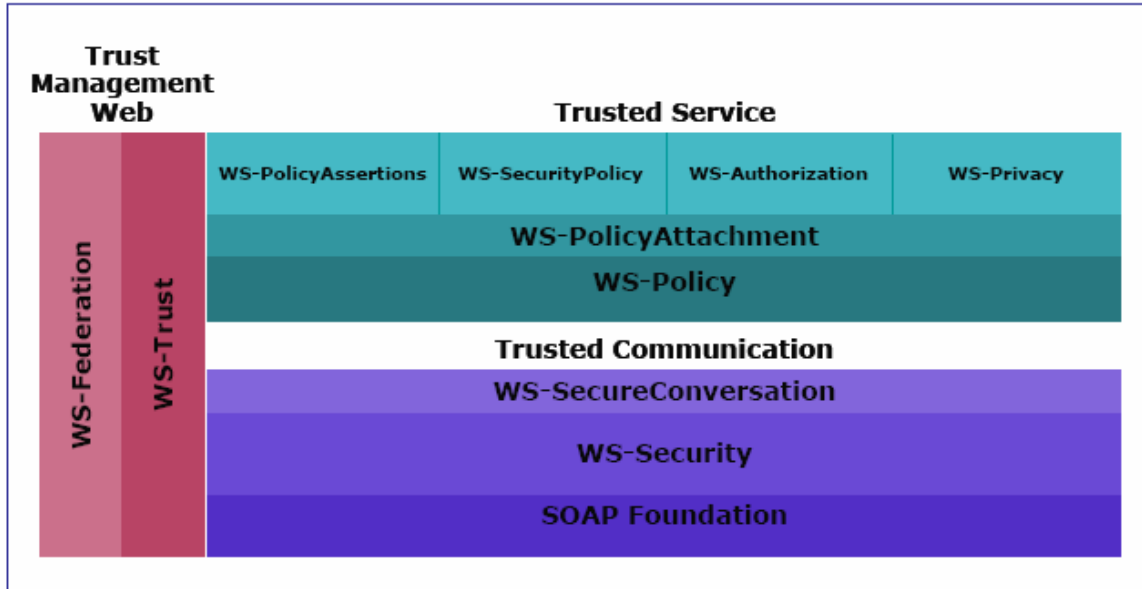
4.2.3.1 Các thành phần của kiến trúc



Hình 4-1 – Kiến trúc bảo mật hướng dịch vụ (SOSA)

Đối tượng sử dụng dịch vụ và cung cấp dịch vụ (ở phía bên trái của Hình 4-1) trao đổi các security token thông qua Standard-based Security Info Exchange Platform. Cơ sở hạ tầng bảo mật ở tầng dưới được cung cấp ra ngoài dưới dạng các web service (phần giữa của hình Hình 4-1). Các kiến trúc, công nghệ, giải pháp bảo mật hiện có được tái sử dụng lại thông qua tầng liên kết Integration backplane.

4.2.3.2 Cấu trúc phân tầng của Standard-based Security Info Exchange Platform



Hình 4-2 – Cấu trúc phân tầng của Standard-based Security Info Exchange Platform.

Tầng Trusted Communication

Tầng này được xây dựng dựa trên các đặc tả: SOAP Foundation, WS-Security và WS-SecureConversation. WS-Security là thành phần chính hỗ trợ để xây dựng một tầng liên kết bền vững, đảm bảo các luồng thông tin được trao đổi một cách an toàn.

- WS-Security được thiết kế một cách linh hoạt, được sử dụng như là nền tảng trong việc xây dựng nhiều mô hình bảo mật khác nhau, bao gồm Public Key Infrastructure, Kerberos và SSL. Đặc biệt, WS-Security cung cấp hỗ trợ cho nhiều dạng security tokens, nhiều vùng ủy thác (trust domain), nhiều định dạng chứng thực và nhiều thuật toán mã hóa.
- WS-Security chỉ định nghĩa thêm cấu trúc mở rộng cho phần đầu của một thông điệp dạng SOAP, nhằm mang các thông tin bảo mật (chữ ký điện tử, security token, mã hoá...) trong quá trình trao đổi thông điệp, với mục đích là phía nhận được thông điệp sẽ tin tưởng vào nội dung mình nhận được cũng như là đối tượng nào đã gửi thông điệp.

Với WS-SecureConversation, hai bên có thể tái sử dụng lại được ngữ cảnh bảo mật (*security context*) trong những lần trao đổi thông điệp (hai bên sẽ không cần phải thực hiện lại những thủ tục như trong lần trao đổi đầu tiên).

Tầng Trusted Service

Tầng này được thiết kế dựa trên các đặc tả: WS-PolicyAssertion, WS-SecurityPolicy, WS-Authorization, WS-Privacy, WS-PolicyAttachment, WS-Policy. Tầng này cho phép xây dựng cơ chế tạo ra những định nghĩa policy mở rộng và gắn kèm các thông tin này vào phần thông tin mô tả của web service.

- WS-Policy : định nghĩa các mô hình cơ bản của policy, policy statement, và cơ chế xác nhận policy.
- WS-PolicyAssertion: định nghĩa một vài cơ chế xác nhận policy tổng quát.
- WS-PolicyAttachment: định nghĩa cách gắn một policy vào một dịch vụ, hoặc là trực tiếp bằng cách nhúng vào trong thông tin WSDL, hay gián tiếp thông qua UDDI.
- WS-SecurityPolicy: định nghĩa các cơ chế xác nhận policy tương ứng cho các thuộc tính trong WS-Security, như là các yêu cầu về toàn vẹn thông điệp, yêu cầu về độ tin cậy của thông điệp, yêu cầu về loại security token...Đối tượng sử dụng dịch vụ phải lấy được các thông tin này để đáp ứng được các yêu cầu về bảo mật mà dịch vụ đưa ra.

Tầng Trusted Management Web

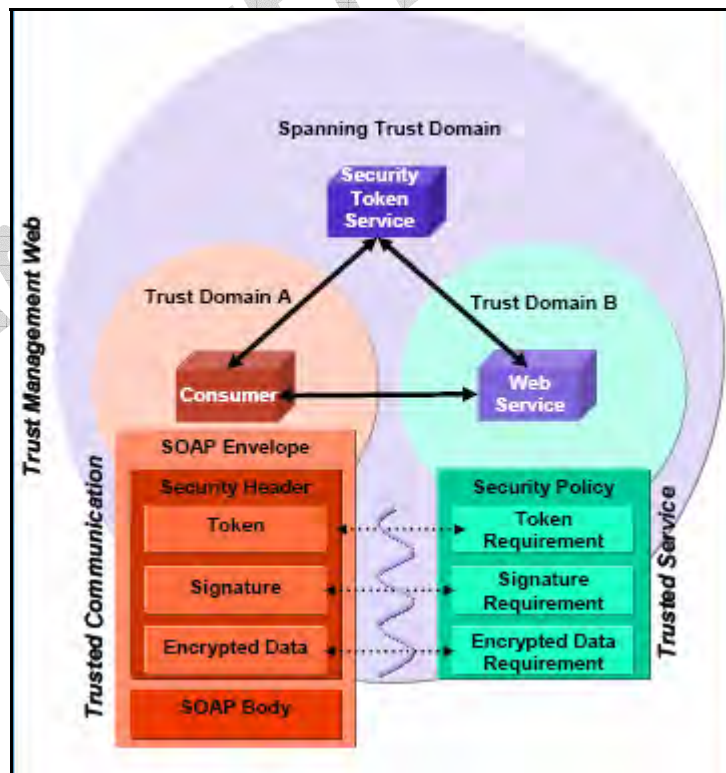
Tầng này được xây dựng dựa trên các đặc tả: WS-Federation, WS-Trust. Hai tầng Trusted Communication và Trusted Service đảm bảo cho hai đối tượng sử dụng và cung cấp dịch vụ tương tác trực tiếp với nhau trong một môi trường bảo mật và tin cậy. Hai tầng này làm được điều này dựa trên giả thiết rằng:

- Cả hai phía đều sử dụng cùng một công nghệ, kỹ thuật bảo mật.
- Cả hai phía đều tin tưởng vào cùng một domain (ví dụ: cùng tin tưởng vào cùng một tổ chức chức thực).

Điều này có nghĩa là: vấn đề sẽ phát sinh khi các bên tương tác thuộc những domain khác nhau, sử dụng những công nghệ mã hóa khác nhau. Và vai trò của tầng Trust Management Web là giải quyết vấn đề này.

➤ WS-Trust

- ▶ WS-Trust đưa ra một khái niệm gọi là **Security Token Service (STS)**. Nhiệm vụ của thành phần này là cấp phát, kiểm tra và chuyển đổi giữa các loại security token khác nhau (khi có yêu cầu.)



Hình 4-3 – Security Token Service.

- ▶ Về cơ bản, vai trò của STS rất giống với một tổ chức chứng thực PKI (PKI Certificate Authority). Tuy nhiên, nó vượt trội hơn với hai đặc điểm nổi bật:
 - Nó cấp phát tất cả các loại token (về quyền, vai trò..), chứ không chỉ là token dùng để định danh. STS không chỉ đơn giản là cấp phát và

chứng thực các token, mà nó còn có khả năng thực hiện chuyển đổi giữa các loại token với nhau (ví dụ như: X.509 certifate → Kerberos ticket). Đây chính là tính năng khiến cho nó có thể đảm trách vai trò làm trung gian trong qui trình tương tác giữa các đối tượng. Và điều đương nhiên rằng, để thực hiện được vai trò chuyển đổi như thế, một STS phải được tin cậy, uỷ thác để có khả năng cấp phát một số loại token nào đó, và bản thân nó cũng phải tin tưởng vào các token do các STS khác cấp phát.

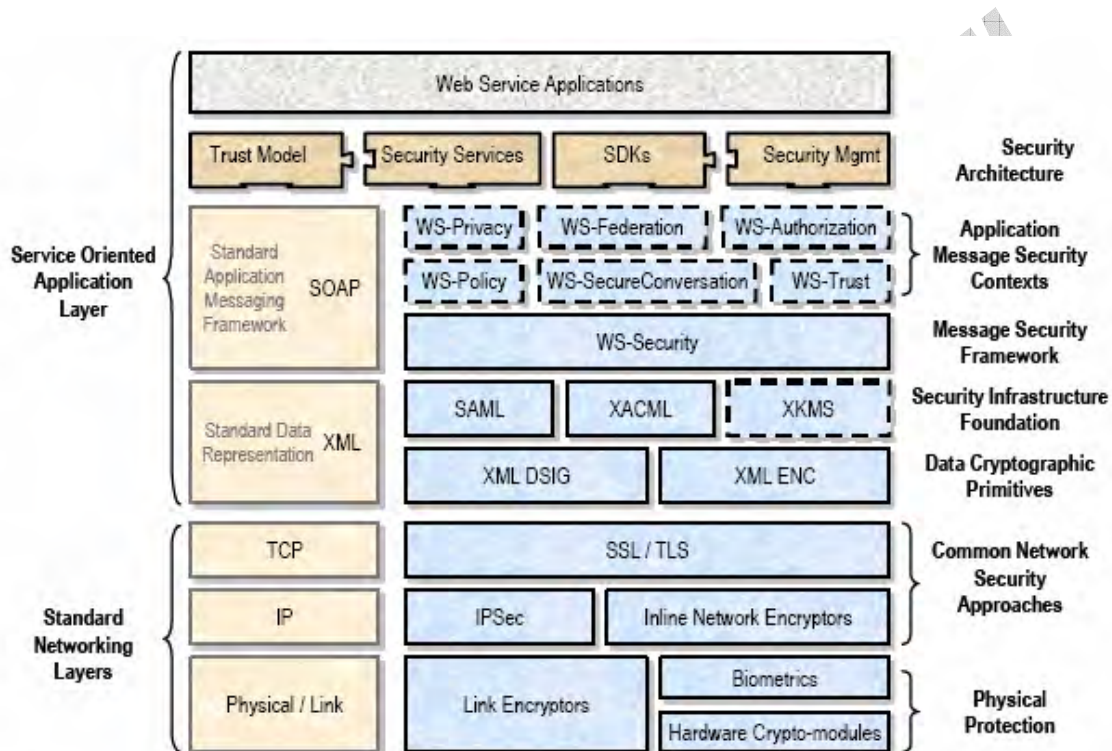
- ▶ Vì thế, vai trò của STS là kết nối các hệ thống bảo mật đơn lẻ để tạo thành một “mắc xích” liên kết thống nhất với nhau. Các hệ thống bảo mật đơn này vừa có thể duy trì những chính sách, cơ sở hạ tầng nội bộ của riêng mình, lại vừa có thể thiết lập được cách giao tiếp “đáng tin cậy” với các hệ thống khác trong cùng mắc xích.
- ▶ Các STS phải được quản lý sao cho “trong suốt” (transparent) với đối tượng trong quá trình tương tác. Ta gọi mạng các STS này là *Trusted Management Web*. Và cũng giống như các hệ thống mạng trong môi trường Internet đó là gồm tập hợp các mạng chức năng khác nhau (ví dụ, mạng các DNS-Domain Name Servers cung cấp dịch vụ phân giải tên miền, mạng các SNMP-Simple Network Management Protocol servers cung cấp các dịch vụ quản lý..) thì Trust Management Web có thể được cấu thành bởi nhiều mạng các STS khác nhau. Mỗi mạng STS cung cấp một dịch vụ tin cậy khác nhau, như là các máy chủ chứng thực sẽ cung cấp khả năng chức thực của một STS trong mạng các STS, hay các máy chủ quản lý về quyền sẽ có nhiệm vụ kiểm tra quyền của nhiều STS.

➤ **WS-Federation**

- ▶ Trong khi WS-Trust đã đưa ra một mô hình kiến trúc cơ bản của Trusted Management Web thì vẫn còn một vài vấn đề liên quan đến việc quản trị các thành phần bên trong kiến trúc mà chưa được WS-Trust đề cập đến:
 - Nên chọn một mô hình ủy thác nào cho các mắc xích STSs? Vấn đề này sẽ do WS-Federation giải quyết.
 - Việc quản lý chu kỳ hoạt động các thành phần của kiến trúc (STS, policy, cơ chế bảo mật ở tầng dưới..) sẽ được thực hiện như thế nào? WS-Management được xây dựng để xử lý vấn đề này.
 - Hệ thống cũng cần phải được trang bị một cơ chế cho phép quản lý thông tin trạng thái của các policy và token phân tán, cụ thể là cơ chế ghi nhớ lại một lượng thông tin nào đó. Từ đây, sẽ dẫn đến phát sinh một vấn đề mới đó là xử lý vấn đề đồng nhất về thông tin. DNS có thể được xem là một ví dụ cho mô hình này, trong đó các DNS server sẽ lưu lại kết quả phân giải, để khi xử lý yêu cầu tiếp theo, nếu yêu cầu cùng một kết quả thì nó sẽ đáp ứng ngay mà không cần phải truy vấn đến các root server.
 - Một vấn đề nữa đó là sự thiếu sót về một mô hình hỗ trợ cho quá trình lưu lại các ghi chú về thông tin trạng thái, kết quả xử lý...

4.3 Giới thiệu một số chuẩn về bảo mật trong XML

Phần này sẽ cung cấp một tầm nhìn tổng thể về các chuẩn bảo mật dành cho các thông điệp tựa XML hiện nay. Hình 4-4 minh họa về mối liên hệ cũng như là vị trí tương đối của các chuẩn này bên trong toàn bộ các tầng kỹ thuật của một hệ thống bảo mật. Điều cần làm rõ là bản thân mỗi chuẩn không được xây dựng để giải quyết mọi vấn đề về bảo mật. Một hệ thống bảo mật thật sự tốt phải được xây dựng dựa trên rất nhiều tầng khác nhau.



Hình 4-4 – Các tầng kỹ thuật bên dưới của một hệ thống bảo mật.

4.3.1 WS-Security

WS-Security là chuẩn được đề xuất dưới sự hợp tác của các hãng IBM, Microsoft và Verisign. Hiện chuẩn này đã được tổ chức OASIS thông qua và đang tiếp tục phát triển.

WS-Security là nền tảng để giải quyết các vấn đề bảo mật cho các thông điệp SOAP, với ba mục tiêu chính:

- Sử dụng các security token trong phần đầu của các thông điệp SOAP để hỗ trợ cho việc định danh và chứng thực.
- Sử dụng chuẩn XML-Signature đảm bảo tính toàn vẹn và xác thực của dữ liệu.
- Sử dụng chuẩn XML-Encryption đảm bảo độ tin cậy cho dữ liệu.

Vẫn còn khá nhiều vấn đề hiện vẫn chưa được giải quyết triệt để bởi WS-Security. WS-Security chỉ là một trong loạt những chuẩn đầu tiên được công bố, mà tương lai hướng đến sẽ là đưa ra được một nền tảng bảo mật rộng hơn cho Web service.

Một số chuẩn khác cũng đã và đang được nghiên cứu bởi nhiều hãng và tổ chức nhằm giải quyết những vấn đề còn tồn đọng như là: vấn đề về phân quyền, chính sách, sự tin cậy, an toàn trong tương tác và khả năng liên kết.

4.3.2 XML-Signature

Chuẩn này đã được chứng thực bởi tổ chức W3C. Chuẩn này quan tâm đến cú pháp và cách xử lý trong việc chứng thực một thành phần nào đó trong tài liệu XML bằng các công nghệ chứng thực khóa đồng bộ hay bất đồng bộ.

4.3.3 XML-Encryption

Chuẩn này cũng được đưa ra bởi W3C, và được xây dựng để đưa ra các qui tắc cú pháp cũng như là luật xử lý trong việc mã hóa và giải mã các thành phần trong một tài liệu XML. Mục tiêu đặt ra là bảo mật dữ liệu.

4.3.4 XML Key Management Specification:

Đặc tả này gồm hai phần:

- X-KRSS (XML Key Registration Service Specification)
 - ▶ Giải quyết cho vấn đề đăng ký và thu hồi các khóa ngoại
- X-KISS (XML Key Information Service Specification)
 - ▶ Giải quyết vấn đề định vị và chứng thực các khóa

4.3.5 Security Assertion Markup Language (SAML)

Chuẩn này được đưa ra bởi tổ chức OASIS (Organization for the Advancement of Structured Information Standard). SAML định nghĩa một nền tảng cho việc trao đổi các thông tin bảo mật dưới định dạng XML. Những thông tin bảo mật này có thể là: các thông tin về chứng thực, các quyết định về phân quyền, hay có thể là những thuộc tính của các đối tượng được biểu diễn dưới dạng XML và được cấp phát bởi các nơi cung cấp chứng thực SAML. Đặc tả SAML cũng định nghĩa các giao thức, những qui tắc trong quá trình vận chuyển các thông tin bảo mật.

4.4 Khai thác tính năng bảo mật web service của bộ thư viện WSE (Web Services Enhancements)

Web Service Enhancements 2.0 là bộ thư viện lập trình trên nền .NET, hỗ trợ trong việc xây dựng các web service theo những chuẩn mới nhất như WS-Security, WS-SecureConversation, WS-Trust, WS-Policy, WS-SecurityPolicy, WS-Addressing, WS-Messaging và WS-Attachments. Với bộ thư viện này, ta có thể đưa các tính năng liên quan đến bảo mật này vào web service trong lúc thiết kế bằng cách sử dụng mã lệnh, hay vào thời điểm triển khai thông qua việc sử dụng các tập tin policy.

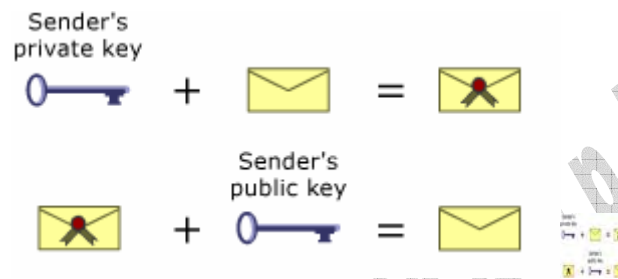
4.4.1 Những tính năng chính của bộ thư viện WSE

4.4.1.1 Bảo mật web service

WSE sử dụng các cơ chế được định nghĩa trong WS-Security để đặt các ủy quyền chứng thực (security credential) như security token vào trong các thông điệp SOAP. WSE sau đó sẽ thực hiện kiểm tra tính hợp lệ của những security credentials trước khi chuyển quyền thực thi cho các web service. WSE 2.0 hỗ trợ các loại security token sau: tên/mật khẩu, X.509 Certificate, Kerberos ticket, Security Context token và các loại security token do người dùng định nghĩa.

WSE còn cho phép các nhà phát triển xây dựng riêng cho mình các dịch vụ security token. Các dịch vụ này có thể tạo ra các loại security token khác mà có thể dùng trong quá trình tương tác với các web service nào tin tưởng vào dịch vụ này. Thông qua việc hỗ trợ xác nhận số và/hay mã hóa các thông điệp SOAP sẽ tăng cường khả năng an toàn cho các web service.

- Xác nhận số một thông điệp SOAP sẽ giúp cho đối tượng nhận thông điệp kiểm tra được thông điệp đó có bị thay đổi hay không?



Hình 4-5 – Xác nhận số một thông điệp.

- Mã hóa một thông điệp SOAP sẽ đảm bảo cho chỉ những web service mong muốn mới có thể đọc được nội dung của thông điệp đó.



Hình 4-6 – Mã hóa một thông điệp

4.4.1.2 Hỗ trợ Policy

WSE hỗ trợ các nhà phát triển đưa ra các yêu cầu về quá trình gửi và nhận thông điệp bằng cách dùng các tập tin cấu hình. Trước đây, một đối tượng khi nhận được một thông điệp SOAP phải dùng mã lệnh để kiểm tra các thông tin về thông điệp nhận được như là có được xác nhận số hay mã hóa chưa? Và cũng tương tự như thế, phía gửi cũng phải viết mã lệnh để lấy được các yêu cầu này từ phía nhà cung cấp. Nay thì các yêu cầu này có thể được cung cấp thông qua các tập tin cấu hình.

Khi các cơ chế xác nhận policy được chỉ định:

- Các thông điệp SOAP khi được gửi đi sẽ qua quá trình kiểm tra để đảm bảo chúng thỏa mãn các policy assertion của phía gửi. Nếu không thỏa, WSE sẽ ném ra một biệt lệ.
- Các thông điệp SOAP trước khi được nhận vào sẽ phải được kiểm tra xem có đáp ứng được các policy assertion của phía nhận hay không? Nếu không, thông điệp đó sẽ được gửi trả về hay một biệt lệ sẽ được ném ra.

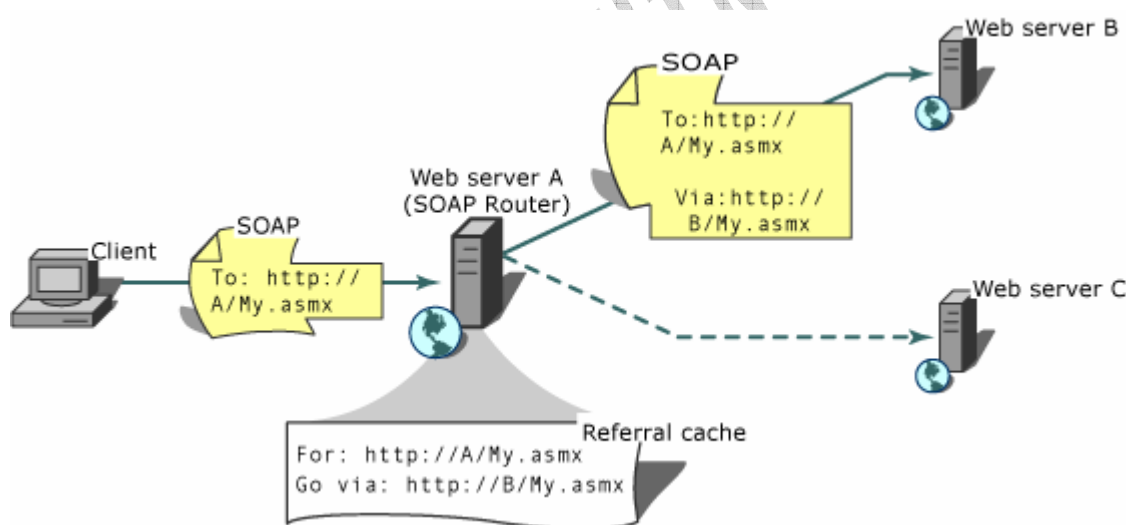
WSE đã hỗ trợ sẵn một vài cơ chế xác nhận policy (ví dụ như: yêu cầu phần body của thông điệp phải được xác nhận-signed bởi một X.509 certificate). Ngoài ra, hệ thống policy còn cho phép thêm mới những cơ chế xác nhận policy khác do người dùng định nghĩa.

4.4.1.3 SOAP Messaging

Đây là một tính năng nổi trội của WSE. SOAP Messaging hỗ trợ nhiều nghi thức ở tầng vận chuyển như HTTP, TCP, với giao tiếp bất đồng bộ hay đồng bộ. Đặc biệt, khi thực hiện việc gửi và nhận các thông điệp theo nghi thức TCP thì ta không cần phải có một Web server.

4.4.1.4 Điều phối các thông điệp SOAP

Ta có thể sử dụng WSE để xây dựng các ứng dụng phân tán mà kiến trúc phân tán của nó là trong suốt đối với người dùng. Ta sử dụng một máy tính trung gian và cấu hình nó chạy WSE router. Người dùng sẽ gửi yêu cầu đến WSE router thay vì trực tiếp đến web service. WSE router sau đó sẽ chuyển thông điệp SOAP đến máy đang chạy web service dựa trên thông tin cấu hình của router. Giải pháp này giúp hệ thống ta linh hoạt hơn, bền vững hơn, vì ta có thể thay đổi thông tin về các máy đích khi có sự cố xảy ra.



Hình 4-7 – Điều phối thông điệp SOAP.

4.4.1.5 Gửi những đối tượng kèm theo các thông điệp dạng SOAP

WSE hỗ trợ nghi thức DIME (Direct Internet Message Encapsulation). Nghi thức này định nghĩa cơ chế để đính kèm những đối tượng khác trong các thông điệp SOAP. Điều này cần thiết cho những web service nào có nhu cầu muốn gửi những thông tin có kích thước lớn (dạng text hay binary) như tập tin dạng ảnh hay âm thanh.

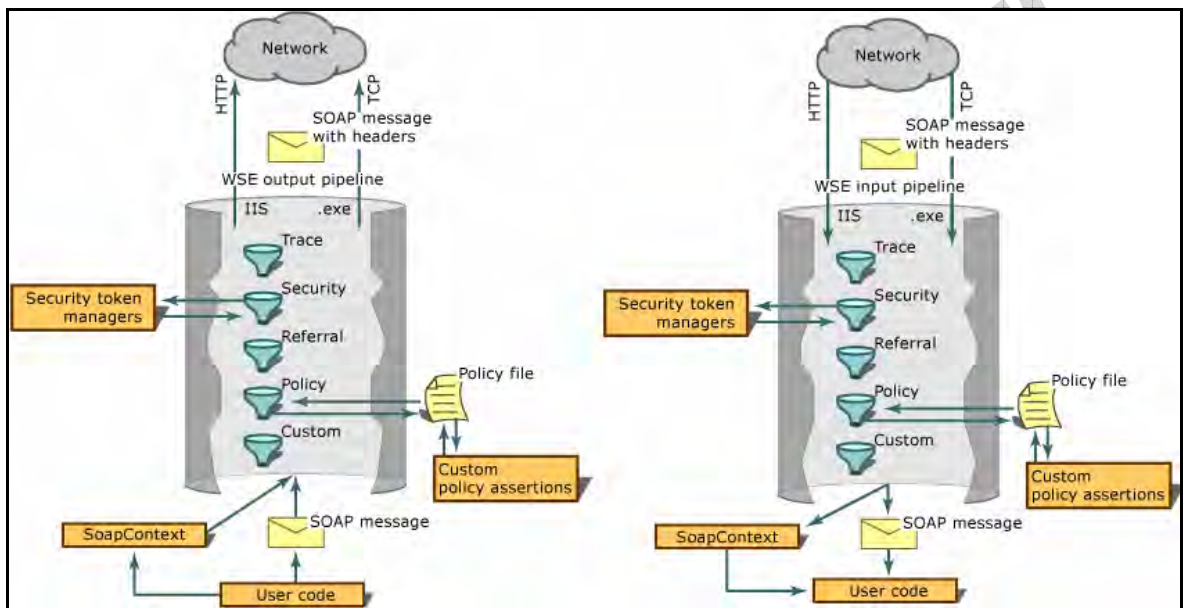
Theo mặc định thì các thông điệp SOAP không thích hợp để gửi đính kèm các tập tin lớn. Vì định dạng của một thông điệp SOAP là theo XML, khi thêm một tập tin vào thông điệp SOAP thì đòi hỏi tập tin đó phải được chuyển đổi thành dạng XML. Điều này có thể làm tăng kích thước của tập tin lên đáng kể so với kích thước thật của nó. Nghi thức DIME giải quyết vấn đề này bằng cách định nghĩa một cơ chế để đặt toàn bộ nội dung tập tin gốc nằm ở bên ngoài thông điệp SOAP, như vậy sẽ loại bỏ được việc phải chuyển đổi nội dung tập tin sang dạng XML.

4.4.2 Kiến trúc của WSE

Phần lõi bên trong của WSE là một bộ xử lý chính được xây dựng để đưa các tính năng cao cấp của web service vào các thông điệp SOAP. Điều này đòi hỏi thao tác ghi thông tin vào phần đầu của các thông điệp SOAP mà sẽ được gửi đi và đọc thông tin phần đầu của các thông điệp SOAP được gửi đến. Bên cạnh đó còn có các thao tác biến đổi trên phần thân của thông điệp SOAP như là mã hóa nội dung trên thông điệp gửi, và giải mã trên thông điệp nhận.

Các thao tác trên được thực hiện bởi hai bộ lọc, một cho các thông điệp gửi và một cho các thông điệp nhận.

- Mọi thông điệp khi được gửi đi (thông điệp yêu cầu từ phía máy khách hay phản hồi từ phía máy chủ) đều phải qua các bộ lọc dành cho các thông điệp gửi.
- Mọi thông điệp khi được nhận vào (thông điệp đến máy chủ hay phản hồi về máy khách) đều phải qua các bộ lọc dành cho thông điệp nhận.



Hình 4-8 – Cơ chế sử dụng bộ lọc của WSE

✍ Một trong các lợi ích mà kiến trúc hướng dịch vụ đem lại đó chính là tính liên kết cao và khả năng dễ mở rộng của hệ thống. Và ứng dụng SOA để giải quyết vấn đề tích hợp hệ thống đang được rất nhiều sự quan tâm của các nhà quản lý hệ thống. Thế thì, SOA giải quyết vấn đề tích hợp như thế nào?

Chương 5

SOA VÀ VẤN ĐỀ TÍCH HỢP

Chương 5 sẽ trình bày và phân tích về nhu cầu và một số khó khăn gây trở ngại trong vấn đề tích hợp hệ thống. Qua đó xem xét một số giải pháp được sử dụng trong tích hợp, bao gồm giải pháp sử dụng các sản phẩm middleware và giải pháp ứng dụng SOA và Web services: Web Service Integration (WSI) và Service-Oriented Integration (SOI). Sau đó, xem xét cụ thể giải pháp ứng dụng SOA và Web services trong tích hợp các hệ thống xây dựng trên nền .NET và J2EE và trong tái sử dụng lại các hệ thống cũ.

5.1 Giới thiệu về Enterprise Application Integration

5.1.1 Hiện trạng

Hệ thống của các tổ chức doanh nghiệp ngày càng trở nên cồng kềnh hơn với việc triển khai hàng loạt các ứng dụng phân tán dựa trên nhiều hệ nền, kỹ thuật, công nghệ khác nhau. Tình trạng chưa định hình được hoàn chỉnh các chuẩn hỗ trợ trong vấn đề giao tiếp giữa các hệ thống khiến cho việc tương tác, trao đổi thông tin giữa các hệ thống khác nhau thực sự trở thành một vấn đề nan giải. Khó khăn này không chỉ là đối với các nhà quản trị hệ thống, mà còn cả đối với các hãng, công ty đã cung cấp các kỹ thuật, công nghệ, giải pháp... Vì thế, việc xây dựng, triển khai các hệ thống phân tán hiện nay đòi hỏi ngày càng cao về chi phí cũng như độ phức tạp.

Ngày nay, nhiều công ty, hãng phát triển, các tổ chức đã bắt tay nhau để xây dựng các chuẩn chung. Bên cạnh đó là sự ra đời của 2 hệ nền .NET và J2EE được thiết kế với nhiều ưu điểm nổi trội về tính linh hoạt, tương thích, dễ mở rộng ... và khả năng liên kết cao đã góp phần giải quyết được các vấn đề khó khăn mà các công nghệ trước chưa làm được. Nhưng vấn đề ở đây là ta không thể thực hiện việc “gỡ bỏ và thay mới” các hệ thống, các ứng dụng và chức năng hiện có. Vì chi phí bỏ ra để xây dựng

lại từ đầu các hệ thống là rất cao. Ngoài vấn đề chi phí, còn có một nguyên nhân khác nữa là: các hệ thống hiện hành đã được chứng thực về tính đúng đắn, hiệu quả trong suốt quá trình vận hành, trong khi hệ thống mới chưa được đảm bảo về các vấn đề này.

Vấn đề đặt ra lúc này đó là làm sao gắn kết những công nghệ mới cho hệ thống trong khi vẫn giữ lại được những gì hiện có. Chúng ta đang nói đến khái niệm tích hợp hệ thống.

5.1.2 Một số lý do khiến các tổ chức doanh nghiệp phải quan tâm đến vấn đề tích hợp (xét về mặt nghiệp vụ)

➤ Thay đổi cơ cấu tổ chức tổng thể

- Đây là tình trạng khi các tổ chức sáp nhập với một tổ chức khác, hay khi bán đi một chi nhánh của tổ chức. Khi đó các hệ thống tin học nào có những qui trình xử lý trùng lặp thì cần phải được kết hợp lại với nhau nhằm tăng hiệu quả hoạt động. Một ví dụ đó là khi các ngân hàng sáp nhập lại, thì các hệ thống phục vụ khách hàng cần phải được hỗ trợ để có thể quản lý được các tài khoản mà trước đó trực thuộc các ngân hàng khác nhau.

➤ Thay đổi cơ cấu tổ chức nội bộ

- Mặc dù đây chỉ là những thay đổi diễn ra trong phạm vi nội bộ một tổ chức, nhưng cũng sẽ đưa đến những kết quả tương tự như ở trường hợp đầu. Và thông thường những hoạt động này còn diễn ra với mật độ thường xuyên hơn.

➤ Sáp nhập các hệ thống, ứng dụng:

- Nhiều hệ thống tin học của các tổ chức thường được sáp nhập lại hay thay thế hẳn để tiết kiệm chi phí quản trị các hệ thống này, cũng như là để tăng hiệu quả hoạt động của các xử lý nghiệp vụ. Ví dụ như một công ty truyền thông, có nhiều hệ thống thanh toán tương ứng với các cơ sở hạ tầng về mạng: mạng không dây, mạng có dây, mạng băng thông rộng... thì có thể tiết kiệm một lượng đáng kể thời gian và tiền bạc khi kết hợp các hệ thống này lại.

➤ Tình trạng không đồng nhất, trùng lặp, phân mảnh dữ liệu

- Các dữ liệu quan trọng thường được phân bố ở nhiều hệ thống. Và khi có nhu cầu, thì các dữ liệu này phải được chia sẻ, kết hợp và tinh chế để hỗ trợ cho các hoạt động phân tích, thống kê,... và đưa ra quyết định.
- **Thay đổi các chiến lược kinh doanh**
 - Các công ty thường có những thay đổi trong đường lối, chiến lược kinh doanh. Điều này khiến cho một số yếu tố môi trường khác cũng phải thay đổi theo để thích nghi. Một phần trong các yếu tố này là các hệ thống tin học.
- **Thay đổi để thích nghi với qui định chung**
 - Trong lãnh vực kinh doanh thì các qui định, ràng buộc là điều không thể thiếu. Và các qui trình nghiệp vụ đều gắn chặt với các qui định này. Một khi các qui định này có sự thay đổi thì các qui trình nghiệp vụ liên quan cũng đòi hỏi phải được thay đổi theo để thích ứng. Khi đó các hệ thống tin học của doanh nghiệp cũng phải kịp thời thay đổi, điều chỉnh để hỗ trợ những qui trình mới.
- **Tối ưu hóa các qui trình nghiệp vụ**
 - Những qui trình xử lý nào mà đòi hỏi phải nhập dữ liệu một cách thủ công thì cần phải được thay thế bằng các hệ thống mới hỗ trợ xử lý tự động các luồng giao tác. Ví dụ: một công ty thương mại với qui trình hoạt động lúc trước như sau: nhận đơn đặt hàng từ Internet, sau đó phải thực hiện các thao tác thủ công để nhập các thông tin này vào hai hệ thống quản lý đơn đặt hàng và quản lý sản xuất hàng. Ta cần đưa ra một giải pháp để tích hợp hai hệ thống trên vào hệ thống của ta, như thế thì quá trình chuyển dữ liệu sẽ được thực hiện một cách tự động.

5.1.3 Các vấn đề kỹ thuật gặp phải trong tích hợp hệ thống

- Các xung đột giữa các qui trình xử lý trong các hệ thống.
- Sự khác biệt về cấu trúc cũng như là ngữ nghĩa của dữ liệu được dùng trong các hệ thống.
- Sự không tương thích giữa các chuẩn, kỹ thuật, công nghệ sử dụng trong các hệ thống.

5.1.4 Các yêu cầu cho một giải pháp tích hợp

- Chi phí triển khai không quá cao.
- Dễ nắm bắt và quản lý
- Không làm ảnh hưởng đến các hệ thống không liên quan.
- Đáp ứng tốt các yêu cầu về tính dễ mở rộng, ổn định, hiệu quả, khả năng chịu lỗi, bảo mật...
- Linh hoạt và dễ tùy biến để có thể dễ dàng thích nghi với những yêu cầu của từng dự án khác nhau.

5.1.5 Việc tích hợp có thể được áp dụng ở nhiều tầng khác nhau

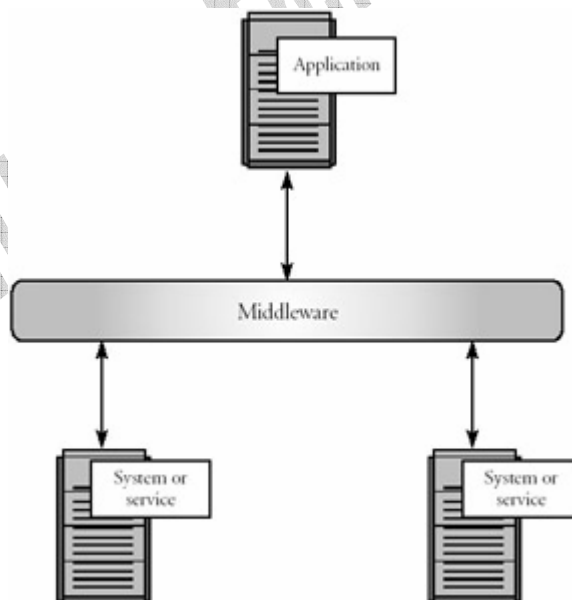
- Tích hợp dữ liệu
 - ▶ Quan tâm đến vấn đề tích hợp ở tầng dữ liệu, thường là thực hiện đồng bộ hóa dữ liệu ở các nguồn khác nhau (database, datamarts, data warehouses). Vấn đề khó khăn chính đó là phải dung hòa được lược đồ cơ sở dữ liệu của các database cũng như là ngữ nghĩa của các thành phần dữ liệu.
- Tích hợp thông điệp
 - ▶ Giải quyết vấn đề tích hợp bằng cách định nghĩa và xây dựng cơ chế trao đổi thông điệp giữa các hệ thống. Vấn đề khó khăn chính đó là sự chuyển đổi giữa dữ liệu ứng dụng và thông điệp. Ngoài ra còn phải thực hiện chuyển đổi giữa các định dạng thông điệp sao cho mọi hệ thống đều hiểu.
- Tích hợp thành tố
 - ▶ Xử lý bao bọc các hệ thống cũ bằng công nghệ hướng thành tố (CORBA, .NET, J2EE) và liên kết các thành tố này lại với nhau thông qua các thành phần giao tiếp. Khó khăn gặp phải đó là phải thực hiện liên kết các thành tố được xây dựng trên những công nghệ khác nhau (CORBA và .NET, J2EE và .NET)

- Tích hợp ứng dụng
 - ▶ Thực hiện tích hợp các ứng dụng bằng cách sử dụng tập các hàm APIs, mô hình đối tượng, định dạng thông điệp, lược đồ cơ sở dữ liệu hay bất cứ kỹ thuật nào mà lập trình viên có thể tiếp cận. Khó khăn gặp phải đó là sự khác nhau về mô hình dữ liệu giữa các ứng dụng. Mô hình tích hợp này thường được dùng cho các ứng dụng đóng gói.
- Tích hợp dịch vụ
 - ▶ Mô hình tích hợp này tạo ra các dịch vụ nghiệp vụ trừu tượng, không gắn chặt vào một cơ sở dữ liệu, mô hình thành tố hay ứng dụng đóng gói nào. Và sẽ sử dụng các dịch vụ này như những đơn vị cơ sở trong việc tích hợp hệ thống. Khó khăn của mô hình này đó là thường phải đòi hỏi phải xây dựng một kiến trúc tích hợp hoàn chỉnh như là SOA để có thể thực hiện tách biệt giữa thành phần giao tiếp và thành phần thực thi của service.
- Tích hợp tiến trình
 - ▶ Giải pháp đó là tạo ra các tiến trình nghiệp vụ bằng cách tích hợp các tài nguyên có sẵn (dữ liệu, thành tố, ứng dụng và dịch vụ). Và sau đó là tập trung vào việc quản lý các tiến trình nghiệp vụ một cách độc lập với một ứng dụng riêng biệt nào đó. Vấn đề gặp phải đó là đạt được sự “thỏa thuận” giữa các tổ chức về việc định nghĩa các tiến trình nghiệp vụ và một kiến trúc tích hợp hoàn chỉnh nhằm hỗ trợ việc tích hợp những tài nguyên của các hệ thống được thực hiện một cách dễ dàng.
- Tích hợp thành phần giao tiếp người dùng
 - ▶ Giải pháp này thường có hai cách thực hiện khác nhau:
 - Sử dụng hệ giao tiếp người dùng của các hệ thống cũ như là các thành phần giao tiếp để truy cập dữ liệu từ các ứng dụng hay các giao tác đang thực hiện. Cách này rất không bền vững, đặc biệt là khi hệ giao tiếp người dùng của ứng dụng cần lấy dữ liệu bị thay đổi
 - Việc tích hợp được thực hiện ở tầng thể hiện như là desktop hay portal.

5.2 Phân tích một số kỹ thuật tích hợp sử dụng Middleware

5.2.1 Khái niệm middleware

Middleware được sử dụng trong rất nhiều ngữ cảnh, và trong mỗi ngữ cảnh như thế lại có một ý nghĩa khác nhau. Trong ngữ cảnh của chúng ta ở đây, integration, thì middleware là một phần mềm hỗ trợ trong việc tạo ra môi trường trao đổi dữ liệu giữa các hệ thống. Middleware che dấu đi sự phức tạp trong giao tiếp của các hệ thống hay dịch vụ, làm đơn giản hóa sự phát triển những hệ thống, dịch vụ này. Hình 5-1 thể hiện rõ vai trò cơ bản của middleware trong kiến trúc của một hệ thống tích hợp.



Hình 5-1 – Vai trò cơ bản của middleware.

5.2.2 Các sản phẩm Middleware sử dụng trong tích hợp hệ thống

5.2.2.1 Adapter

Mỗi nguồn dữ liệu trong một hệ thống (có thể là một database hay một ứng dụng) được truy cập thông qua một thành phần gọi là adapter. Một vài hệ thống không được xây dựng sẵn cho mình những thành phần adapter như thế. Vì vậy, khi những hệ thống được tích hợp vào một hệ thống khác thì đòi hỏi phải thiết kế một adapter cho nó.

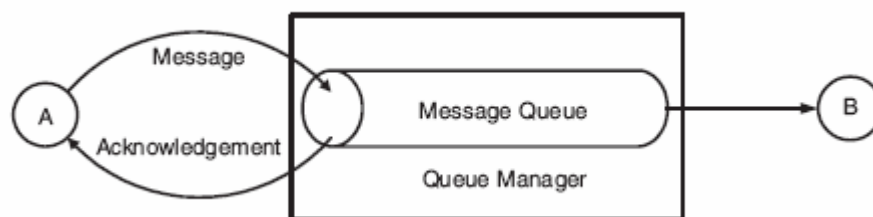
Hầu hết các hệ thống hay hệ quản trị cơ sở dữ liệu đóng gói ngày nay đều đi kèm với những bộ adapter cho nó. Những bộ adapter này có thể được viết bởi chính hãng cung cấp gói sản phẩm đó, và cũng có thể là từ một hãng khác.

5.2.2.2 Message Oriented Middleware (MOM)

MOM là phần mềm hỗ trợ trao đổi dữ liệu giữa hệ thống dưới hình thức những gói tin rời rạc gọi là thông điệp. Cơ chế thông điệp đã xây dựng và sử dụng từ những năm 1970, và là một trong những cơ chế đầu tiên được dùng trong quá trình giao tiếp giữa các hệ thống phân tán. Nhưng khi đó, các cơ chế này được thiết kế “cứng” để thỏa mãn một yêu cầu nào đó. Còn các cơ chế thông điệp bây giờ được xây dựng dựa trên các chuẩn chung.

Hầu hết các sản phẩm MOM đều cung cấp rất nhiều tính năng, bao gồm: truyền nhận thông điệp thông qua hàng đợi, đảm bảo an toàn cho dữ liệu truyền, hỗ trợ xử lý đồng bộ và bất đồng bộ và cho phép gửi một lúc đến nhiều nơi thông qua cơ chế publish/subscribe.

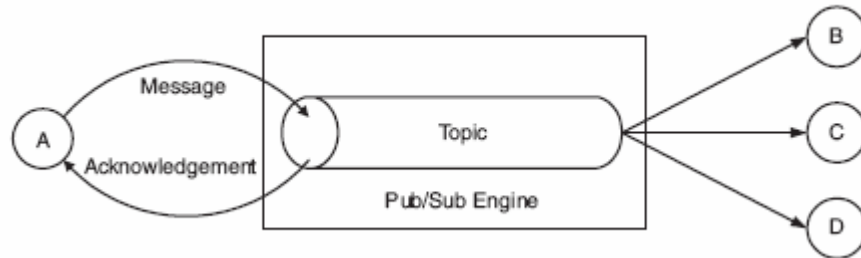
- Cơ chế hàng đợi thông điệp:
 - Các sản phẩm “hàng đợi thông điệp” cho phép gửi một thông điệp từ ứng dụng này đến ứng dụng khác thông qua hàng đợi. Một thành phần được gọi là “quản lý hàng đợi” sẽ quản lý chuyển nhận thông điệp vào và gửi thông tin xác nhận cho đối tượng đã gửi. Hầu hết các thành phần quản lý hàng đợi đều cung cấp một dịch vụ “chuyển dữ liệu an toàn” để đảm bảo dữ liệu đã được nhận đầy đủ trước khi gửi xác nhận cho đối tượng gửi.



Hình 5-2 – Cơ chế hàng đợi

- Cơ chế Publish/subscribe:
 - Cơ chế giao tiếp publish/subscribe tách rời mối liên kết giữa phía gửi và phía nhận. Đối tượng gửi thay vì gửi trực tiếp thông tin đến đối tượng nhận sẽ gửi thông điệp đến một thành phần gọi là pub/sub. Thành phần này sẽ

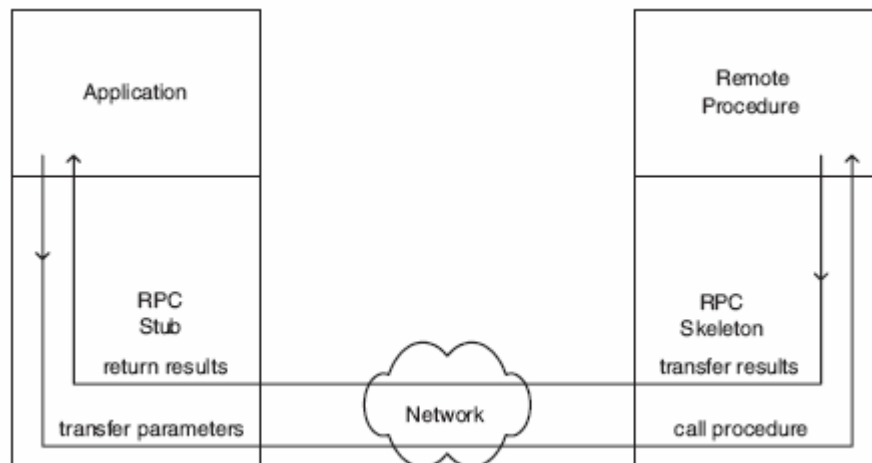
nhận dạng tiêu đề của thông điệp và chuyển đến cho những đối tượng nào đã đăng ký nhận loại thông điệp này.



Hình 5-3 – Cơ chế Publish/Subscribe

5.2.2.3 Remote Procedure Call (RPC)

RPC được xây dựng nhằm hỗ trợ gọi thực thi các dịch vụ từ xa một cách đơn giản giống như gọi một thủ tục cục bộ bằng cách giấu đi cơ chế phức tạp của mạng. RPC về cơ bản sẽ hoạt động theo cơ chế đồng bộ, nghĩa là đối tượng gọi sẽ bị tình trạng blocked cho tới khi kết quả được trả về. Nhưng cũng có thể hỗ trợ cơ chế xử lý bất đồng bộ cho RPC thông qua kỹ thuật đa luồng.

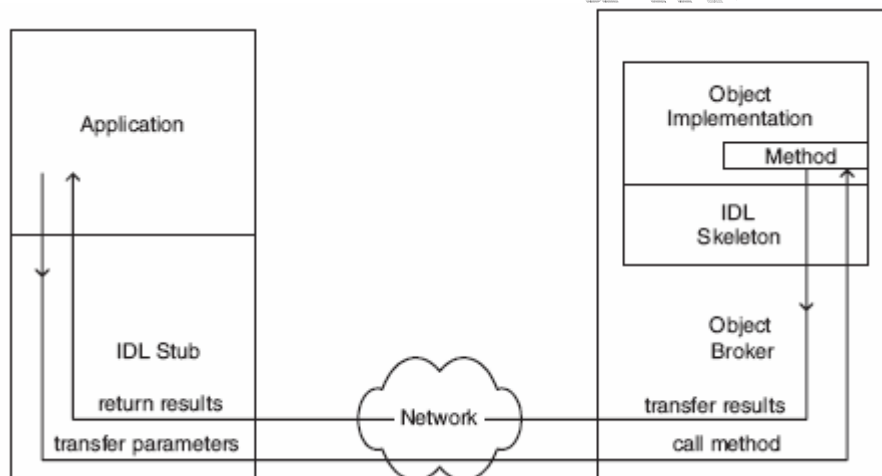


Hình 5-4 - Remote Procedure Call

Một đặc trưng quan trọng của RPC đó là cách thiết kế thành phần giao tiếp độc lập với phần thực thi.

5.2.2.4 Distributed Object Technology (DOT)

Đây là phiên bản hướng đối tượng của RPC. Một ứng dụng có thể sử dụng DOT để định nghĩa một tập các đối tượng có thể gọi thực thi thông qua môi trường mạng.



Hình 5-5 – Distributed Object Model

Object Broker sẽ cung cấp môi trường để quản lý, giám sát quá trình giao tiếp cũng như là chu kỳ sống của tất cả các đối tượng. DOT dùng một tập tin giao tiếp gọi là IDL file (Interface Description Language). IDL là một ngôn ngữ cấp cao dùng để mô tả thông tin về thuộc tính và phương thức của những đối tượng.

- **Common Object Request Broker Architecture (CORBA)**
 - ▶ Đây là một tập các đặc tả được phát triển bởi Object Management Group (OMG). Phần thực thi của CORBA còn được gọi là ORBs (Object Request Brokers). CORBA được thiết kế để giải quyết các yêu cầu về tính toán, hỗ trợ tính dễ mở rộng và khả năng chịu lỗi. Một ưu điểm của CORBA đó là nó hoàn toàn không phụ thuộc vào hãng phát triển, cho phép sử dụng nhiều ngôn ngữ lập trình để tạo các đối tượng CORBA.
- **Enterprise JavaBeans (EJB)**
 - ▶ EJB là kỹ thuật được phát triển bởi Sun Microsystem nhằm đáp ứng các yêu cầu của các nhà phát triển Java trong việc xây dựng các ứng dụng phân tán. EJB được phát triển trên nền J2EE, với mục tiêu là xây dựng các ứng dụng có khả năng dễ mở rộng, hỗ trợ quản lý giao tác và đáp ứng yêu cầu về bảo mật.

- Microsoft Distributed Component Object Model (DCOM)
 - ▶ DCOM là một tập các khái niệm, giao diện lập trình do Microsoft đưa ra vào năm 1995, nhằm hỗ trợ việc giao tiếp với các đối tượng COM thông qua môi trường mạng. DCOM sử dụng cơ chế RPC để thực hiện trừu tượng hóa quá trình gửi và nhận thông tin giữa các đối tượng COM.

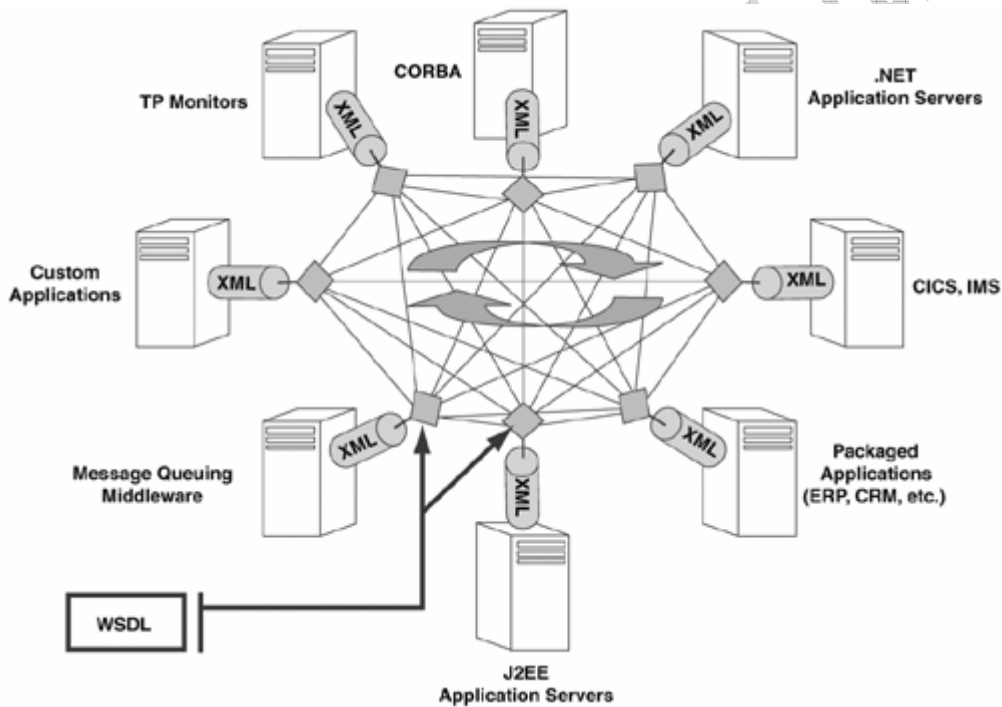
5.3 SOA và web service giải quyết vấn đề tích hợp như thế nào

5.3.1 Công nghệ XML và web service

Những nghiên cứu gần đây cho thấy rằng: các khó khăn trong việc tích hợp các hệ thống có thể được khắc phục bằng cách định nghĩa thêm một tầng trừu tượng cho các hệ thống tin học hiện có và mới xây dựng. Tầng này sẽ được xây dựng dựa trên các chuẩn của web service. Một số giải pháp chung trong việc dùng web service cho vấn đề tích hợp:

- Tích hợp hướng dữ liệu:
 - ▶ Xác định thông tin dữ liệu nào cần được chia sẻ (các bảng dữ liệu, các định dạng file và thông điệp)
 - ▶ Xây dựng các lược đồ mô tả XML (Xml Schema) cho các thông tin này.
 - ▶ Sử dụng SOAP như là định dạng của thông điệp.
- Tích hợp hướng chức năng/hàm APIs:
 - ▶ Xác định các phương thức từ xa nào sẽ được thể hiện ra ngoài như các web service.
 - ▶ Định nghĩa kiểu dữ liệu XML cho đối số của các phương thức này.
 - ▶ Sử dụng SOAP như là định dạng thông điệp.
- Tích hợp hướng thành phần giao tiếp:
 - ▶ Định nghĩa thông tin mô tả web service (WSDL).

- Tạo ra các đối tượng bọc và thực hiện ánh xạ tương ứng giữa thành phần giao tiếp vừa định nghĩa với dữ liệu, thông điệp và các lời gọi hàm APIs (cần được chia sẻ) của hệ thống hiện hành.



Hình 5-6 – Sử dụng web service trong vấn đề tích hợp.

Với sự hỗ trợ kỹ thuật của web service, dữ liệu trao đổi giữa các hệ thống được tự động chuyển đổi sang dạng chuẩn (XML) mà mọi hệ thống đều hiểu được. Tại mỗi hệ thống sẽ chịu trách nhiệm xử lý chuyển đổi dữ liệu từ dạng chuẩn thành những kiểu đặc thù của nó trong quá trình nhận thông điệp và thực hiện xử lý ngược lại (chuyển dữ liệu từ định dạng riêng sang dạng chuẩn chung-XML) trong quá trình gửi thông điệp.

Tuy nhiên, các quá trình chuyển đổi này không thật sự lúc nào cũng cần thiết. Đó là khi cả hai phía của kênh giao tiếp đều sử dụng những định dạng dữ liệu, kỹ thuật, công nghệ ... giống nhau. Trong những trường hợp như thế, giải pháp tích hợp của ta cần có những xử lý linh hoạt nhằm không phải thực hiện những quá trình chuyển đổi không cần thiết, như vậy sẽ nâng cao hiệu quả cũng như là hiệu suất hoạt động của hệ thống hơn.

5.3.2 Web services integration (WSI) và Service-oriented integration (SOI)

WSI và SOI hiện đang là hai giải pháp giành được nhiều sự quan tâm của các tổ chức doanh nghiệp cũng như là các nhà quản trị hệ thống trong việc giải quyết các vấn đề tích hợp hợp hệ thống.

5.3.2.1 Web services integration (WSI)

Một dự án WSI thường được xây dựng chỉ để giải quyết vấn đề trao đổi dữ liệu giữa một số lượng nhỏ các hệ thống (hai hay bốn).

Đầu tiên nhóm phát triển sẽ định nghĩa những thông điệp SOAP căn cứ theo những cơ sở sau:

- Dữ liệu cần trao đổi giữa các hệ thống
- Các định dạng thông điệp mà các hệ thống hiện hành có thể hiểu và làm việc trên đó.
- Khả năng truy cập vào các dữ liệu này trong hệ thống thông qua tập các phương thức hay hàm APIs mà các hệ thống này hỗ trợ.

Tiếp theo, nhóm sẽ định nghĩa các thông tin mô tả service, bao gồm thông tin về cách thức giao tiếp, tập các phương thức, và các mẫu trao đổi thông điệp sao cho đáp ứng được các yêu cầu của dự án (hiện tại). Bên cạnh đó, các vấn đề liên quan đến chất lượng dịch vụ như là bảo mật, an toàn đường truyền, quản lý giao tác, khả năng xử lý lỗi... sẽ được xử lý sau (xem như phần mở rộng, cần đến đâu thì thực hiện đến đó.)

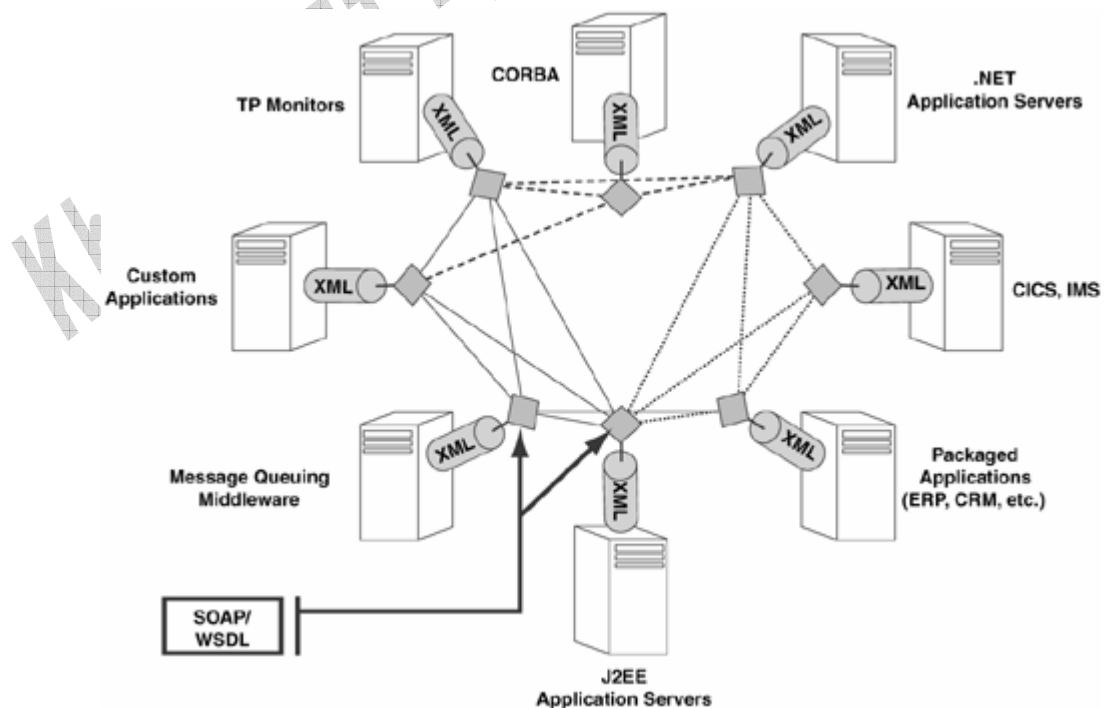
Một vài ưu điểm của giải pháp WSI:

- Thời gian triển khai nhanh, đặc biệt khi số lượng các hệ thống không nhiều.
- Chi phí đầu tư thấp

Một số hạn chế của giải pháp này:

- Không quan tâm, đầu tư nhiều cho việc xây dựng các mô hình xử lý về dữ liệu, dịch vụ... nhằm mục đích có thể tái sử dụng trong các dự án sau.

- Các hệ thống gửi thông điệp SOAP một cách trực tiếp thông qua tầng vận chuyển hay dùng các APIs của các middleware. Điều này sẽ gây khó khăn khi có nhu cầu chuyển đổi sang một tầng vận chuyển hay một sản phẩm middleware khác.
- Vấn đề bảo mật chỉ được giải quyết không triệt để, nghĩa là không đưa ra một kiến trúc hoàn chỉnh để có thể dùng chung cho nhiều dự án. Vì thế sẽ gặp nhiều khó khăn sau này khi cần thực hiện liên kết, phối hợp hoạt động giữa các hệ thống.



Hình 5-7 – Web services integration (WSI)

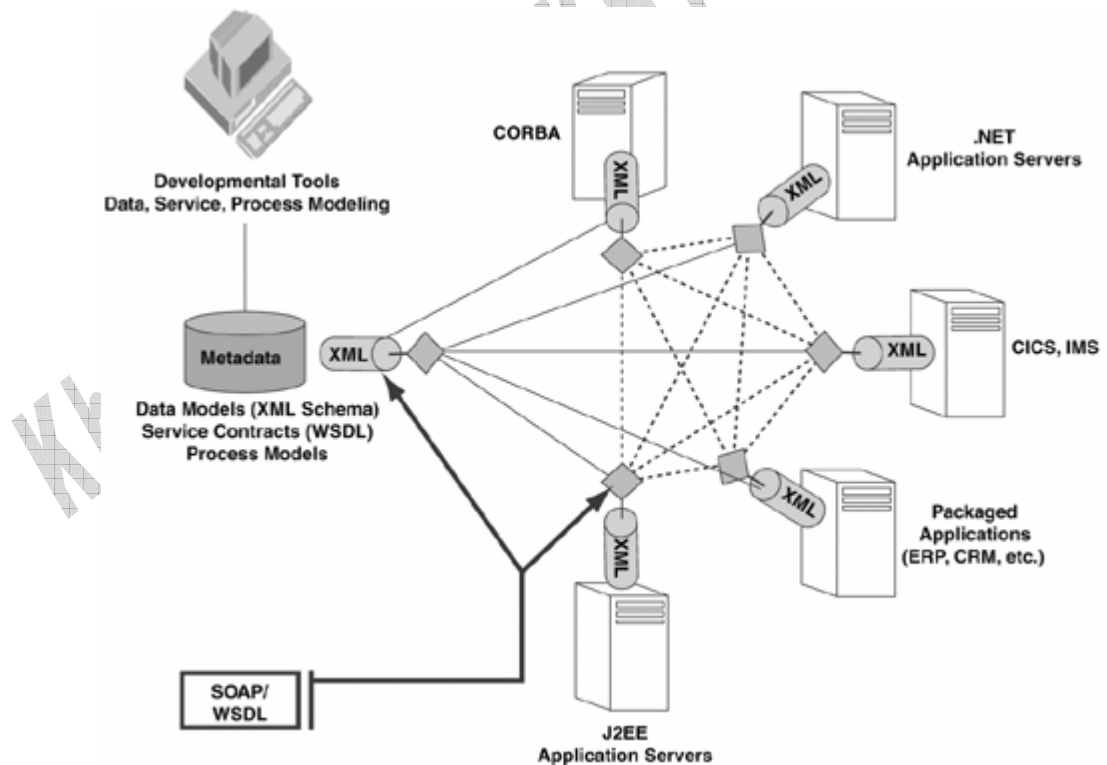
5.3.2.2 *Service-oriented integration (SOI)*

SOI là giải pháp tích hợp sử dụng web service với những nguyên tắc thiết kế của kiến trúc hướng dịch vụ (SOA). SOI là giải pháp có tính chất chiến lược và thích hợp cho các dự án mà có quan tâm đến lợi ích lâu dài.

SOI bắt đầu bởi giai đoạn “khởi tạo”

- Xây dựng một nền tảng cho kiến trúc hướng dịch vụ (SOA), các quy trình, nguyên tắc xử lý, mô hình và công cụ hỗ trợ.

- Xác định mô hình về tập các dịch vụ sẽ được sử dụng. Các mô hình này không cần phải thật toàn diện, chỉ cần xác định rõ ràng thông tin về kiểu dữ liệu, thông tin về dịch vụ, và các quy trình cần chia sẻ với các hệ thống khác.
- Thực hiện liệt kê và phân loại toàn bộ các dịch vụ được dùng trong các dự án nhằm hỗ trợ việc tái sử dụng lại các dịch vụ trong các dự án sau.



Hình 5-8 – Service-oriented integration (SOI)

Một dự án SOI sẽ thực hiện các công việc sau:

- Tinh chế lại các mô hình dữ liệu của các hệ thống sao cho phù hợp với dự án hiện tại.
- Xây dựng các đặc tả dịch vụ dựa trên những gì có trong phần thông tin mô tả của dịch vụ, bao gồm nguyên tắc xử lý, yêu cầu về bảo mật, yêu cầu về quản lý...
- Xây dựng các đối tượng bao bọc các hệ thống cũ dựa trên các đặc tả dịch vụ (thực hiện đóng gói lại thành phần xử lý bên trong của các hệ thống này).
- Xây dựng các dịch vụ cần thiết cho dự án.

- Xây dựng các bộ chuyển đổi dữ liệu nhằm thực hiện chuyển đổi dữ liệu giữa các mô hình dữ liệu khác nhau của các hệ thống.
- Xây dựng môi trường thực thi cho web service nhằm hỗ trợ những tính năng mở rộng như quản lý giao tác, đảm bảo an toàn thông điệp truyền, khả năng xử lý lỗi...

Các ưu điểm của giải pháp SOI:

- Khả năng tái sử dụng lại các mô hình dữ liệu, dịch vụ và qui trình xử lý trong nhiều dự án tích hợp sau này.
- Giảm sự lệ thuộc vào một hãng, hay một middleware nào đó thông qua việc xây dựng được một tầng trừu tượng dựa trên các chuẩn của web service.
- Thừa hưởng được những lợi ích của web service như bảo mật, an toàn đường truyền, quản lý giao tác, xử lý lỗi..
- Hình thành được một kiến trúc bảo mật có khả năng mở rộng để đáp ứng được yêu cầu liên kết giữa các hệ thống, tổ chức. (hỗ trợ việc triển khai single-sign on, ...)

Một số giới hạn của giải pháp này:

- Đòi hỏi chi phí *ban đầu* lớn
- Thời gian triển khai *ban đầu* lớn
- Nhóm phát triển đòi hỏi phải có kỹ năng và trình độ về kiến trúc hệ thống.
- Đòi hỏi sự hỗ trợ của các nhà quản lý nghiệp vụ và quản lý kỹ thuật.

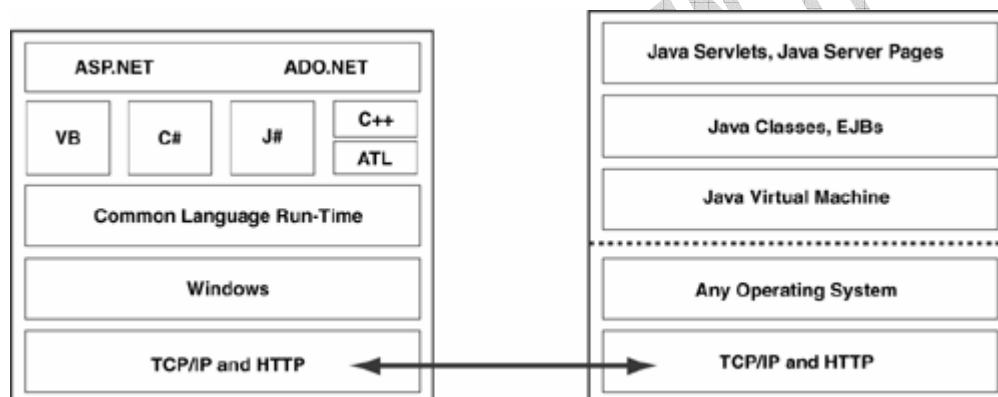
5.4 Ứng dụng SOA và web service để tích hợp các hệ thống được xây dựng trên .NET và J2EE

Các web service được xây dựng nhằm hỗ trợ trong việc trao đổi dữ liệu giữa các ứng dụng hay dịch vụ, và cho phép cung cấp phương thức của một đối tượng để có thể được truy cập bởi các đối tượng phần mềm hay ứng dụng khác thông qua môi trường mạng.

Hiện nay, .NET và J2EE là hai hệ nền được nhiều người sử dụng để triển khai các hệ thống lớn. Và nhu cầu cho việc tích hợp hay liên kết giữa các hệ thống này là có thực và ngày càng trở nên cấp thiết. Mọi người mong muốn và đang cố gắng có thể tạo được sự liên kết giữa các đối tượng được phát triển trên J2EE (Java Bean) và các đối tượng được phát triển trên nền .NET. Nhưng điều này không dễ dàng thực hiện vì kiến trúc của hai hệ nền này tương đối khác nhau nhiều.

- .NET được thiết kế để có thể tương thích tốt với hệ điều hành Windows, và sử dụng lại phần lớn những tính năng nền tảng của Windows như đa luồng, quản lý bộ nhớ, truy cập hệ thống tập tin và tập những hàm APIs cấp hệ thống.
- J2EE được xây dựng dựa trên những tính năng của máy ảo Java để hỗ trợ cho nhiều hệ điều hành.

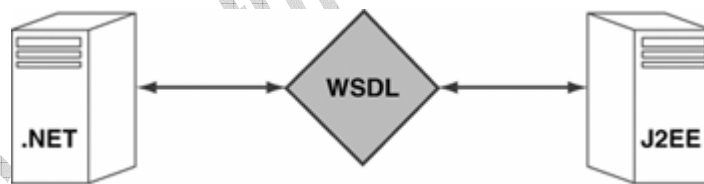
Các web service có thể dùng để thiết lập mối liên kết giữa các hệ thống, ứng dụng được phát triển dựa trên hai hệ nền này. Tuy nhiên, vẫn còn một vài hạn chế bởi các tính năng hiện đang được hỗ trợ (đến thời điểm hiện tại) của web service vẫn chưa thật sự là đầy đủ. Ngoài ra còn là sự khác biệt quá lớn về kiến trúc giữa hai hệ nền này. Hình 5-9 đặt các tầng của kiến trúc .NET và J2EE cạnh nhau để thể hiện rõ những sự khác nhau giữa hai hệ nền



Hình 5-9 – Sự khác nhau giữa kiến trúc .NET và J2EE

Với sự khác nhau cơ bản như thế, nên việc tích hợp, liên kết giữa hai hệ nền .NET và J2EE có một số giới hạn và chỉ có thể đạt được ở một mức độ trừu tượng khá cao. Giải pháp tốt nhất đó là xây dựng các đặc tả dịch vụ (như là các tập tin WSDL) để xác định gói các đối tượng sẽ được trao đổi hay gói các phương thức sẽ được gọi.

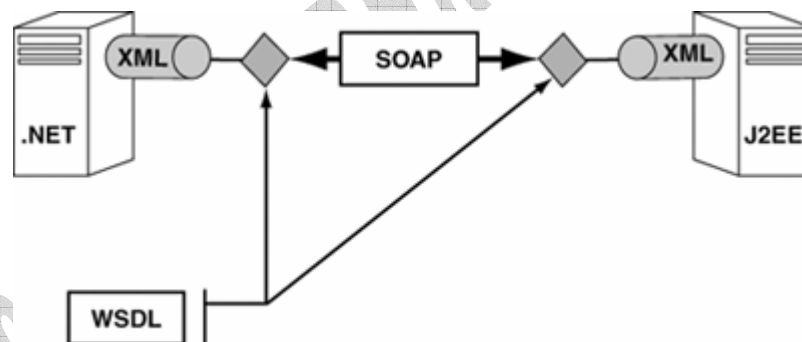
- Ví dụ như, nếu các ứng dụng cần chia sẻ thông tin về khách hàng, thì ta có thể định nghĩa một lược đồ (Xml schema) mô tả loại thông tin trên, định nghĩa thông tin mô tả các phương thức dựa trên lược đồ này (trong các tập tin WSDL) và sau cùng là xây dựng các thông điệp SOAP dựa trên các nguồn thông tin vừa xây dựng. Ta thấy rằng, khi đó các tập tin WSDL và các lược đồ dữ liệu đóng vai trò rất quan trọng trong việc thực hiện mỗi liên kết này vì đây chính là các mô hình dữ liệu mà hai bên cùng chia sẻ.



Hình 5-10 – Vai trò của WSDL trong liên kết các hệ thống.

- o Hình 5-10 cho thấy khi một hệ thống xây dựng trên nền .NET và một hệ thống xây dựng trên nền J2EE cùng chia sẻ và cùng hiểu một tập tin WSDL thì chúng có thể liên kết với nhau.

Bởi vì web service không hỗ trợ đầy đủ các đặc trưng và tính năng của .NET và J2EE nên mức độ liên kết vẫn còn hạn chế. Cụ thể, các chức năng về quản lý chu kỳ sống của đối tượng, quản lý các phiên giao dịch ... của .NET và J2EE không được hỗ trợ bởi web service. Hình 5-11 minh họa các trao đổi thông điệp SOAP giữa hai hệ thống sau khi đã thiết lập và chia sẻ cùng một đặc tả dịch vụ WSDL



Hình 5-11 – WSDL mô tả cách các thông điệp SOAP được xử lý

Vì các thông điệp SOAP là các tài liệu XML, nên đòi hỏi các bên phải có khả năng hiểu và xử lý các dữ liệu dạng XML. Ngoài ra, các thông điệp này được phát sinh dựa trên WSDL mà đã đạt được thỏa thuận của các bên, nên có thể xem như là một “ngôn ngữ chung” cho các hệ thống để có thể hiểu và xử lý trong quá trình liên kết.

5.5 Ứng dụng SOA và web service trong việc tích hợp các hệ thống cũ

Công nghệ web service có thể được dùng để xây dựng cầu nối giao tiếp cho các hệ thống xây dựng trên những hệ nền, sử dụng những công nghệ hay chuẩn khác xa nhau, như là .NET, J2EE, CORBA, WebSphere MQ, hay các ứng dụng đóng gói. Thành phần giao tiếp này sử dụng ngôn ngữ XML và thể hiện các qui ước về trao đổi thông điệp giữa các hệ thống.

Một trong các ưu điểm của một cầu nối giao tiếp được định nghĩa tốt đó là nó có thể được mở rộng để có thể bổ sung thêm các đặc tính mới của web service hay tích hợp thêm các hệ thống cũ. Khi đó, các hệ thống cũ sẽ được dịch vụ hóa bằng cách định nghĩa thêm các đặc tả dịch vụ cho chúng và xây dựng thêm các bộ xử lý thông điệp SOAP. Các bộ xử lý thông điệp SOAP này có khả năng nhận các thông điệp SOAP và chuyển đổi chúng sang dạng thông điệp hay lời gọi phương thức đặc thù của hệ thống cũ. Lợi ích đạt được của cách làm này là vô cùng lớn, nó cho phép các tổ chức có thể tái sử dụng lại các giá trị sẵn có và không phải thực hiện công việc gỡ-bỏ-và-thay-thế đầy tốn kém và rủi ro.

Một số dạng hệ thống cũ có khả năng dịch vụ hóa, bao gồm:

- Các hệ thống mainframe (ví dụ như CICS và IMS)
- Các ứng dụng phân tán hướng đối tượng (như CORBA, DCOM, EJB)
- Các hệ thống xử lý giao tác (như Tuxedo và Encina)
- Các ứng dụng đóng gói (như các ứng dụng của SAP, PeopleSoft, Oracle).
- Các hệ quản trị cơ sở dữ liệu (như Oracle, Sybase, DB2, SQL Server)
- Các hệ thống B2B và xử lý thông điệp (như SWIFT, EDIFACT, X12, HL7, WebSphere MQ, JMS, MSMQ).

❑ **Ví dụ :** Service hóa một CORBA server:

- IDL (Interface Definition Language) là ngôn ngữ chuẩn dùng để xây dựng phần giao tiếp cho các CORBA server. Tổ chức OMG đã định nghĩa một đặc tả để thực hiện ánh xạ từ IDL sang WSDL. Dựa trên cơ sở này ta có thể chuyển đổi một CORBA IDL sang một đặc tả dịch vụ WSDL, bao gồm các thông tin về: kiểu dữ liệu, các thông điệp, cổng giao tiếp, và các thông tin kết nối.
- Bước đầu tiên trong việc dịch vụ hóa một CORBA server đó là phải chuyển đổi CORBA IDL sang WSDL. Sau đây là một ví dụ đơn giản về một CORBA IDL với một thành phần giao tiếp và hai phương thức:

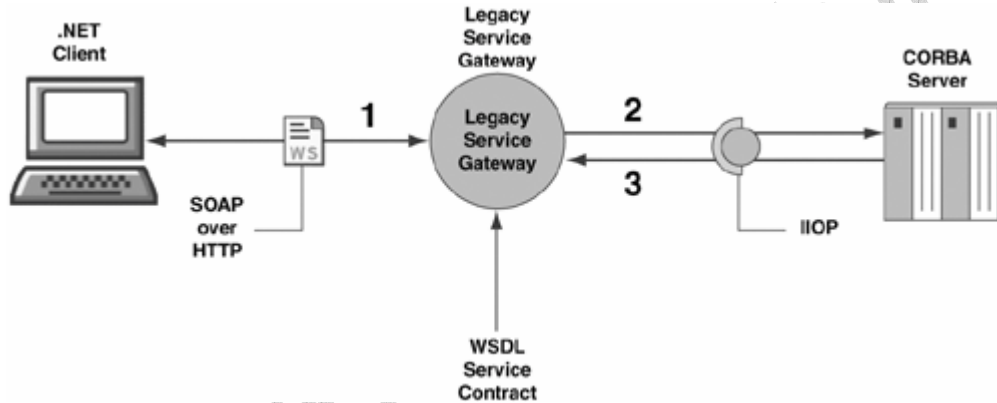
```
interface HelloWorld {
    string sayHi ();
    string greetMe (in string user);
};
```

- Và đây là một phần của đặc tả dịch vụ WSDL tương ứng, bao gồm một cổng giao tiếp và hai phương thức.

```
<types>
  <schema
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    <element name="HW.HelloWorld.sayHi.return" type="xsd:string"/>
    <element name="HW.HelloWorld.greetMe.user" type="xsd:string"/>
    <element name="HW.HelloWorld.greetMe.return" type="xsd:string"/>
  </schema>
</types>

<message name="HW.HelloWorld.sayHi"/>
<message name="HW.HelloWorld.sayHiResponse">
  <part element="xsd1:HW.HelloWorld.sayHi.return" name="return"/>
</message>
<message name="HW.HelloWorld.greetMe">
  <part element="xsd1:HW.HelloWorld.greetMe.user" name="user"/>
</message>
<message name="HW.HelloWorld.greetMeResponse">
  <part element="xsd1:HW.HelloWorld.greetMe.return" name="return"/>
</message>
<portType name="HW.HelloWorld">
  <operation name="sayHi">
    <input message="tns:HW.HelloWorld.sayHi" name="sayHi"/>
    <output message="tns:HW.HelloWorld.sayHiResponse"
name="sayHiResponse"/>
  </operation>
  <operation name="greetMe">
    <input message="tns:HW.HelloWorld.greetMe" name="greetMe"/>
    <output message="tns:HW.HelloWorld.greetMeResponse"
name="greetMeResponse"/>
  </operation>
</portType>
```


- Một file WSDL đầy đủ có thể nhúng vào một IDE (như là Microsoft Visual Studio) để xây dựng một đối tượng yêu cầu dịch vụ (proxy)



Hình 5-12 – Dịch vụ hóa một CORBA server

- Cơ chế vận hành sẽ như sau:
 - o Đối tượng sử dụng dịch vụ sẽ gửi một thông điệp SOAP đến legacy service gateway thông qua nghi thức HTTP.
 - o Legacy service gateway có nhiệm vụ chuyển đổi các thông điệp SOAP này sang một hay nhiều lời gọi các đối tượng trên CORBA server
 - o Legacy service gateway cũng sẽ phải thực hiện chuyển các thông tin phản hồi từ CORBA server thành các thông điệp SOAP và trả chúng về cho phía yêu cầu dịch vụ
- Tùy thuộc vào việc xây dựng, legacy service gateway có thể là một thư viện được nạp vào khi chạy các đối tượng yêu cầu dịch vụ, hay khi chạy CORBA server, và cũng có thể là một ứng dụng độc lập.
- Đặc biệt, sẽ tốt hơn nếu legacy service gateway hỗ trợ việc quản lý cơ chế hoạt động của nó các thông qua thông tin cấu hình dựa trên đặc tả dịch vụ WSDL. Điều này thực hiện được nếu như đặc tả dịch vụ WSDL chứa thêm các thông tin như là cổng giao tiếp (portType) và kết nối (SOAP binding và CORBA/IIOP binding).

o HelloWorld portType

```
<portType name="HW.HelloWorld">
  <operation name="sayHi">
    <input message="tns:HW.HelloWorld.sayHi" name="sayHi"/>
    <output message="tns:HW.HelloWorld.sayHiResp" name="sayHiResp"/>
  </operation>
  <operation name="greetMe">
    <input message="tns:HW.HelloWorld.greetMe" name="greetMe"/>
    <output message="tns:HW.HelloWorld.greetMeResp" name="greetMeResp"/>
  </operation>
</portType>
```

o SOAP binding

```
<binding name="HW.HelloWorldf_SOAPBinding" type="tns:HW.HelloWorld">
  <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="sayHi">
    <soap:operation soapAction="" style="document"/>
    <input name="sayHi"><soap:body use="literal"/></input>
    <output name="sayHiResponse"><soap:body
use="literal"/></output>
  </operation>
  <operation name="greetMe">
    <soap:operation soapAction="" style="document"/>
    <input name="greetMe"><soap:body use="literal"/></input>
    <output name="greetMeResponse"><soap:body
use="literal"/></output>
  </operation>
</binding>
...
<port binding="tns:HW.HelloWorldf_SOAPBinding" name="SoapPort">
  <soap:address location="http://localhost:9000"/>
</port>
```

o CORBA/IIOP binding

```
<binding name="HW.HelloWorldBinding" type="tns:HW.HelloWorld">
  <corba:binding repositoryID="IDL:HW/HelloWorld:1.0"/>
  <operation name="sayHi">
    <corba:operation name="sayHi">
      <corba:return idltype="corba:string" name="return"/>
    </corba:operation>
    <input name="sayHi"/>
    <output name="sayHiResponse"/>
  </operation>
  <operation name="greetMe">
    <corba:operation name="greetMe">
      <corba:param idltype="corba:string" mode="in" name="user"/>
      <corba:return idltype="corba:string" name="return"/>
    </corba:operation>
    <input name="greetMe"/>
    <output name="greetMeResponse"/>
  </operation>
</binding>
```

```
...  
<port binding="tns:HW.HelloWorldBinding" name="HW.HelloWorldPort">  
  <corba:address location="file:../../HelloWorld.ior"/>  
</port>
```

Kỹ thuật này có thể được sử dụng cho những hệ thống cũ nào mà hỗ trợ IDL (như là Tuxedo FML) hay hỗ trợ các tính năng về reflection (Java, COM, lược đồ dữ liệu của các hệ quản trị cơ sở dữ liệu quan hệ.). Và dễ thấy rằng, kỹ thuật này sẽ đơn giản hơn khi có những chuẩn để thực hiện ánh xạ sang đặc tả dịch vụ WSDL.

Chương 6

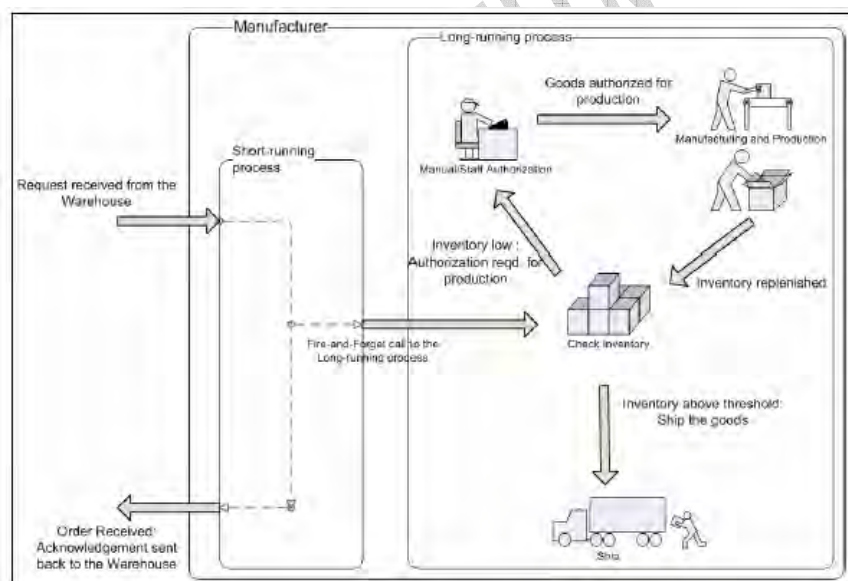
SOA VÀ QUẢN LÝ TIẾN TRÌNH NGHIỆP VỤ

Chương 6 sẽ trình bày một số khái niệm liên quan về quản lý tiến trình. Phân tích mối quan hệ kết hợp giữa quản lý tiến trình, SOA và Web services. Xem xét các vấn đề liên quan đến thiết kế tiến trình nghiệp vụ. Ngoài ra, chương này cũng sẽ giới thiệu về một số ngôn ngữ đặc tả tiến trình hiện đang được sử dụng phổ biến, như là Web Service Flow Language (WSFL), XLANG, Web Service Choreography Interface (WSCI) và Business Process Execution Language For Web Service (BPEL4WS)

6.1 Một số khái niệm cơ bản về Quản lý tiến trình nghiệp vụ

6.1.1 Tiến trình nghiệp vụ

Tiến trình nghiệp vụ là hoạt động trong thế giới thực, trong đó bao gồm một chuỗi các tác vụ được liên kết, phối hợp và thực hiện theo một trình tự thích hợp, với những qui định, ràng buộc nhằm hướng đến một mục tiêu.



Hình 6-1 – Minh họa một tiến trình nghiệp vụ

Có hai loại tiến trình nghiệp vụ:

- **Tiến trình không trạng thái:** là những tiến trình chỉ được thực thi trong bộ nhớ mà không lưu lại trạng thái vào cơ sở dữ liệu khi chúng ngừng hoạt động. Các tiến trình không trạng thái thích hợp cho những quy trình mà đòi hỏi thời gian xử lý ngắn (tương tác theo cơ chế đồng bộ), hiệu suất hoạt động cao và trong quá trình thực thi, không cần sự tác động của yếu tố con người.
- **Tiến trình có trạng thái:** là những tiến trình mà được thực thi trong nhiều giao tác. Thông tin trạng thái của tiến trình được lưu trữ trong cơ sở dữ liệu mỗi khi ngừng hoạt động. Dạng tiến trình này thích hợp cho những quy trình phức tạp, đòi hỏi thời gian xử lý dài (hỗ trợ tương tác bất đồng bộ). Các tiến trình dạng này cũng phải đáp ứng được các yêu cầu về tính ổn định, độ tin cậy, khả năng xử lý lỗi và khôi phục trạng thái. Trong quá trình thực thi, có thể cần tương tác với yếu tố con người.

6.1.2 Quản lý tiến trình

Quản lý tiến trình quan tâm đến cách xác định, mô hình hóa, phát triển, triển khai, và quản lý các tiến trình nghiệp vụ, bao gồm các tiến trình mà đòi hỏi sự hỗ trợ của các hệ thống tin học và tác động của yếu tố con người. Quản lý tiến trình ra đời từ rất lâu, bắt đầu là các hệ thống luồng công việc và phát triển cho tới bây giờ là các hệ thống phối hợp các web service.

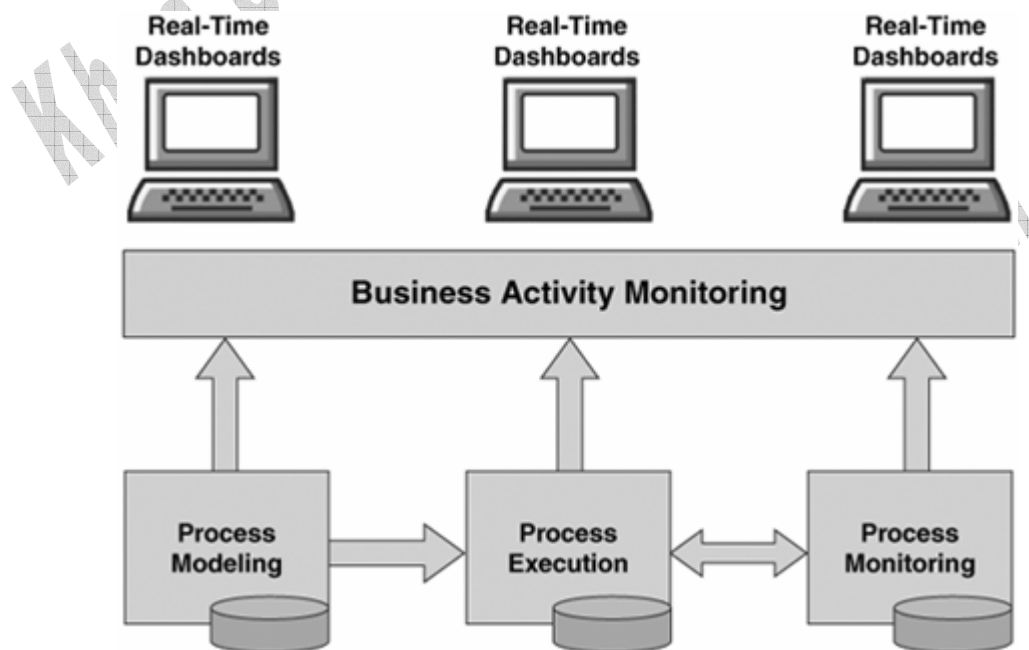
Những mục tiêu và lợi ích chính của BPM:

- Giảm những khó khăn về sự không nhất quán giữa các yêu cầu nghiệp vụ và các hệ thống tin học bằng cách cho phép những đối tượng làm công tác nghiệp vụ đưa ra các mô hình về tiến trình xử lý và sau đó chuyển mô hình này cho bộ phận tin học để xây dựng cơ chế thực thi và quản lý cho các tiến trình này.
- Tăng hiệu suất làm việc của các nhân viên bằng cách qui trình hóa và tự động hóa các thao tác nghiệp vụ.

- Tăng tính linh hoạt và khả năng cơ động bằng cách tách biệt phân xử lý ra khỏi các quy tắc nghiệp vụ, biểu diễn các quy trình xử lý dưới một dạng mà nó có thể dễ dàng đáp ứng được các thay đổi của yêu cầu, của thị trường.

6.1.3 Hệ quản lý tiến trình:

Một hệ quản lý tiến trình cung cấp các kỹ thuật để hỗ trợ việc định nghĩa, mô hình hóa, phát triển, triển khai, và quản lý các tiến trình nghiệp vụ. Hầu hết các hệ quản lý tiến trình đều cung cấp một bộ designer để mô hình hóa các tiến trình, trong đó mỗi nút sẽ tương ứng với một tác vụ và mỗi đường nối tượng trưng cho các luồng xử lý hay luồng dữ liệu liên kết các tác vụ với nhau. Tuy nhiên, một hệ quản lý tiến trình đầy đủ, phải có nhiều hơn thế nữa.



Hình 6-2 – Các thành phần của một hệ thống quản lý tiến trình nghiệp vụ

- Mô hình hóa tiến trình xử lý:
 - ▶ Mô hình hóa các yêu cầu nghiệp vụ (trong giai đoạn phân tích)
 - ▶ Mô hình hóa ràng buộc giữa các tác vụ: trình tự thực hiện, khi nào thì được kích hoạt, đối tượng nào sẽ thực hiện.
 - ▶ Mô hình hóa các nguyên tắc kèm theo với tiến trình.

- Thực thi tiến trình:
 - ▶ Bao gồm các engine đảm nhiệm việc thực thi tiến trình, quản lý các thể hiện của tiến trình.
 - ▶ Thực thi tiến trình với các ràng buộc sau:
 - Đảm bảo thực thi các tác vụ đúng trình tự.
 - Đảm bảo đối tượng thực thi tác vụ có đầy đủ quyền.
 - Theo dõi trạng thái của tiến trình: tác vụ nào đã hoàn thành, tác vụ nào đủ điều kiện để thực thi, kiểm tra thời gian còn hiệu lực của tiến trình và các tác vụ ...
- Giám sát tiến trình:
 - ▶ Bao gồm các công cụ hỗ trợ những người sử dụng tiến trình và các chuyên viên quản trị hệ thống có thể theo dõi và điều khiển các tiến trình.
 - Các thông tin theo dõi bao gồm: thông tin về các tiến trình đang được thực thi, thông tin về các tiến trình đã hoàn thành...
 - Các điều khiển bao gồm: hoãn và tiếp tục thực thi một tiến trình, thay đổi quyền ưu tiên của tiến trình...
- Business Activity Monitor (BAM):
 - ▶ Lắng nghe các sự kiện, phân tích các thông tin trạng thái của tiến trình nhằm kịp thời gửi các phản hồi cho các đối tượng có trách nhiệm theo dõi, giám sát...

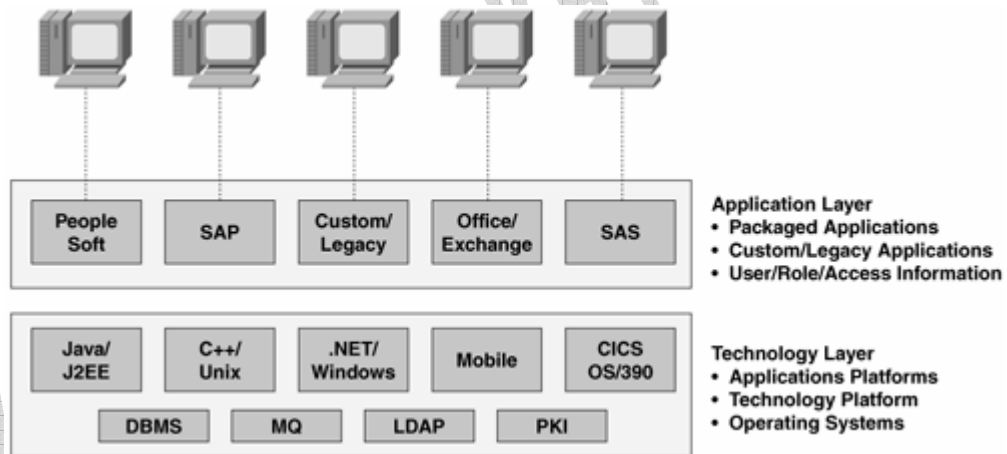
6.2 Quản lý tiến trình, SOA và Web Service

Phần này sẽ bàn về lợi ích của việc kết hợp Quản lý tiến trình, SOA và web service. Các lợi ích này bao gồm:

- Xây dựng được một hệ thống quản lý tiến trình linh hoạt và bền vững hơn.
- Khả năng tạo, quản lý, và bảo trì các ứng dụng tổng hợp một cách dễ dàng hơn.

6.2.1 Quản lý tiến trình, SOA và Web Service được kết hợp thế nào

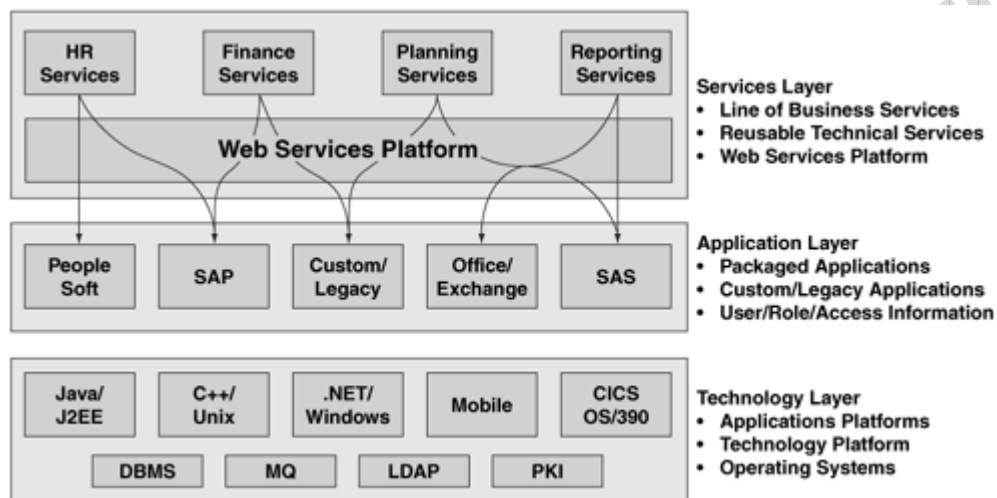
Hầu hết các tổ chức đều có nhiều hệ thống “không đồng nhất”, với nhiều loại ứng dụng, và sử dụng nhiều loại công nghệ khác nhau. Việc chia sẻ thông tin giữa các ứng dụng này gặp rất nhiều khó khăn bởi sự khác biệt về công nghệ và các mô hình dữ liệu, mô hình đối tượng.



Hình 6-3 – Thực trạng “không đồng nhất” của các hệ thống doanh nghiệp.

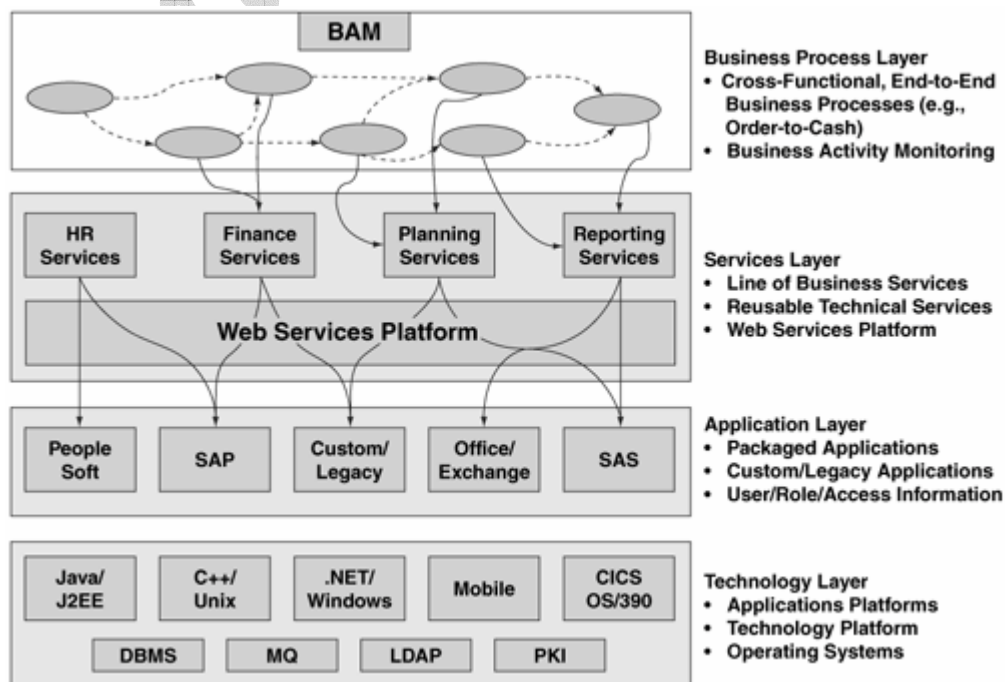
Khi triển khai SOA với công nghệ web service thì xuất hiện thêm một tầng mới: tầng dịch vụ. Trong Hình 6-4, tầng dịch vụ này bao gồm:

- Các dịch vụ nghiệp vụ tương ứng với các lĩnh vực nghiệp vụ trong thực tế (Nhân sự, Tài chính, Kế hoạch, Thống kê), kèm theo các mô hình dữ liệu.
- Các dịch vụ kỹ thuật với khả năng tái sử dụng để có thể chia sẻ giữa các lĩnh vực nghiệp vụ.
- Web services platform hỗ trợ cho việc xây dựng và sử dụng các dịch vụ một cách độc lập với tầng ứng dụng và tầng kỹ thuật bên dưới.



Hình 6-4 – Tầng dịch vụ dựa trên mô hình SOA với công nghệ Web Service

Tầng kế tiếp sẽ là tầng tiến trình nghiệp vụ



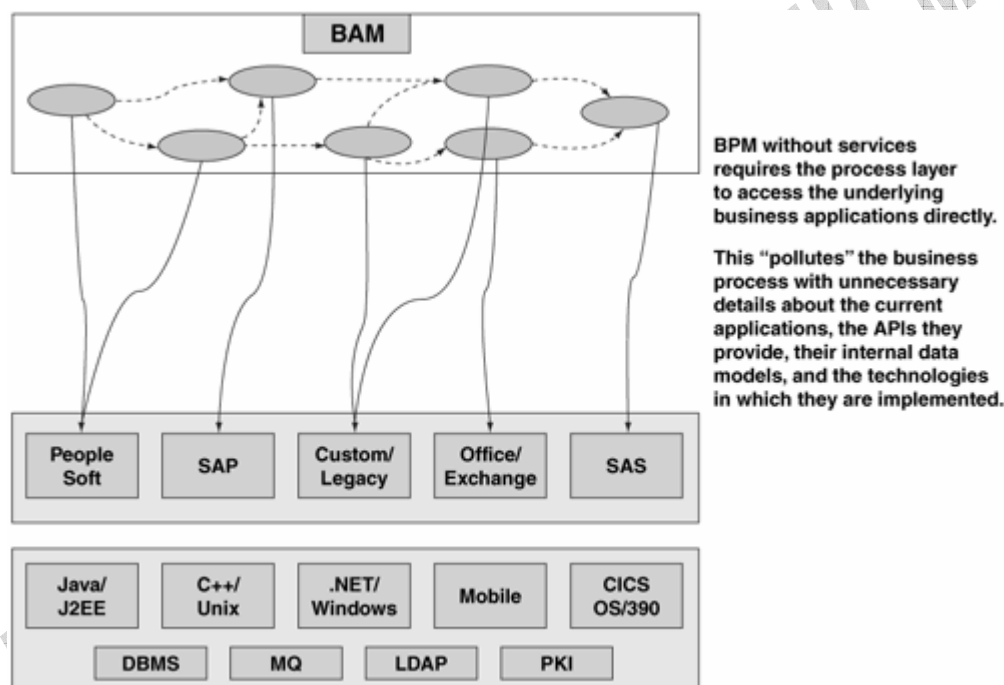
Hình 6-5 – Tầng tiến trình nghiệp vụ sử dụng tầng dịch vụ

Sự hỗ trợ của tầng dịch vụ cho tầng tiến trình nghiệp vụ như sau:

- Việc liên kết các dịch vụ nghiệp vụ sẽ tương ứng với các tác vụ trong một tiến trình nghiệp vụ.

- Các tiến trình nghiệp vụ sẽ liên kết với các dịch vụ nghiệp vụ thông qua các đặc tả dịch vụ. Như thế, các tiến trình nghiệp vụ sẽ không cần quan tâm đến chi tiết của các tầng ứng dụng và tầng kỹ thuật bên dưới.
- Cung cấp các tính năng quản lý dịch vụ nhằm hỗ trợ tầng tiến trình nghiệp vụ có thể linh động hơn trong việc xác định và truy cập các dịch vụ.
- Mô hình bảo mật của tầng dịch vụ cung cấp tính năng “single sign-on” và “điều khiển truy cập thông qua vai trò” để hỗ trợ tầng tiến trình nghiệp vụ không cần phải quan tâm đến những cơ chế bảo mật khác nhau mà hai tầng ứng dụng và kỹ thuật ở bên dưới sử dụng.

Trước đây, hầu hết các hệ thống đều không xây dựng tầng dịch vụ dựa trên mô hình SOA với công nghệ web service. Một hệ thống quản lý tiến trình mà không có tầng dịch vụ thì rất phức tạp, dễ đổ vỡ. Bởi vì tầng tiến trình nghiệp vụ khi đó cần phải truy cập trực tiếp xuống tầng ứng dụng. Khi đó sự ràng buộc giữa hai lĩnh vực: nghiệp vụ và kỹ thuật là quá chặt chẽ.



Hình 6-6 – Các hệ thống BPM khi không có tầng services

- Giải pháp này phức tạp bởi vì tầng tiến trình nghiệp vụ phải truy cập trực tiếp xuống tầng ứng dụng thông qua các thành phần giao tiếp định nghĩa bởi các ứng dụng (hàm APIs, thông điệp, các bảng dữ liệu...). Điều này đòi hỏi nhóm triển khai tầng tiến trình phải quan tâm đến các thành phần giao tiếp này. Sự ràng buộc này dễ dẫn đến một kết quả không tốt trong trường hợp các thành phần giao tiếp được định nghĩa không tốt. Ngoài ra, lúc này tầng tiến trình nghiệp vụ cũng phải quan tâm đến việc chuyển đổi dữ liệu trong khi tương tác với tầng ứng dụng.
- Giải pháp này dễ đổ vỡ bởi vì tầng tiến trình bị ràng buộc quá chặt vào các đặc điểm của ứng dụng và thành phần giao tiếp của ứng dụng. Một ví dụ đơn giản, đó là khi cần phải cài đặt lại một phiên bản mới hơn của ứng dụng hay thay thế bằng các ứng dụng của các hãng khác thì có thể sẽ phá hỏng những liên kết giữa tiến trình và ứng dụng mà đã được xây dựng trước đó.

6.2.2 Phân tích một ví dụ kết hợp Quản lý tiến trình, SOA và web service

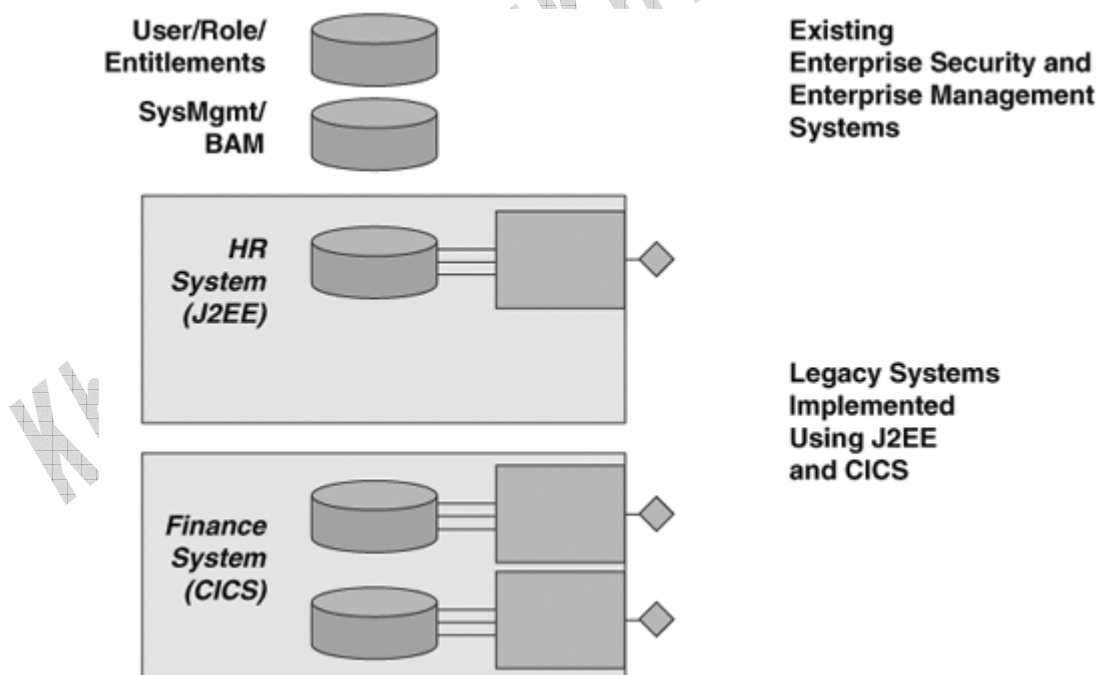
Phần này ta sẽ ứng dụng giải pháp kết hợp quản lý tiến trình, SOA và web service để giải quyết bài toán sau: “Tạo và cung cấp các dịch vụ nghiệp vụ cho các người dùng thuộc nội bộ và ở bên ngoài tổ chức dựa trên những hệ thống cũ.”

6.2.2.1 Mô tả các hệ thống cũ

Các hệ thống cũ của ta bao gồm:

- HR system (hệ thống quản lý nhân sự):
 - ▶ Hệ thống này được xây dựng trên nền J2EE.
 - ▶ Bao gồm: một hệ cơ sở dữ liệu, một mô hình đối tượng.
 - ▶ Cung cấp một thành phần giao tiếp dựa trên EJB
- Finance system (hệ thống quản lý tài chính)
 - ▶ Chạy trên máy mainframe.

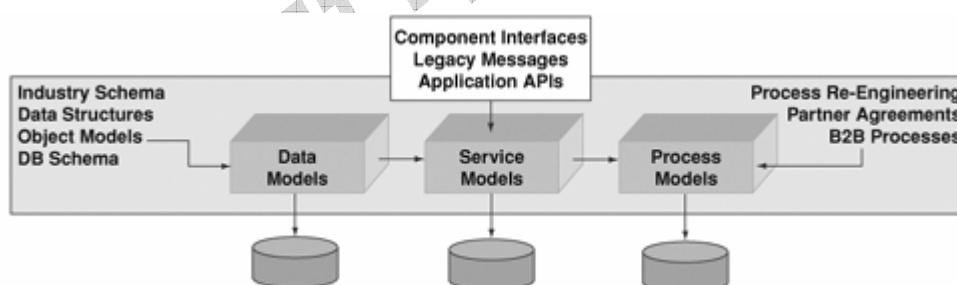
- ▶ Sử dụng CICS (Customer Information Control System) kết hợp với một cơ sở dữ liệu để quản lý giao tác.
- Enterprise Security (Hệ thống bảo mật)
 - ▶ Quản lý các thông tin về người dùng, vai trò, và quyền.
- Enterprise Management Systems (Hệ thống quản lý hệ thống)



Hình 6-7 – Ví dụ về các hệ thống cũ cần được service hóa.

6.2.2.2 Xây dựng các mô hình dữ liệu, dịch vụ và tiến trình

Bước đầu tiên trong việc tạo ra các dịch vụ (có khả năng tái sử dụng) theo các nguyên tắc thiết kế của SOA đó là phải định nghĩa các mô hình dữ liệu, dịch vụ và tiến trình



Hình 6-8 – Mô hình dữ liệu, dịch vụ và tiến trình.

➤ **Mô hình dữ liệu:**

- Định nghĩa các mô hình dữ liệu sẽ được trao đổi giữa các dịch vụ và giữa dịch vụ và đối tượng sử dụng dịch vụ. Vì thế mô hình này còn được gọi là mô hình dữ liệu mức dịch vụ.
- Mô hình này bao gồm: định nghĩa về dữ liệu (Xml schema), các nguyên tắc kiểm tra và ràng buộc về dữ liệu (Xml Schema và XPath), các nguyên tắc chuyển đổi dữ liệu (XSLT).
- Mô hình được xây dựng dựa trên các cấu trúc dữ liệu, mô hình đối tượng, lược đồ cơ sở dữ liệu của các hệ thống hiện tại.

➤ **Mô hình dịch vụ:**

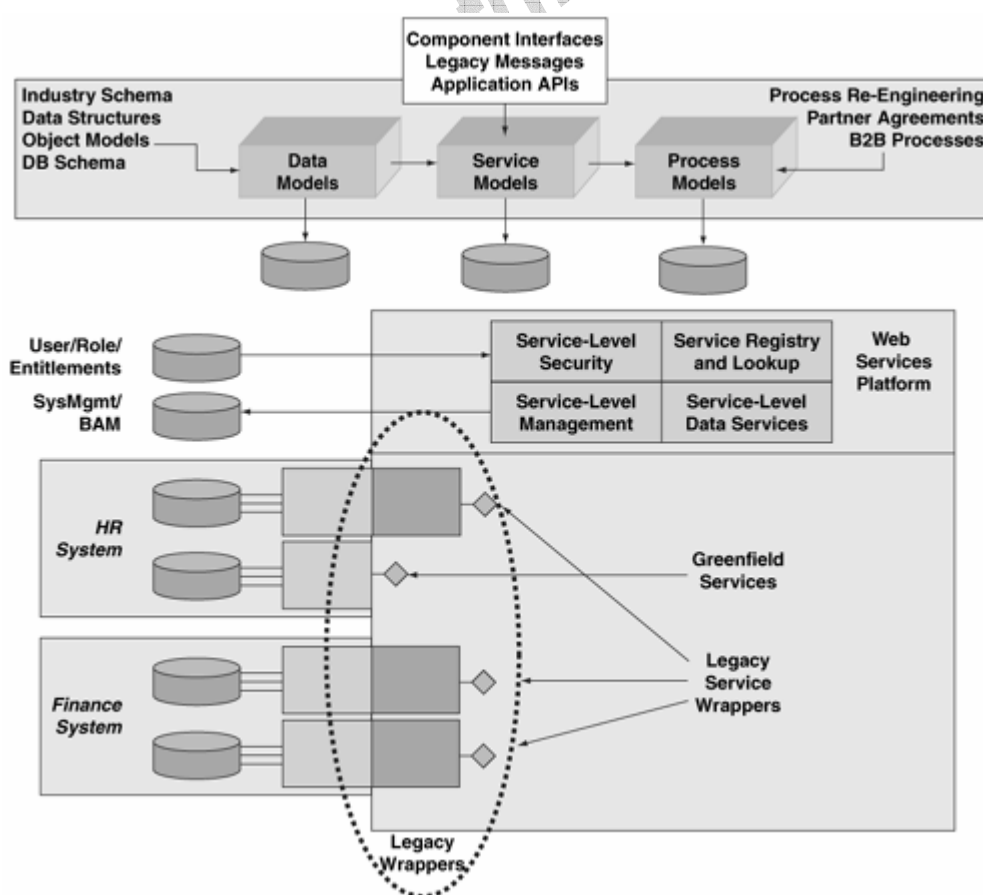
- Mô hình này xây dựng các đặc tả dịch vụ (bao gồm WSDL + WS-Policy) cho các dịch vụ nghiệp vụ.
- Các đặc tả nghiệp vụ này xác định các thông tin sau:
 - ▶ Các tham số đầu vào và ra cho các dịch vụ (có kiểu được xác định trong mô hình dữ liệu).
 - ▶ Mô tả về cơ chế bảo mật của dịch vụ (như là quyền, danh sách đối tượng được phép truy cập ...)
 - ▶ Chất lượng của dịch vụ (độ ưu tiên, an toàn đường truyền, quản lý giao tác,...).
 - ▶ Một số thông tin cấu hình (thời gian phản hồi, tình trạng).
- Mô hình này được xây dựng dựa trên thông tin giao tiếp của các thành phần hiện có trong hệ thống (COM/DCOM type library, CORBA IDL...), các định dạng thông điệp, hay tập các hàm APIs.

➤ **Mô hình tiến trình:**

- Mô hình này xây dựng các tiến trình nghiệp vụ dựa trên các ngôn ngữ đặc tả tiến trình (WS-BPEL).
- Mỗi tiến trình bao gồm các tác vụ cần thực hiện, luồng điều khiển giữa các tác vụ đó, luồng dữ liệu giữa các tác vụ...
- Mô hình này được xây dựng dựa trên các định nghĩa về các quy trình hiện có.

6.2.2.3 Xây dựng các dịch vụ cơ sở và Web service platform

Các dịch vụ đơn vị được xây dựng dựa trên mô hình dữ liệu và mô hình dịch vụ. Trong ví dụ này, ba trong số dịch vụ được định nghĩa bằng cách tạo ra các đối tượng bọc cho các hệ thống cũ. Các đối tượng bọc này thực hiện dịch vụ hóa các hệ thống cũ (SOAP và WSDL), có nhiệm vụ nhận các thông điệp SOAP được gửi đến, chuyển chúng sang định dạng mà hệ thống cũ có thể hiểu, và chuyển yêu cầu xử lý đến cho các hệ thống cũ.



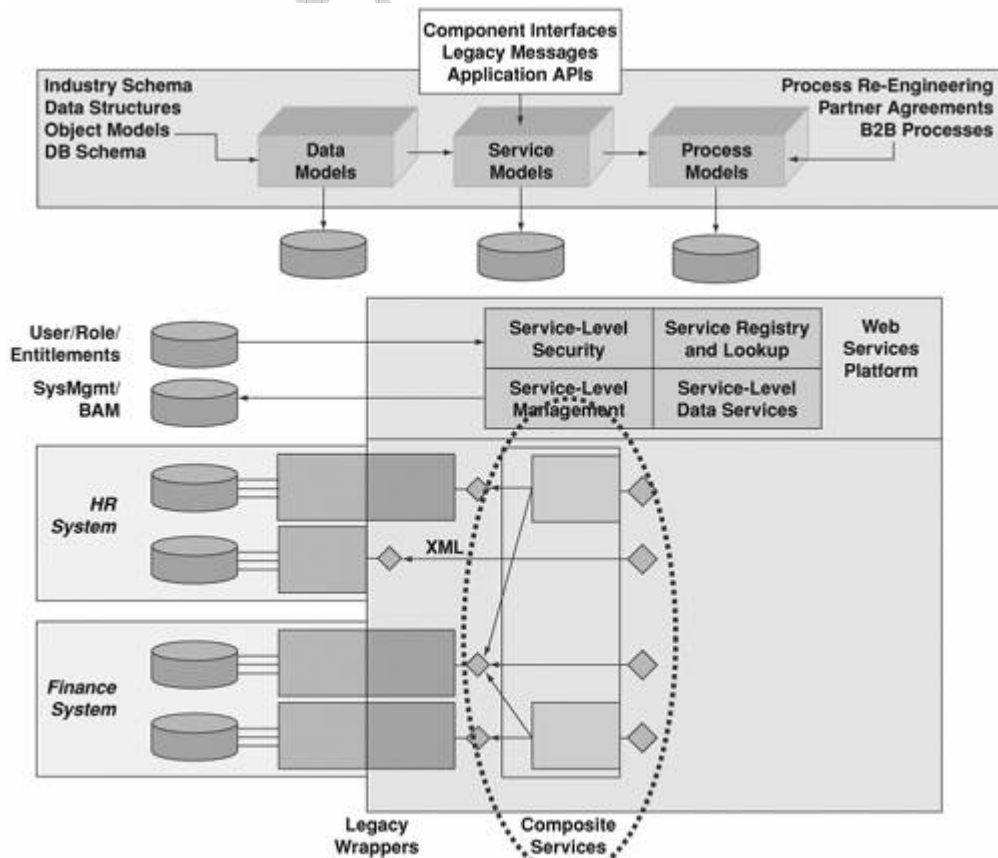
Hình 6-9 – Xây dựng các service đơn vị

Ngoài các dịch vụ cơ sở, ta còn phải xây dựng nền tảng khung Web service để cung cấp các tính năng cơ bản như định nghĩa, đăng ký, bảo mật và quản lý các dịch vụ cơ sở. Nền tảng khung Web service sẽ sử dụng các tính năng bảo mật và quản lý của các hệ thống cũ.

6.2.2.4 Xây dựng các dịch vụ tích hợp

Các dịch vụ tích hợp là các dịch vụ mà sử dụng đến các dịch vụ khác, một cách nói khác là được tạo thành bằng cách liên kết các dịch vụ với nhau. Dịch vụ tích hợp cũng giống như một web service bình thường: có một đặc tả dịch vụ WSDL và được gọi bằng các thông điệp SOAP.

Dịch vụ tích hợp có thể được tạo ra bằng cách lập trình (một EJB có thể thể hiện như một web service, trong đó có gọi đến các web service khác) hay sử dụng kỹ thuật orchestration kết hợp với WS-BPEL.



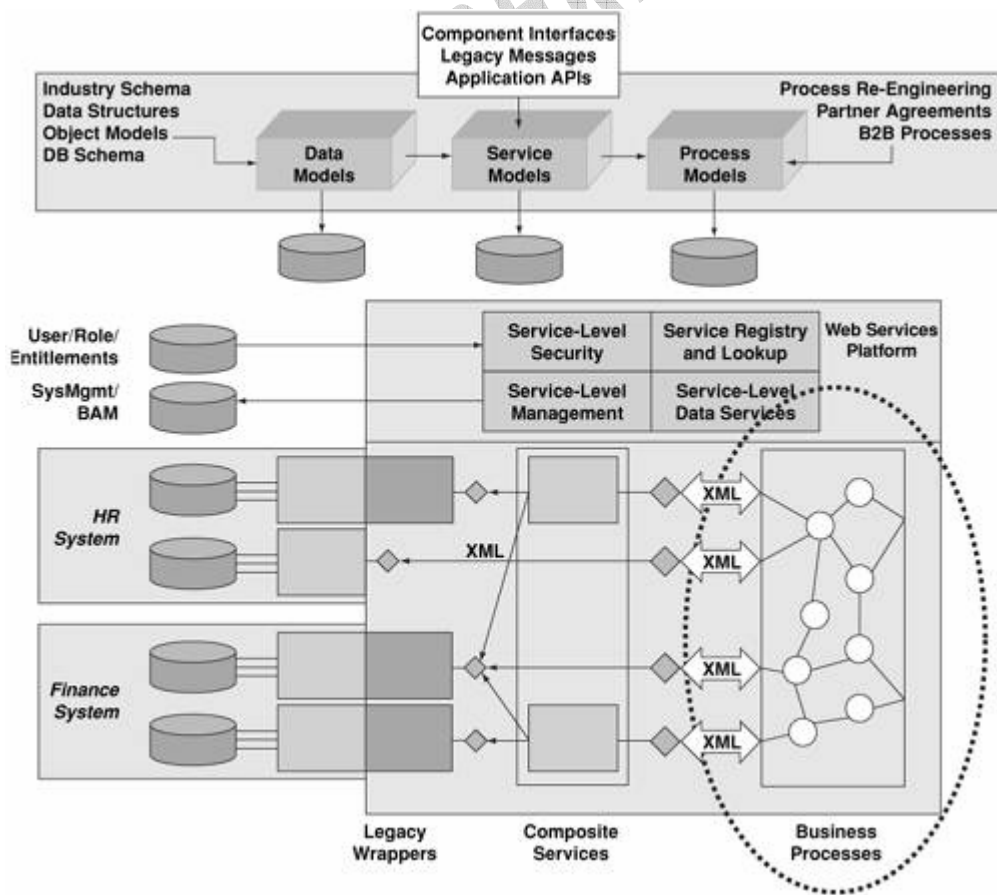
Hình 6-10 – Xây dựng các dịch vụ tích hợp.

6.2.2.5 Xây dựng các tiến trình nghiệp vụ

Một tiến trình nghiệp vụ thực hiện một chuỗi các tác vụ liên quan đến nhiều đối tượng (người dùng nội bộ, các đối tượng sử dụng tiến trình ở bên ngoài, và các dịch

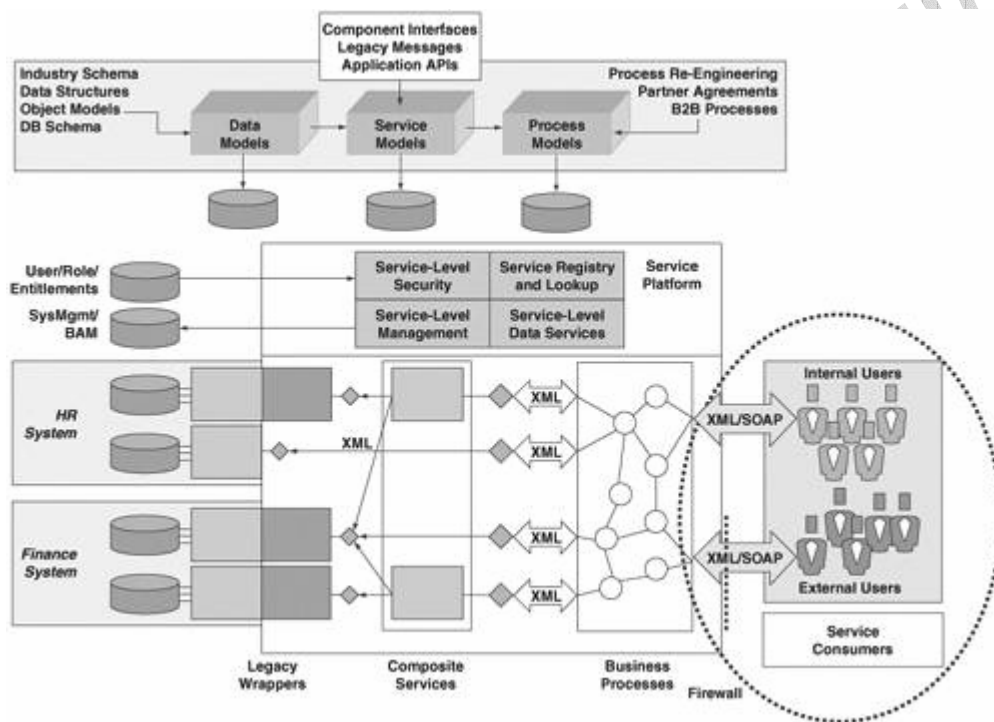
vụ liên quan). Quy trình xử lý của tiến trình nghiệp vụ được định nghĩa bằng ngôn ngữ đặc tả tiến trình WS-BPEL, và được thực thi bởi một engine.

Mỗi tác vụ trong tiến trình nghiệp vụ được thực hiện bởi một web service hay một người dùng. Khi tác vụ được thực hiện bởi web service thì engine sẽ xác định và gọi web service này, còn nếu tác vụ được thực hiện bởi một người dùng thì engine sẽ chuyển yêu cầu xử lý đến đối tượng đã được phân quyền.



Hình 6-11 – Xây dựng các tiến trình nghiệp vụ

6.2.2.6 Cung cấp các dịch vụ nghiệp vụ cho người dùng



Hình 6-12 – Cung cấp các tiến trình nghiệp vụ cho người dùng.

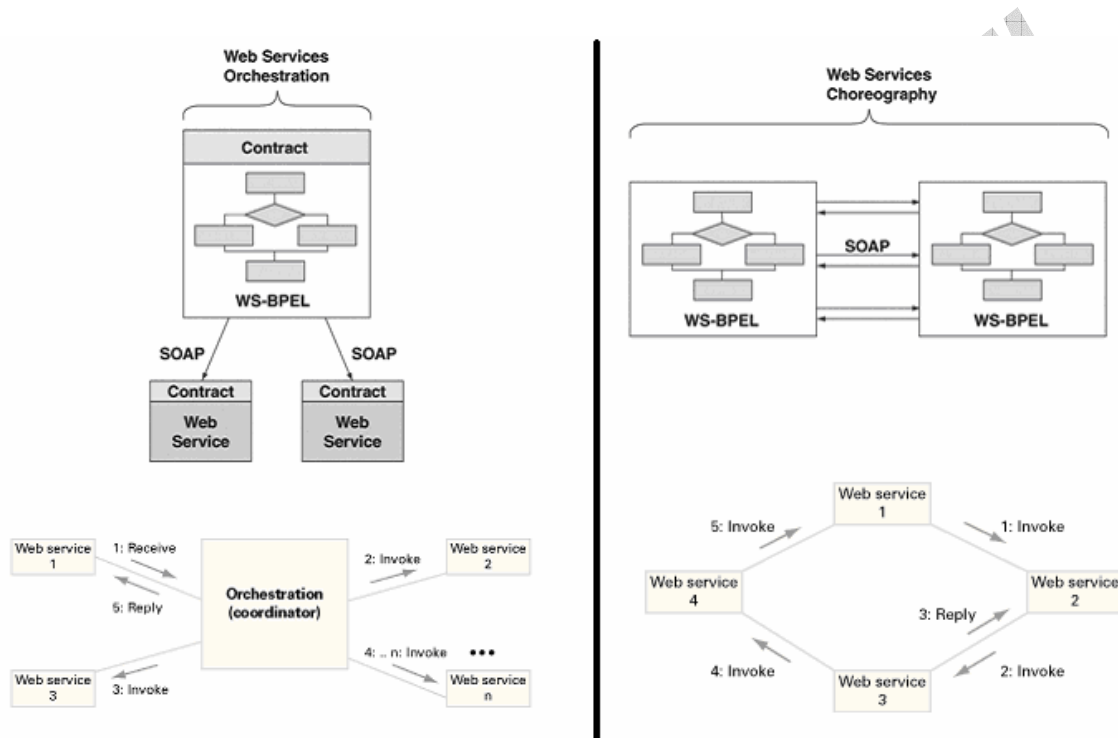
6.3 Thiết kế tiến trình

6.3.1 Orchestration và Choreography

Hai khái niệm orchestration và choreography thường được dùng để mô tả hai giải pháp kết hợp các web service. Mặc dù hai khái niệm có vẻ giống nhau, nhưng Web services orchestration (WSO) và Web services choreography (WSC) vẫn có sự khác biệt.

➤ Về ý nghĩa :

- WSO dùng để tạo ra các tiến trình nghiệp vụ (quan tâm đến qui trình thực hiện các thao tác, xử lý)
- WSC dùng để tạo ra các mô hình cộng tác nghiệp vụ (quan tâm đến sự cộng tác giữa các đối tượng).



Hình 6-13 – Sự khác nhau giữa orchestration và choreography.

➤ **Về mục đích :**

- WSO nhằm tạo ra các dịch vụ tích hợp bằng cách thực hiện liên kết các web service theo một trình tự xử lý (tuần tự, song song, rẽ nhánh...).
- WSC nhằm định nghĩa cách các đối tượng cộng tác với nhau như một phần của các giao tác. WSC cho phép các đối tượng mô tả vai trò của nó trong quá trình tương tác thông qua các thông điệp được trao đổi giữa các đối tượng web service.

➤ **Về mô hình:**

- WSO dựa trên mô hình requester/provider, trong đó định nghĩa dịch vụ nào sẽ được gọi, và được gọi khi nào. Các dịch vụ không cần biết thông tin về đối tượng gọi nó (ở đây là tiến trình) và khi nào thì nó được gọi.
- WSC dựa trên mô hình peer-to-peer, trong đó định nghĩa sự cộng tác giữa các đối tượng. Các đối tượng web service phải biết khi nào nó được gọi và được gọi bởi ai.

Ta nhận thấy rằng, giải pháp WSO sẽ có tính linh hoạt và mềm dẻo hơn so với giải pháp WSC, thể hiện ở :

- Quy trình xử lý của tiến trình được quản lý tập trung
- Không cần thực hiện chỉnh sửa, cập nhật các dịch vụ thành phần, vì chúng không cần biết thông tin về tiến trình gọi chúng.
- Dễ dàng thay đổi trình tự, kịch bản xử lý để phục vụ cho các yêu cầu khác nhau.

6.3.2 Các yêu cầu kỹ thuật khi thiết kế tiến trình

Trước khi giới thiệu về các chuẩn trong ngôn ngữ đặc tả, ta cần quan tâm đó là làm rõ một số yêu cầu kỹ thuật trong khi thiết kế các tiến trình. Vì đây sẽ là nền tảng cho việc thiết kế ngôn ngữ đặc tả, cũng như xây dựng các cơ chế xử lý ở tầng dưới của quá trình điều khiển orchestration và choreography.

6.3.2.1 Tính mềm dẻo

Một trong những yêu cầu quan trọng mà cần phải được quan tâm trước tiên đó là tính mềm dẻo, linh động của ngôn ngữ đặc tả. Điều này có thể đạt được bằng cách thực hiện tách biệt giữa phân mô tả quy trình xử lý và phân mô tả cách gọi thực hiện một dịch vụ. Như thế, khi có những thay đổi về xử lý nghiệp vụ thì ta có thể thực hiện thay đổi các dịch vụ mà không cần phải can thiệp vào chi tiết xử lý bên trong của tiến trình.

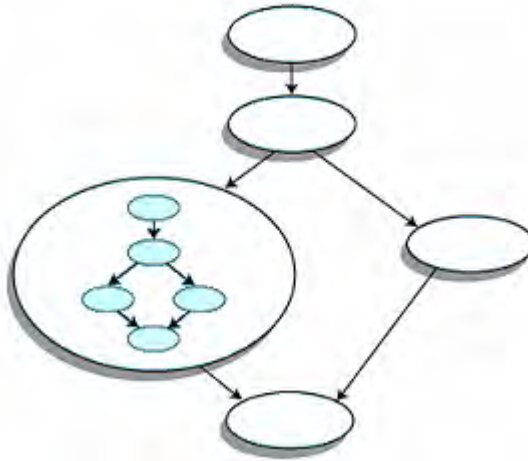
6.3.2.2 Các xử lý cơ bản và xử lý có cấu trúc

Ngôn ngữ đặc tả các tiến trình phải hỗ trợ đầy đủ các ngữ nghĩa xử lý, bao gồm việc truyền thông với các dịch vụ cũng như là điều khiển luồng xử lý của tiến trình.

Ta có thể hình dung các xử lý cơ bản là các xử lý hỗ trợ tương tác với tất cả những đối tượng ở bên ngoài tiến trình như dịch vụ, đối tượng sử dụng tiến trình... Còn các xử lý có cấu trúc sẽ hỗ trợ điều khiển luồng xử lý của tiến trình : xử lý nào sẽ được gọi, và được gọi khi nào.

6.3.2.3 Tiến trình phải có khả năng tái sử dụng

Điều này có nghĩa là cho phép định nghĩa các tiến trình mới từ những tiến trình hiện có. Các tiến trình ngoài việc sử dụng các chức năng của dịch vụ bên ngoài trong quá trình thực thi, thì còn phải cung cấp các chức năng của mình ra cho các đối tượng bên ngoài có thể sử dụng, nói một cách khác bản thân tiến trình cũng phải là một dịch vụ.



Hình 6-14 – Tiến trình cung cấp khả năng tái sử dụng.

6.3.2.4 Tiến trình phải có khả năng lưu trạng thái

Khả năng lưu trữ được trạng thái khi phải xử lý nhiều yêu cầu là một yêu cầu quan trọng mà tiến trình cũng phải hỗ trợ. Điều này thật sự cần thiết đối với các tiến trình mà quá trình thực thi có thể kéo dài. Khi đó tương tác giữa các đối tượng và tiến trình là một tương tác phức tạp, kéo dài. Tiến trình phải xử lý vấn đề bất đồng bộ, cũng như là điều phối thông tin trao đổi sao cho không làm lẫn lộn giữa các đối tượng.

6.3.2.5 Khả năng xử lý lỗi và quản lý giao tác

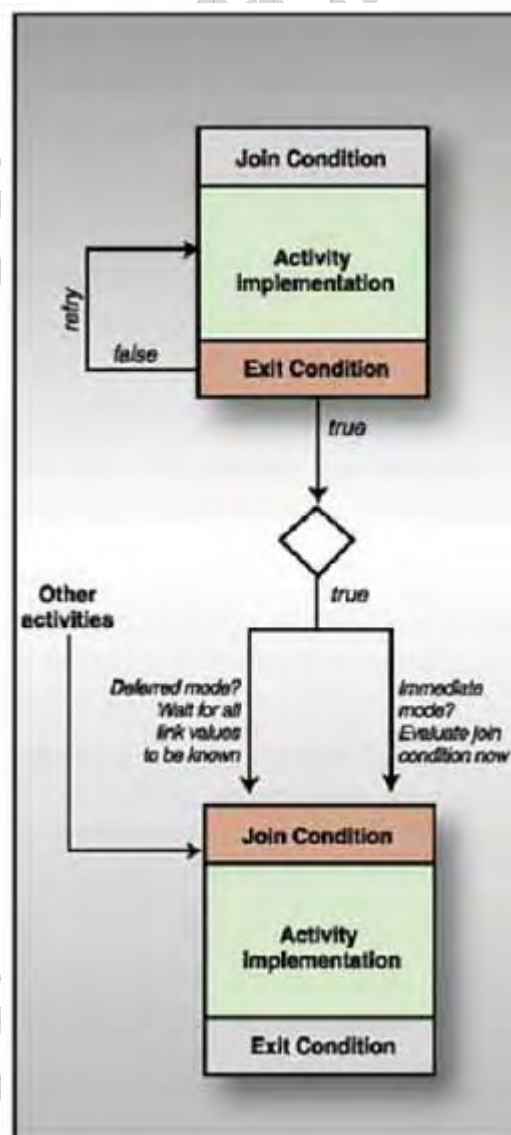
Đối với những tiến trình có quá trình thực thi kéo dài thì yêu cầu này cũng thật sự quan trọng. Điều này đảm bảo tính ổn định, cũng như là độ an toàn cho tiến trình trong quá trình thực thi.

6.3.3 Giới thiệu một số ngôn ngữ đặc tả tiến trình

6.3.3.1 Web Service Flow Language (WSFL)

WSFL là ngôn ngữ dùng để định nghĩa các tiến trình nghiệp vụ từ các web service. Những tiến trình được định nghĩa bằng WSFL, sau đó có thể được dùng như những web service. Điều này cho phép tích hợp nhiều tiến trình để tạo thành các tiến trình tích hợp có tính chất coarse-grained.

WSFL đưa ra giải pháp để tách biệt phần mô tả qui trình các luồng xử lý và phần chi tiết thực thi các thành phần xử lý bên dưới. Điều này cho phép tách biệt sự ràng buộc về mặt kỹ thuật và chuyên môn nghiệp vụ. Các nhà quản lý có thể tạo ra những tiến trình mà không cần các kiến thức về kỹ thuật, sau đó các tác vụ trong tiến trình sẽ được ánh xạ đến các dịch vụ thực thi. Đối với các nhà phát triển, họ chỉ cần tập trung vào việc thiết kế các chức năng xử lý, mà không cần phải quan tâm đến việc phải liên kết chúng lại như thế nào.

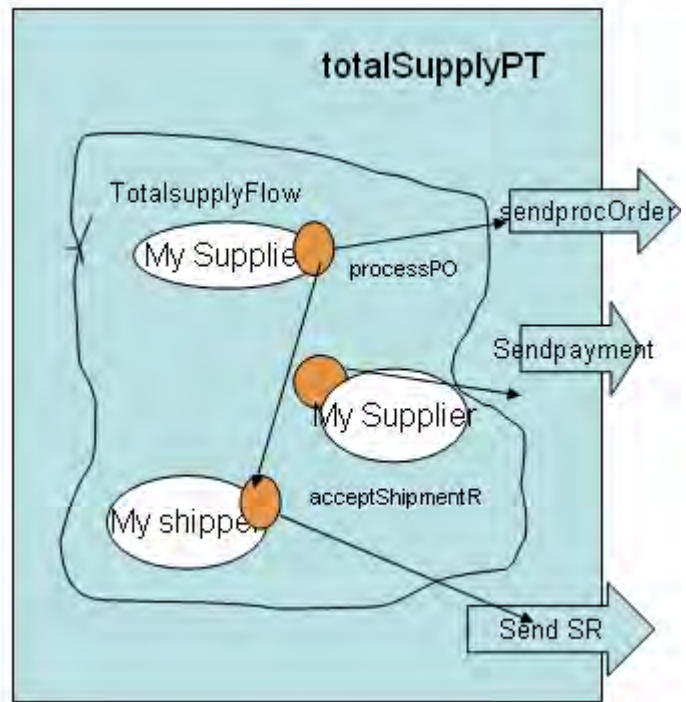


Hình 6-15 – Tiến trình được định nghĩa bằng WSFL

WSFL đưa ra khái niệm hai loại đối tượng sau:

- **Sơ đồ luồng**

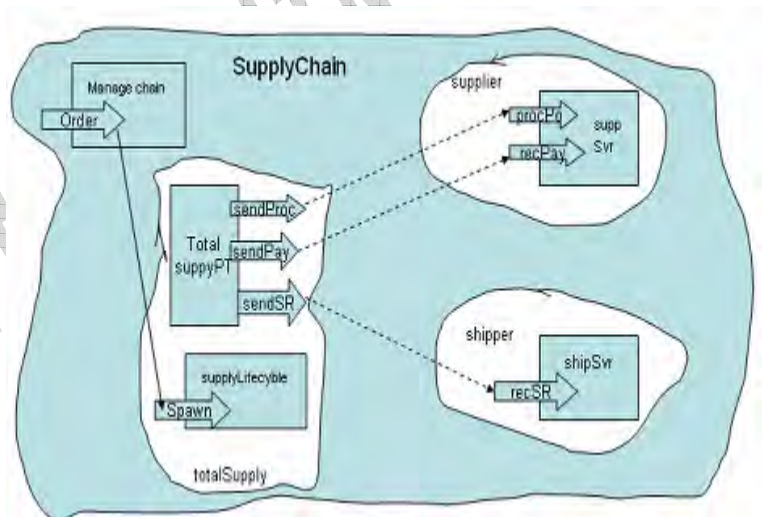
Sơ đồ này dùng để mô tả qui trình xử lý của tiến trình, bao gồm các xử lý, các thông điệp, các liên kết điều khiển, và các liên kết dữ liệu.



Hình 6-16 – Sơ đồ luồng trong WSFL

- **Sơ đồ tổng thể**

Sơ đồ này sẽ kết nối các hành động và thông điệp định nghĩa trong sơ đồ luồng với các dịch vụ cần thực thi. Ngoài ra, sơ đồ tổng thể cũng sẽ định nghĩa luôn phần giao tiếp của toàn bộ tiến trình. Thông tin này sẽ cần được dùng đến khi tiến trình được dùng lại như một dịch vụ.

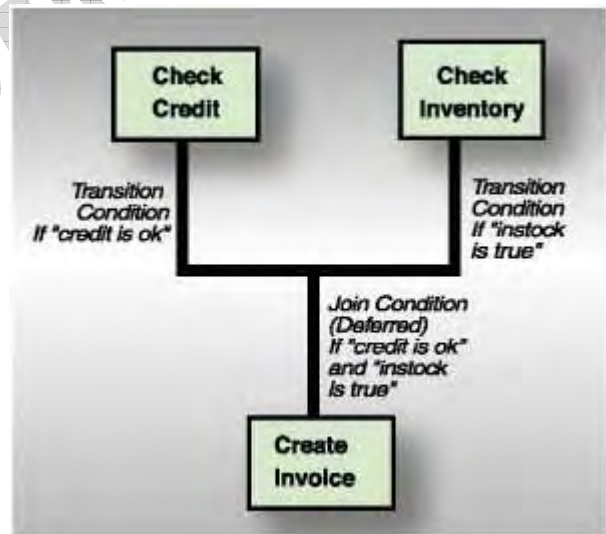


Hình 6-17 – Sơ đồ tổng thể trong WSFL

Một số khái niệm trong WSFL:

- **Xử lý:** Đây là hành động gọi đến một web service, và thật sự được ánh xạ đến các phương thức của các web service đó. Trong sơ đồ luồng, khi định nghĩa các xử lý, ta sẽ định nghĩa thêm thông tin về thông điệp vào, thông điệp ra, và thông điệp lỗi phát sinh của xử lý đó.
- **Liên kết điều khiển:**

Các xử lý được liên kết với nhau thông qua các liên kết điều khiển. Số lượng liên kết không giới hạn, nhưng một liên kết chỉ dùng để ràng buộc giữa hai xử lý.



Hình 6-18 – Liên kết giữa các xử lý trong WSFL

- **Liên kết dữ liệu:** Một liên kết dữ liệu sẽ chỉ ra luồng dữ liệu di chuyển từ một xử lý này đến một xử lý khác, bao gồm thông điệp vào, thông điệp ra.

6.3.3.2 XLang

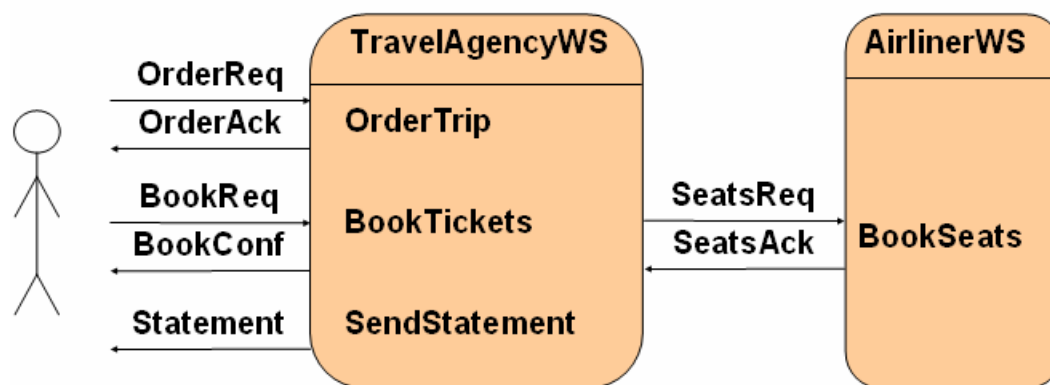
XLang là ngôn ngữ đặc tả tiến trình nghiệp vụ tựa XML, được sử dụng bởi Microsoft Biztalk Server. XLang được dùng để đặc tả cho các tiến trình mà sẽ được thực thi trên Biztalk Orchestration engine.

Một đặc điểm nổi bật của XLang là có hỗ trợ giao tác, nghĩa là tiến trình sẽ lưu lại thông tin trạng thái trong quá trình thực thi, và có thể ‘quay lui’ lại các xử lý đã được thực hiện trước đó khi có sự cố.

6.3.3.3 Web Service Choreography Interface (WSCI)

WSCI là ngôn ngữ dựa XML, được xây dựng nhằm mục đích mô tả quá trình tương tác của một dịch vụ, cụ thể là các quá trình vận chuyển các luồng thông điệp, trong bối cảnh của một tiến trình xử lý.

WSCI có thể xem như là phần mở rộng của ngôn ngữ WSDL (Web Service Description Language). Nếu như, ngôn ngữ WSDL chỉ dừng ở việc mô tả những thông tin “tĩnh” của một web service (tên phương thức, loại thông điệp trao đổi) thì WSCI đã định nghĩa thêm những khái niệm mới để mô tả cho việc kết hợp các phương thức, những quy định khi gọi thực hiện các phương thức, cũng như là điều khiển luồng trao đổi thông điệp, cách xử lý lỗi trong quá trình tương tác.

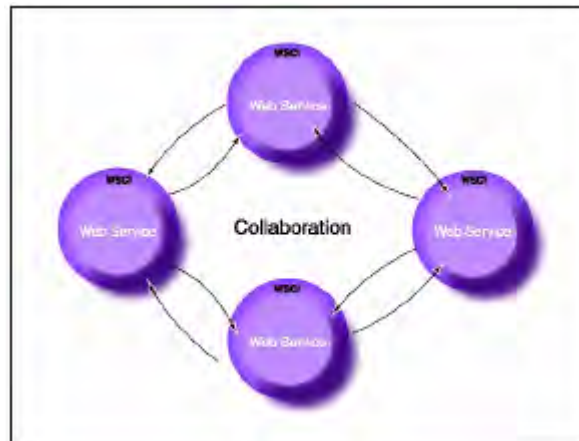


Hình 6-19 – Một ví dụ về các luồng thông điệp trong tương tác giữa các service

WSCI là ngôn ngữ hỗ trợ đặc tả các tiến trình theo phương pháp choreography. Nó mô tả, theo vết các thông điệp được trao đổi giữa các dịch vụ tham gia.

Một đặc trưng của WSCI là nó chỉ mô tả những thành phần có thể nhìn thấy được trong quá trình tương tác, như là các thông điệp. Và nó không quan tâm đến việc định nghĩa tiến trình đang được thực thi, hay nói đúng hơn là nó không định nghĩa một cách tổng thể toàn bộ quy trình xử lý, mà nó sẽ sử dụng đến sự hỗ trợ của một ngôn ngữ khác đó là Business Process Management Language (BPML) để làm việc này.

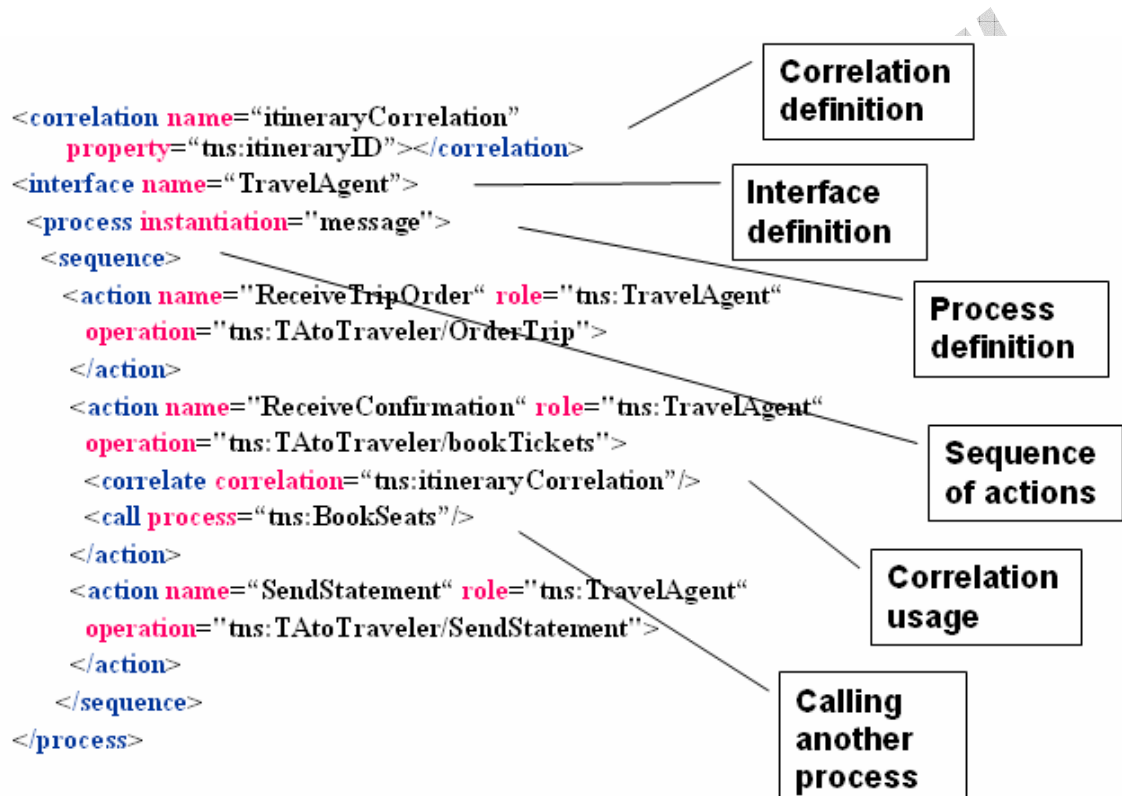
Nếu giữa hai dịch vụ có sự tương tác với nhau thì WSCI sẽ định nghĩa hai thành phần giao tiếp tương ứng để hỗ trợ trong quá trình trao đổi thông điệp. Theo minh họa trong Hình 6-20 ta thấy rằng, khi số lượng dịch vụ tương tác với nhau lớn, thì số thành phần giao tiếp được định nghĩa sẽ tăng thêm rất nhiều.



Hình 6-20 – Minh họa Web Service Choreography Interface

Một số khái niệm trong WSCI:

- Thành phần giao tiếp:
 - ▶ Thành phần giao tiếp sẽ định nghĩa những mối liên kết, những qui cách trao đổi thông điệp trong quá trình tương tác của các web service.
- Xử lý:
 - ▶ Xử lý cơ sở: mô tả các tác vụ cơ bản của một tiến trình: như là gửi thông điệp, nhận thông điệp, hay là chờ trong một khoảng thời gian...
 - ▶ Xử lý phức tạp: đây là các tác vụ phức tạp, hay còn gọi là 'có cấu trúc', vì nó sẽ chứa các xử lý cơ sở.
- Thuộc tính:
 - ▶ Đây là cách dùng để tham chiếu đến một yếu tố trong thành phần giao tiếp.
- Ngữ cảnh:
 - ▶ Mô tả môi trường mà trong đó các xử lý sẽ được thực thi.
- Lưu vết thông điệp:
 - ▶ Cơ chế này đảm bảo cho việc trao đổi dữ liệu một cách đúng đắn. Vì tại một thời điểm, hai dịch vụ có thể thực hiện nhiều tương tác, như vậy cần đảm bảo thông điệp được gửi/nhận trong đúng bối cảnh của nó.



Hình 6-21 – Một tiến trình được mô tả bằng WSCI

6.3.3.4 Business Process Execution Language For Web Service (BPEL4WS)

Việc kết hợp một cách có hiệu quả các dịch vụ hỗ trợ rất nhiều trong việc tích hợp các hệ thống. Điều này thật sự cần thiết trong bối cảnh phát triển của cộng đồng công nghệ thông tin ngày nay, khi mà xuất hiện ngày càng nhiều các nền tảng, các công nghệ mới. Và vấn đề mở rộng các hệ thống hiện có, tích hợp thêm các hệ thống mới để tiếp cận các lợi ích, các thành tựu của công nghệ mới đã trở nên là vấn đề cấp bách và hiện đang giành được rất nhiều sự quan tâm. Điều này thể hiện rõ ở sự ra đời của ngôn ngữ BPEL4WS (Business Process Execution Language For Web Service), với sự hỗ trợ phát triển của các công ty lớn như là Microsoft, IBM, Siebel Systems, BEA, và SAP. Và hiện đang trở thành một ngôn ngữ chuẩn trong việc đặc tả các tiến trình để tạo các dịch vụ tích hợp.

Tổng quan về ngôn ngữ BPEL4WS

BPEL4WS được xây dựng dựa trên ngôn ngữ WSFL (Web Service Flow Language) của IBM và ngôn ngữ XLANG của Microsoft. Vì thế nó kế thừa được những tính năng nổi trội của hai ngôn ngữ này (tính có cấu trúc của XLang và khả năng mô hình hóa của WSFL).

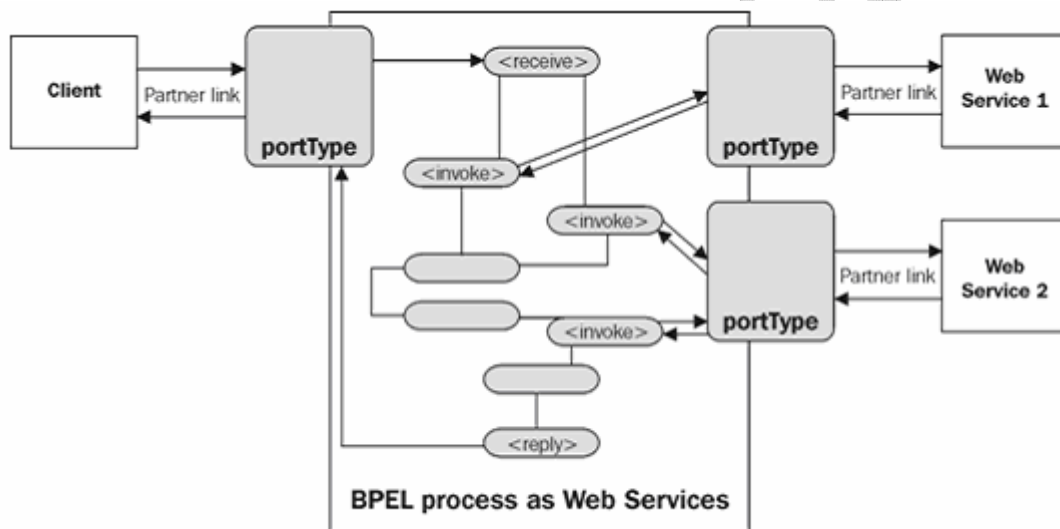
BPEL4WS hỗ trợ tạo ra hai loại tiến trình:

- **Tiến trình trừu tượng:** đưa ra những qui tắc trao đổi thông điệp giữa những dịch vụ tham gia, nhưng không chỉ rõ về cấu trúc bên trong của các thông điệp.
- **Tiến trình thực thi:** xác định rõ trình tự thực hiện của từng xử lý, các dịch vụ liên quan, các thông điệp trao đổi trong khi tương tác, cơ chế bắt lỗi và xử lý biệt lệ.

Đặc tả tiến trình của ngôn ngữ BPEL4WS có dạng sơ đồ luồng. Mỗi tác vụ trong tiến trình được gọi là một xử lý. Có hai loại xử lý:

- Các xử lý cơ bản:
 - ▶ <invoke> gọi thực hiện một phương thức của dịch.
 - ▶ <receive> chờ nhận một thông điệp từ một đối tượng bên ngoài tiến trình.
 - ▶ <reply> gửi thông điệp đến một đối tượng bên ngoài tiến trình.
 - ▶ <wait> dừng tiến trình để chờ trong một khoảng thời gian.
 - ▶ <assign> sao chép dữ liệu giữa các kho chứa dữ liệu.
 - ▶ <throw> thông báo lỗi trong quá trình xử lý.
 - ▶ <terminate> kết thúc tiến trình.
- Các xử lý có cấu trúc:
 - ▶ <sequence> điều khiển các xử lý bên trong thực hiện một cách tuần tự.
 - ▶ <flow> điều khiển các xử lý bên trong thực hiện một cách song song.
 - ▶ <while> lặp lại một xử lý trong khi điều kiện lặp còn được thỏa.
 - ▶ <switch> chọn lựa xử lý cần thực hiện dựa theo các điều kiện.

- ▶ <pick> chờ nghe sự kiện và thực hiện những xử lý tương ứng.
- ▶ <link> điều khiển trình tự thực hiện các xử lý trong khối <flow> (nếu có nhu cầu).



Hình 6-22 – Một tiến trình đặc tả bởi ngôn ngữ BPEL

Một số mẫu luồng xử lý của BPEL4WS

WP1: Sequence

- Mô tả: một xử lý được kích hoạt sau khi xử lý trước nó đã hoàn thành.
- Giải pháp: mẫu này đã được hỗ trợ sẵn trong xử lý có cấu trúc <sequence>

WP2: Parallel Split

- Mô tả: tại một thời điểm nào đó, một luồng xử lý chính được tách ra thành nhiều luồng khác nhau cùng thực hiện song song, như thế sẽ giúp cho các xử lý được thực thi song song và theo một thứ tự bất kỳ.
- Giải pháp: (xem giải pháp ở WP3)

WP3: Synchronization

- Mô tả: tại một thời điểm nào đó của tiến trình mà các luồng xử lý khác nhau cần tích hợp lại thành một luồng xử lý đơn. Vì thế, ta cần quan tâm đến việc đồng bộ giữa luồng.

- Giải pháp:
 - ▶ Mẫu WP2 được giải quyết bằng cách sử dụng xử lý <flow> để bao bọc các xử lý nào cần được thực hiện song song.
 - ▶ Nếu không định nghĩa các <link> trong xử lý <flow> thì các xử lý bên trong sẽ được thực thi cùng lúc.
 - ▶ Nếu thêm một xử lý B, thì ta sẽ có được WP3:
 - ▶ Để giải quyết vấn đề, ta có thể sử dụng các liên kết:
 - ▶ Các liên kết dùng để ràng buộc hai xử lý: xử lý nguồn và xử lý đích. Khi xử lý nguồn hoàn thành, thì xử lý đích mới được thực thi.
 - ▶ Nếu một xử lý là xử lý đích của nhiều liên kết thì có thể chỉ định là xử lý đó được kích hoạt khi một trong các xử lý nguồn hoàn thành (joinCondition='false' // mặc định) hay khi tất cả xử lý nguồn đều đã hoàn thành (joinCondition='true').

<pre> <sequence> <flow> activityA1 activityA2 </flow> activityB </sequence> </pre>	<pre> <flow name="F"> <links> <link name="L1"/> <link name="L2"/> </links> activityA1 <source linkName="L1"/>... activityA2 <source linkName="L2"/>... activityB <joinCondition="L1 AND L2"> <target linkName="L1"/> <target linkName="L2"/>... </joinCondition> </flow> </pre>
--	---

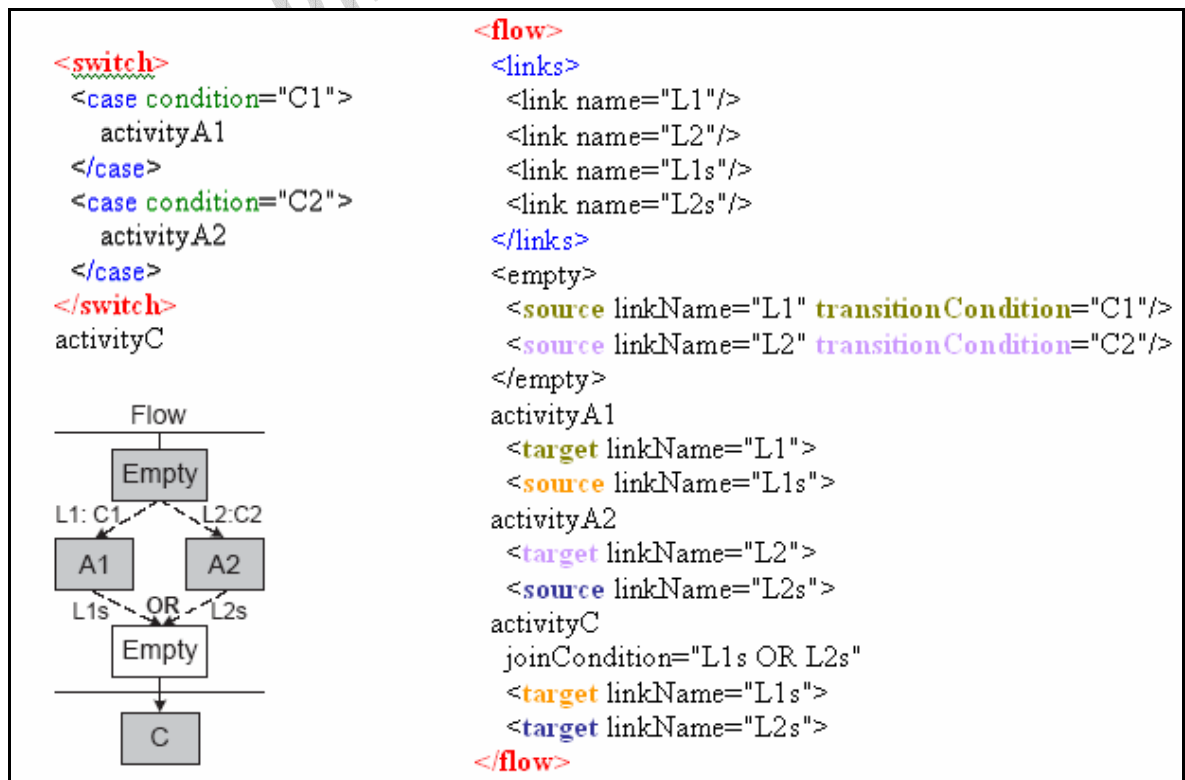
Hình 6-23 – Mẫu xử lý WP2 và WP3

WP4: Exclusive choice

- Mô tả: Tại một thời điểm thực thi tiến trình, tùy theo một điều kiện gì đó mà sẽ quyết định chọn một xử lý nào đó để thực thi.
- Giải pháp: (xem giải pháp của WP5)

WP5: Simple merge

- Mô tả: Tại một thời điểm thực thi, có thể có nhiều nhánh điều kiện được thỏa để kích hoạt một xử lý khác. Vậy thì làm sao giải quyết vấn đề đồng bộ để xử lý đó không bị kích hoạt nhiều lần.
- Giải pháp: cũng giống như ở WP2 và WP3, ta cũng có hai giải pháp
 - Dùng <switch>
 - Dùng <link> kết hợp với transitionCondition. Một <link> có thêm transitionCondition thì khi xử lý nguồn hoàn thành, và thì sẽ xét thêm transitionCondition. Nếu transitionCondition được thỏa, thì xử lý đích mới được kích hoạt.



Hình 6-24 – Mẫu xử lý WP4 và WP5

WP6: Multi-choice

- Mô tả: Tại một thời điểm thực thi, tùy theo một điều kiện nào đó mà một số xử lý sẽ được chọn và thực thi cùng lúc.
- Giải pháp: (xem ở giải pháp của WP7).

WP7: Synchronizing merge

- Mô tả: Tại một thời điểm thực thi, có nhiều nhánh được tích hợp thành một luồng đơn duy nhất. Một số trong các nhánh này đang được thực thi, trong khi một số khác thì không. Nếu chỉ một nhánh đang được thực thi, thì sau khi nhánh này hoàn thành thì xử lý ở bên dưới sẽ được kích hoạt. Thế nhưng, khi có nhiều nhánh đang thực thi thì vấn đề trở nên phức tạp hơn. Ta phải quan tâm đến việc đồng bộ giữa các nhánh này sao cho xử lý ở bên dưới chỉ được kích hoạt đúng một lần, nghĩa là sau khi tất cả các nhánh đang thực thi đều đã hoàn thành.
- Giải pháp: giống như giải pháp dùng <link> của WP4-5

WP8: Deferred Choice

- Mô tả: tại một thời điểm thực thi, một trong các nhánh sẽ được chọn dựa trên một thông tin nào đó, và thông tin đó chưa xác định tại thời điểm đó. Trường hợp này khác với exclusive choice (WP4) vì quyết định chọn lựa không được thực hiện ngay lập tức tại thời điểm đó, mà phải đợi cho đến khi thông tin cần có được xác định. Nói cách khác, quyết định lựa chọn được trì hoãn lại cho đến khi nào có sự xuất hiện của một biến cố nào đó.

```
<pick>
  <onMessage m1>
    <sequence>
      activity A1
      activity A2
    </sequence>
  </onMessage>
  <onMessage m2>
    <sequence>
      activity A2
      activity A1
    </sequence>
  </onMessage>
</pick>
```

Hình 6-25 – Mẫu xử lý WP8

- Giải pháp: Mẫu này đã được hỗ trợ bởi BPEL4WS thông qua xử lý <pick>.

Một số mẫu liên quan đến vấn đề giao tiếp

Giao tiếp đồng bộ

CP1: REQUEST/REPLY

- Mô tả: Đây là một dạng của giao tiếp đồng bộ, trong đó đối tượng gửi yêu cầu và đợi nhận trả lời trước khi tiếp tục xử lý. Nội dung trả lời có thể ảnh hưởng đến những xử lý thực hiện sau đó của phía gửi.
- Giải pháp: (xem giải pháp của CP2)

CP2: ONE-WAY:

- Mô tả: đây là hình thức giao tiếp đồng bộ mà người gửi sau khi gửi yêu cầu, sẽ chờ để nhận xác nhận từ phía bên nhận. Vì phía nhận chỉ gửi xác minh là đã nhận được yêu cầu nên nội dung trả lời coi như là trống và chỉ làm chậm đến những xử lý sau đó của phía gửi.
- Giải pháp: hình thức giao tiếp đồng bộ được hỗ trợ bởi BPEL4WS thông qua xử lý `<invoke>` hay cặp xử lý `<receive>` `<reply>`

<pre> <process name="processA"> <sequence> ... <invoke partner="processB" inputVariable="Request" outputVariable="Response"> </invoke> ... </sequence> </process> </pre>	<pre> <process name="processB"> ... <sequence> <receive partner="processA" ... container="Request"> </receive> ... <reply partner="processA" ... container="Response"> </reply> </sequence> </process> </pre>
--	---

Hình 6-26 – Mẫu giao tiếp CP1 và CP2

CP3: SYNCHRONOUS POLLING:

- Mô tả: đây cũng là một dạng của giao tiếp đồng bộ, trong đó, phía gửi sau khi gửi yêu cầu sẽ không dừng lại để chờ, mà sẽ tiếp tục xử lý. Sau một khoảng thời gian, phía gửi sẽ kiểm tra xem đã nhận được trả lời chưa? Khi đã nhận được trả lời, nó xử lý thông tin trả lời sau đó tiếp tục công việc của mình.

- Giải pháp: vấn đề này được giải quyết bằng cách thiết kế hai xử lý <flow>. Một <flow> để chờ nghe trả lời, và một <flow> để tiếp tục thực hiện chuỗi các công việc mà không bị lệ thuộc vào nội dung trả lời.

Giao tiếp bất đồng bộ (Asynchronous Communication)

CP4: MESSAGE PASSING:

- Mô tả: đây là một dạng của hình thức trao đổi bất đồng bộ, trong đó, phía gửi sẽ chuyển yêu cầu đến phía nhận, sau đó nó tiếp tục những công việc của mình.
- Giải pháp: vấn đề giải quyết tương tự như ở CP3, <invoke> không có outputVariable.

```
<process name="A"
  <sequence>
    <invoke partner="processB" ... inputContainer="Request"...> </invoke>
    <flow>
      <sequence> ... </sequence>
      <receive partner="processB" ... container="Result" ...> </receive>
    </flow>
    access container "Result" ...
  </sequence>
</process>
```

Hình 6-27 – Mẫu giao tiếp CP4

Chương 7

ỨNG DỤNG “SOA SUITE”

✍ Chương 7 sẽ giới thiệu tổng quát về ứng dụng SOASuite. Trình bày về hai thành phần ServiceBus và BpelEngine. ServiceBus cung cấp môi trường quản lý các dịch vụ dựa trên cơ chế thông điệp và BpelEngine cung cấp môi trường triển khai và thực thi cho các tiến trình nghiệp vụ.

7.1 Giới thiệu

7.1.1 Ứng dụng “SOA Suite”

Trong một hệ thống SOA, các ứng dụng được hình thành từ nhu cầu **cần xây dựng các quy trình nghiệp vụ mới** từ các **quy trình hay tác vụ có sẵn** (trong nội bộ hay từ bên ngoài). Ví dụ như các ứng dụng sử dụng lại những dịch vụ xử lý thẻ tín dụng, nhận và xử lý yêu cầu thanh toán, xem và cập nhật bảng tồn kho... Vấn đề đặt ra là làm thế nào quản lý và kết hợp những dịch vụ độc lập này lại với nhau vào trong một tiến trình xử lý.

SOA Suite cung cấp một môi trường dùng để:

- Quản lý các dịch vụ một cách hiệu quả.
- Hỗ trợ quá trình thiết kế, phát triển, triển khai và quản lý các tiến trình từ các dịch vụ sẵn có từ môi trường bên ngoài hay bên trong hệ thống.

SOA Suite được xây dựng nhằm giải quyết các vấn đề này.

7.1.2 Các thành phần của SOA Suite

SOA Suite bao gồm ba thành phần chính :

- ServiceBus: cung cấp môi trường quản lý các dịch vụ nội bộ trong hệ thống.
- BpelEngine: cung cấp môi trường thực thi cho các tiến trình nghiệp vụ.
- BpelDesigner: cung cấp môi trường thiết kế các định nghĩa các tiến trình nghiệp vụ.



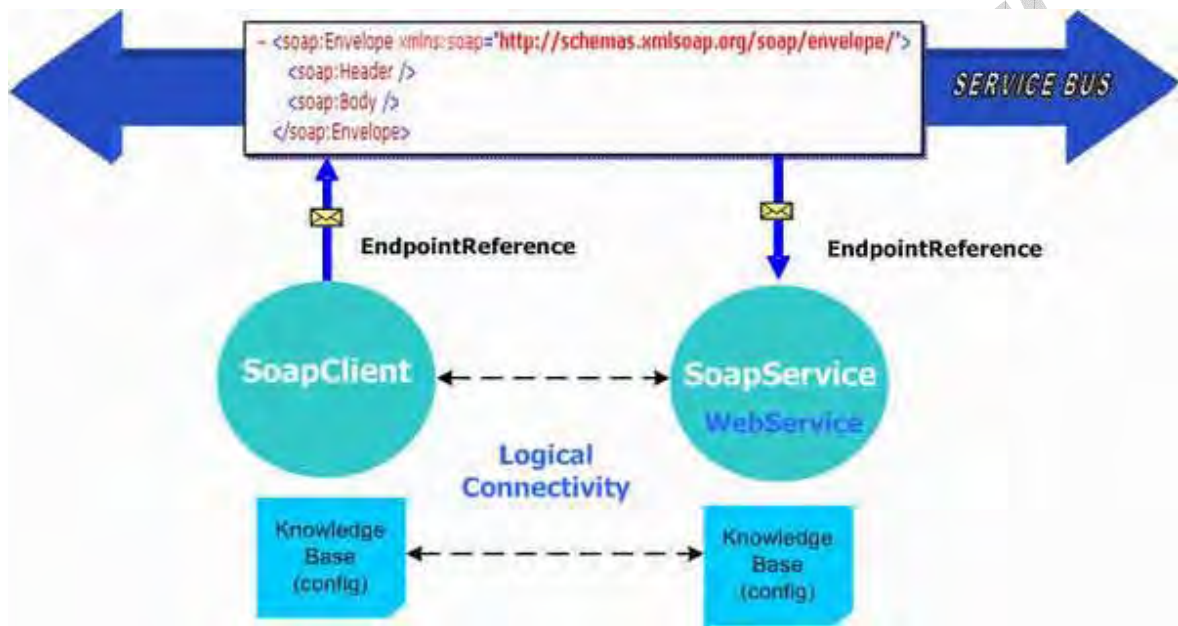
Hình 7-1 – Các thành phần của SOASuite

7.2 ServiceBus

7.2.1 Vai trò chức năng của ServiceBus

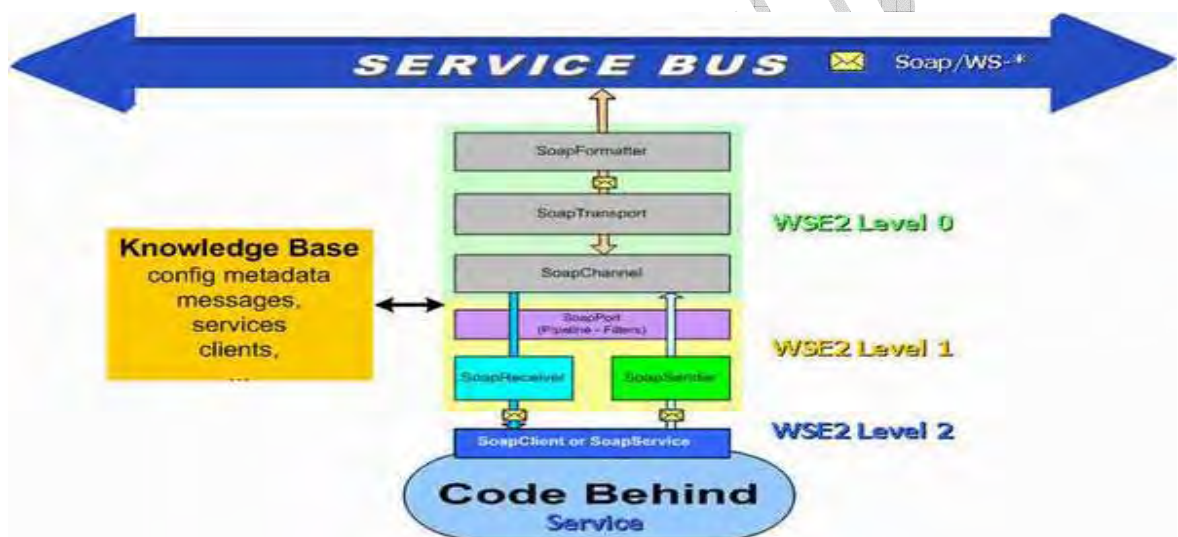
7.2.1.1 Vai trò của ServiceBus

ServiceBus được xây dựng nhằm cung cấp một môi trường kết nối logic giữa các dịch vụ và đối tượng sử dụng dịch vụ thông qua cơ chế truyền thông điệp. Môi trường giao tiếp giữa các dịch vụ này độc lập với xử lý bên trong và được xây dựng dựa trên “cơ sở tri thức liên kết” (Connectivity Knowledge Base - KB). Dữ liệu trong KB bao gồm các thông tin mô tả về các kết nối vật lý (physical connectivity) và cách thức xử lý các thông điệp. KB có thể tồn tại dưới dạng những kho lưu trữ ảo như các cơ sở dữ liệu, các tập tin cấu hình, các thông điệp...



Hình 7-2 – Môi trường trao đổi thông điệp SOAP của serviceBUS

Cơ chế truyền thông điệp của ServiceBus được xây dựng dựa trên sự hỗ trợ của bộ thư viện lập trình WSE 2.0 với các tính năng liên quan đến WS-Messaging. Kiến trúc 3 tầng của WSE 2.0 Messaging cho phép ta xây dựng một kênh giao tiếp độc lập giữa các dịch vụ và đối tượng sử dụng. Kênh giao tiếp này sẽ thực hiện vận chuyển và xử lý các thông điệp SOAP.



Hình 7-3 – Liên kết giữa ServiceBus và WSE Messaging

7.2.1.2 Các tính năng của ServiceBus

ServiceBus có các tính năng sau:

- Độc lập với phần xử lý của các dịch vụ: Tính năng này giúp ta có thể xây dựng các hệ thống từ nhiều nguồn khác nhau mà không quan tâm đến phần xử lý bên trong được xây dựng dựa trên ngôn ngữ hay hệ nền nào. Điều này giúp cho hệ thống ta có tính liên kết cao và khả năng dễ mở rộng.
- Liên kết dạng loose coupling với các KB: Tính năng này thực sự cần thiết. Trong môi trường ngày nay mật độ xảy ra các thay đổi là rất cao, dẫn đến các KB (hệ thống tin tri thức) sẽ thay đổi theo. Do đó, nếu hệ thống của ta càng chịu ít ràng buộc vào KB thì sẽ càng ít chịu ảnh hưởng bởi những sự thay đổi đó.
- Cung cấp một dịch vụ boot để thiết lập trạng thái ban đầu cho hệ thống bus: Cơ chế hoạt động của dịch vụ boot này ta có thể tùy biến một cách linh hoạt thông qua chỉnh sửa KB của nó. Với sự hỗ trợ của dịch vụ boot, ta có thể thay đổi trạng thái khởi động ban đầu của service bus khi cần thiết, bao gồm một số hoạt động liên quan đến việc nạp và khởi động các thành phần khác trong service bus.
- Hỗ trợ một số bộ lọc chuẩn : Cơ chế bộ lọc là một đặc điểm nổi bật của WSE Messaging. Với cơ chế này ta có thể thực hiện tùy biến một cách linh hoạt cho cách thức xử lý của các dịch vụ. Đồng thời, các bộ lọc này cũng có nhiều ý nghĩa trong ý tưởng về tái sử dụng các chức năng dùng chung.
- Cho phép quản lý cơ chế hoạt động của bus thông qua các KB : Điều này cũng có nghĩa là cơ chế hoạt động của service bus sẽ được điều khiển thông qua nội dung của KB. Như vậy, hệ thống của ta sẽ linh hoạt hơn, ổn định hơn trong việc đáp ứng những yêu cầu về thay đổi.
- Hỗ trợ tích hợp với IIS : Các dịch vụ không chỉ được dùng trong môi trường cục bộ, mà có thể sẽ có nhu cầu cung cấp các chức năng của các dịch vụ ra bên ngoài. Khi đó, với tính năng tích hợp vào IIS được xây dựng sẵn, service bus có thể tiếp nhận và đáp ứng với các yêu cầu từ bên ngoài.

7.2.2 ServiceBus và cơ sở tri thức

Mỗi serviceBUS bao gồm một cơ sở tri thức (KB) - nhưng KB này có thể tham chiếu đến nhiều KB khác nữa. ServiceBus thực chất là một thư viện liên kết động (DLL), được nạp lên bởi một tiến trình. Cơ sở tri thức của serviceBUS sẽ được chứa trong tập tin cấu hình của tiến trình đó.

```
<configuration>
  <configSections>
    <section name="rk.ConnectedSystem"

type="SOASuite.ServiceBus.Configuration.ServiceBusConfigurationHandler,
    SOASuite.ServiceBus, Version=1.0.0.0, Culture=neutral,
    PublicKeyToken=8f31f8b1a3da056f" />
  </configSections>

  <soa.ConnectedSystem >
    <ConfigurationHandlers>
      <add name="References"

type="SOASuite.ServiceBus.Configuration.ReferencesConfiguration,
    SOASuite.ServiceBus" />
      <add name="ServiceBusManager"
type="SOASuite.ServiceBus.Configuration.ServiceBusManagerConfig,
SOASuite.ServiceBus" />
      <!-- handlers -->
    </ConfigurationHandlers>

    <References>
      <!-- templates -->
    </References>

    <ServiceBus name="main">
      <!-- components such as services (managers), filters... -->
    </ServiceBus>
```

KB của serviceBUS được chứa trong tag <ConnectedSystem> và có thể coi đây như là entry-point của serviceBUS, trong đó có 3 thành phần con:

- ConfigurationHandlers:
 - ▶ Trong đó chứa thông tin về tất cả các bộ xử lý thông tin KB (kể thừa từ IConfigurationSectionHandler) của các thành phần khác như References, Services, Bootstraper... Các bộ xử lý thông tin này sẽ đọc KB (dữ liệu dạng XML) và thực hiện những khởi tạo ban đầu trong đó.

- Mỗi thành phần xử lý như thế được khai báo như sau:

```
<add name="ServiceBusManager"
type="SOASuite.ServiceBus.Configuration.ServiceBusManagerConfig,
SOASuite.ServiceBus" />
```

Trong đó, “name” là tên của KB, “type” là đường dẫn đến bộ xử lý KB (thực chất là một thư viện liên kết động - Dll).

- References

- Chứa tập hợp những mẫu khai báo của dịch vụ, thông điệp.. mà để sau này khi cần sử dụng đến cấu trúc dữ liệu đó thì không cần khai báo lại, chỉ cần thực hiện tham chiếu đến.

- ServiceBus

- Đây là nơi khai báo KB của các thành phần có trong serviceBUS (BootStraper, ServiceManager, các services, filters...). Với cơ chế này, ta có thể thực hiện bổ sung thêm các thành phần mới vào serviceBUS mà không cần phải viết code để xử lý chuyện đó. Ta chỉ cần bổ sung thêm KB của thành phần đó vào <ServiceBus> và một bộ xử lý tương ứng cho KB đó vào <ConfigurationHandlers>

7.2.3 Các thành phần của ServiceBus:

ServiceBus có các thành phần sau:

- **Dịch vụ**

- Các thành phần được gắn vào serviceBUS bản chất đều là các dịch vụ. Nhưng có thể do có những chức năng hay mục đích sử dụng khác nhau mà còn được gọi theo những tên gọi khác như là BootStrapper, ServiceManager...
- Để tạo các thành phần này, ta tạo một lớp kế thừa từ lớp Processor và sau đó biên dịch để tạo thành một thư viện liên kết động. (chi tiết về lớp Process này sẽ được mô tả chi tiết trong phần cơ chế hoạt động của serviceBUS).

- Các dịch vụ đều có chung một định dạng KB như sau:

```
<soa:Service name=" " mode="SingleCall" type=" " >
  <wsa:EndpointReference>
    <wsa:Address> </wsa:Address>
  </wsa:EndpointReference>

  <soa:InputFilters mode="on/off">
    <rksb:Filter name=" " type=" " ref=" " />
  </soa:InputFilters>

  <soa:OutputFilters mode="on/off">
    <rksb:Filter name=" " type=" " ref=" " />
  </soa:OutputFilters>

  <soa:SendMessages name=" " enable="true/false">
    <soa:SendMessage name=" " ref=" " enable="true"/>
  </soa:SendMessages>

  <soa:AddInfo ref=" " />
</soa:Service>
```

- Một dịch vụ được xác định bởi một tên (*name*); *mode* hiện tại chỉ hỗ trợ kiểu “SingleCall” (nghĩa là với mỗi yêu cầu sẽ tạo một thẻ hiện để giải quyết, do đó không thể lưu trạng thái.), và *type* chính là đường dẫn đến thư viện liên kết động của dịch vụ đó.
- <wsa:EndpointReference> cho biết thông tin địa chỉ của dịch vụ.
- <InputFilters>, <OutputFilters> sẽ chứa các <Filter>, và sẽ được sử dụng nếu mode=“on” và sẽ bị vô hiệu hóa nếu mode=“off”.
- <Filter> cho biết thông tin của bộ lọc, bao gồm tên, type chính là đường dẫn đến thư viện liên kết động của bộ lọc đó, ref là tên tham chiếu đến một bộ lọc đã định nghĩa trước đó trong phần References. Lưu ý, chỉ một trong hai thuộc tính *type* và *ref* được sử dụng.
- <AddInfo> sẽ chứa thêm các thông tin cần thiết khác.
- <SendMessages> sẽ liệt kê danh sách các <SendMessage> mà dịch vụ này hỗ trợ.
- <SendMessage> sẽ qui định cấu trúc của thông điệp trao đổi với dịch vụ, được xác định bởi một tên. Thuộc tính *ref* sẽ được sử dụng nếu <SendMessage> đó đã được định nghĩa trong phần References. Còn nếu <SendMessage> này chưa được định nghĩa trước đó, thì sẽ phải được định nghĩa theo cấu trúc sau:


```

<soa:SendMessage name=" ">
  <wsa:EndpointReference>
    <wsa:Address> </wsa:Address>
  </wsa:EndpointReference>

  <wsa:Action> </wsa:Action>

  <soa:MessageBody/>
</soa:SendMessage>

```

<SendMessage> sẽ phải chỉ ra địa chỉ được gửi đến thông qua <wsa:EndpointReference>, và phương thức yêu cầu thực hiện của dịch vụ <wsa:Action>. Ngoài ra, còn có bổ sung thêm các thông tin riêng trong phần <MessageBody>.

► Một ví dụ về KB của dịch vụ.

```

<soa:Service name="EchoService" mode="SingleCall" enablelog="true"
type="SOASuite.Test.EchoService, SOASuite.ServiceBus" >
  <wsa:EndpointReference>

    <wsa:Address>soap.tcp://localhost:8888/EchoService</wsa:Address>
  </wsa:EndpointReference>
  <soa:InputFilters mode="on">
    < soa:Filter name="OpenServices" mode="on" type="SOASuite.
Filters.OpenServicesInputFilter" />
  </soa:InputFilters>

  < soa:OutputFilters mode="on">
    <soa:Filter name="EchoServiceOutput" mode="on"
type="SOASuite.Filters.CustomerOutputFilter" />
  </soa:OutputFilters>

  <soa:SendMessages name="EchoService" enable="true">
    <rksb:SendMessage name="EnableLog">
      <wsa:EndpointReference>
        <wsa:Address>soap.tcp://localhost:8888/ServiceBusManager</wsa:Address>
        </wsa:EndpointReference>
        <wsa:Action>EnableLog</wsa:Action>
        <soa:MessageBody>
          <EnableLog>true</EnableLog>
        </soa:MessageBody>
      </rksb:SendMessage>

      <soa:SendMessage name="LoggerTo" ref="Logger" enable="true"/>
    </soa:SendMessages>

    <soa:AddInfo>
      <connectionString>server=abc </connectionString>
      <args arg1="12345" arg2="abcd"></args>
    </soa:AddInfo>
  </soa:Service>

```

- **Bootstrapper:**

- ▶ Bootstrapper là một thành phần lõi của serviceBUS.
- ▶ Cách thức xử lý của Bootstrapper sẽ được mô tả trong KB của nó.
- ▶ Tiến trình khi nạp serviceBUS sẽ thực hiện boot serviceBUS bằng cách gọi *ServiceBus.Boot()*
- ▶ Một ví dụ về KB của BootStrapper

```
<soa:Bootstrapper mode="on" enablelog="true">
  <wsa:EndpointReference>
    <wsa:Address>soap.tcp://localhost:911/Bootstrapper</wsa:Address>
  </wsa:EndpointReference>
  <soa:InputFilters mode="on">
    <soa:Filter name="OpenServices" mode="on"
      type="SOASuite.ServiceBus.Filters.OpenServicesInputFilter"/>
  </soa:InputFilters>
  <soa:OutputFilters mode="on">
    <soa:Filter name="NextTo" mode="on"
      type="SOASuite.ServiceBus.Filters.SendMessageOutputFilter" />
    <soa:Filter name="DisableLogger" mode="off"
      type="SOASuite.ServiceBus.Filters.SendMessageOutputFilter"/>
  </soa:OutputFilters>
  <soa:SendMessages name="Bootstrapper" enable="true">
    <soa:SendMessage name="BootMessage" ref="BootstrapMessage" />
  </soa:SendMessages>
  <soa:AddInfo/>
</soa:Bootstrapper>
```

- **Bus Manager:**

- ▶ Bus Manager cũng là một thành phần lõi của serviceBUS.
- ▶ Bus Manager là thành phần quản lý chính của bus, thực hiện các công việc như thêm một service vào bus, gỡ một service ra khỏi bus, quản lý phần Reference..
- ▶ Một ví dụ về KB của Bus Manager

```
<soa:ServiceBusManager mode="on" enablelog="false">
  <wsa:EndpointReference>
    <wsa:Address>soap.tcp://localhost:911/ServiceBusManager</wsa:Address>
  </wsa:EndpointReference>
  <soa:InputFilters mode="on">
    <soa:Filter name="Echo" mode="on"
      type="SOASuite.Filters.CustomerInputFilter"/>
  </soa:InputFilters>
  <soa:OutputFilters/>
  <soa:SendMessages name="ServiceBusManager" enable="true">
    <soa:SendMessage name="LoggerTo" ref="Logger"/>
  </soa:SendMessages>
  <soa:AddInfo/>
</soa:ServiceBusManager>
```

- **Filters:**

- ▶ Filter là một cách đóng gói một quá trình xử lý thông điệp thành một đối tượng chức năng có thể tái sử dụng được.
- ▶ Các thông điệp SoapEnvelope sẽ được xử lý bởi các bộ lọc theo trình tự được chỉ định trong các KB của dịch vụ.
- ▶ Các thông điệp SoapEnvelope gửi đến, trước khi đến dịch vụ sẽ được xử lý bởi các bộ lọc đầu vào (kế thừa SoapInputFilter). Và các thông điệp SoapEnvelope kết quả, trước khi được gửi trả về cho phía yêu cầu sẽ phải qua tầng xử lý của các bộ lọc đầu ra (kế thừa từ SoapInputFilter).

7.2.4 Cơ chế hoạt động của ServiceBus

7.2.4.1 Quá trình khởi động serviceBUS

Quá trình khởi động của bus được thực hiện bởi chức năng của dịch vụ BootStrapper cung cấp. Ứng dụng có thể gọi thực hiện quá trình này bằng cách gọi *ServiceBus.Boot()*, khi đó bus sẽ tiến hành các bước xử lý sau:

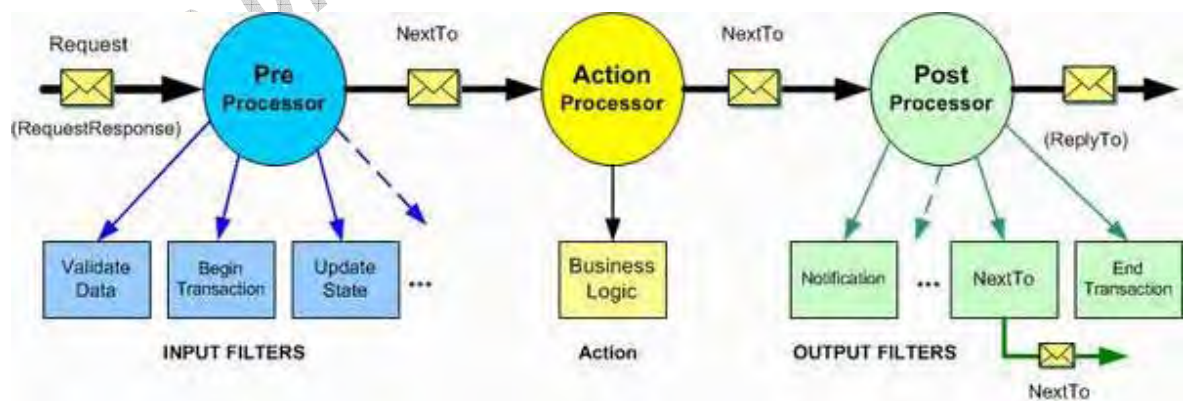
- Nạp thành phần References vào cache.
- Nạp tiếp những thành phần khác của bus thông qua các configuration handlers đã được khai báo.
- Gắn những dịch vụ lỗi vào bus (BootStrapper, ServiceBusManager)
- Đăng ký dịch vụ ServiceBusManager
- Gửi thông điệp BootMessage
- ▶ Một ví dụ của thông điệp BootMessage

```
<soa:SendMessage name="BootMessage" reqrsp="false" >
  <wsa:EndpointReference>
    <wsa:Address>soap.tcp://localhost:1234/Bootstrapper</wsa:Address>
  </wsa:EndpointReference>
  <wsa:Action>BootstrapMessage</wsa:Action>
  <soa:MessageBody name="BootMessageBody">
    <soa:OpenServices>
      <soa:Service name="EventSink" mode="SingleCall" />
    </soa:OpenServices>
  </soa:MessageBody>
</soa:SendMessage>
```

7.2.4.2 Quá trình xử lý một yêu cầu

Quá trình xử lý một thông điệp yêu cầu dịch vụ của serviceBUS được chia làm 3 giai đoạn chính:

- Ở giai đoạn đầu tiên, thông điệp được chuyển qua các bộ lọc đầu vào. Giai đoạn này còn được gọi là tiền xử lý.
- Sau đó, thông điệp được chuyển vào thành phần xử lý để thực hiện đáp ứng yêu cầu. Trong cơ chế của WSE thì trong giai đoạn này, phương thức *Receive* sẽ được gọi.
- Sau cùng, thông điệp được xử lý bởi các bộ lọc đầu ra trước khi được trả về cho đối tượng gọi hay được chuyển tiếp qua một dịch vụ khác.



Hình 7-4 – Qui trình xử lý thông điệp của serviceBUS.

- Đây có thể coi là quá trình xử lý yêu cầu của một dịch vụ. Và đối tượng *Processor* của serviceBUS đã được xây dựng để thực thi mô hình này. Do đó, mọi dịch vụ muốn được gắn vào bus thì nhất định phải kế thừa từ đối tượng *Processor* và phần xử lý chính của dịch vụ đó sẽ được thực hiện thông qua override phương thức *Receive* của đối tượng *Processor* (đây có thể coi như là một mẫu template.)

7.2.5 ServiceBus tích hợp với IIS

ServiceBus đã hỗ trợ tính năng tích hợp vào IIS. Có hai vị trí plug-in trong IIS Server mà ta cần xử lý, đó là HttpModule và HttpHandler. Việc plug-in serviceBUS vào IIS được thực hiện bằng cách cấu hình file *web.config* như sau:

```
<system.web>
  <httpModules>
    <add name="ServiceBus"
        type="SOASuite.ServiceBus.ServiceBusHostedByIIS,
RKiss.ServiceBus"/>
  </httpModules>
  <httpHandlers>
    <add verb="*" path="*.ashx"
        type="SOASuite.ServiceBus.SoapRequestDispatcher,
RKiss.ServiceBus"/>
  </httpHandlers>
</system.web>
```

- httpModule handler:
 - ▶ Sẽ thực hiện boot serviceBUS khi ứng dụng khởi động và giải phóng serviceBUS khi ứng dụng kết thúc.
- httpHandler handler:
 - ▶ Sẽ thực hiện chuyển yêu cầu http đến dịch vụ của bus với điều kiện lọc là `HttpContext.Request.Url` có phần mở rộng là “*ashx*”. Ví dụ như <http://localhost/MyApplication/EventSink.ashx>.

7.3 BpelEngine

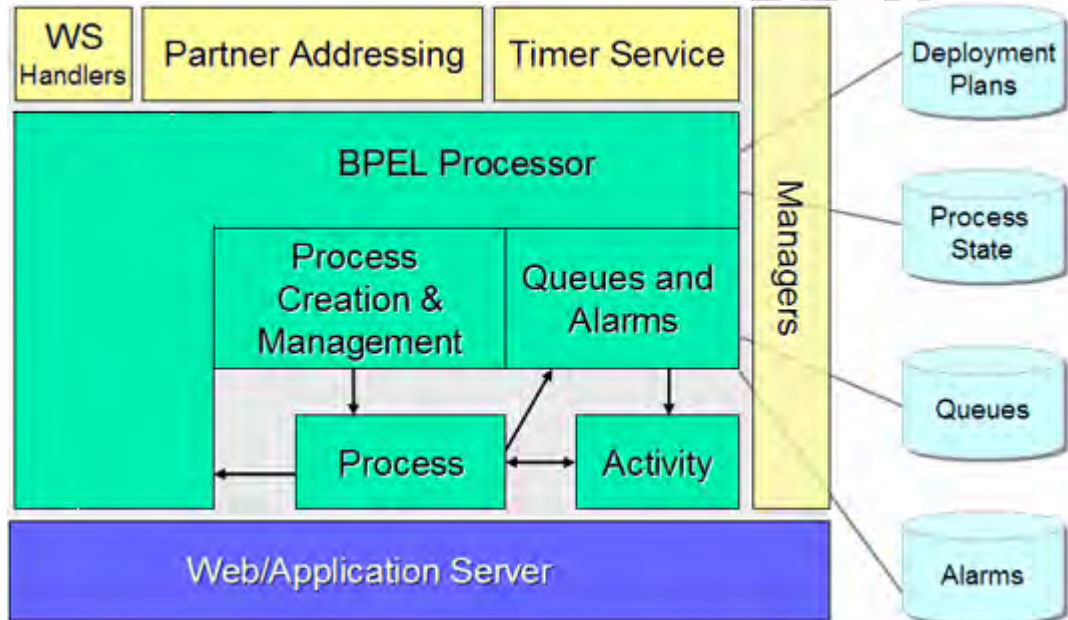
7.3.1 Kiến trúc của BpelEngine

BpelEngine cung cấp môi trường thực thi cho các tiến trình nghiệp vụ. BpelEngine nhận vào định nghĩa của một tiến trình và một số thông tin khác như các thông tin mô tả web service WSDL và tạo các thể hiện của các tiến trình này. Sau đó, với mỗi yêu cầu sử dụng tiến trình, nó sẽ tạo một thể hiện của tiến trình và thực thi thể hiện này.

Có 3 đối tượng mà ta cần quan tâm khi phân tích kiến trúc của BpelEngine, đó là: engine, các tiến trình và các xử lý. BpelEngine có thể thực thi một hay nhiều tiến trình. Các tiến trình lại bao gồm nhiều xử lý, và các xử lý này cũng có thể sẽ chứa bên trong nó các xử lý khác. BpelEngine tạo ra một tiến trình từ thông tin định nghĩa tiến trình đó (sử dụng ngôn ngữ đặc tả tiến trình BPEL) và sau đó thực thi tiến trình này.

7.3.1.1 Engine

Mô hình sau sẽ phát thảo tổng quát về kiến trúc của engine



Hình 7-5 – Sơ đồ kiến trúc của BpelEngine.

Các thành phần “dữ liệu” trong bộ nhớ như:

- Deployment plans: bao gồm tập hợp các thông tin mô tả cần thiết để triển khai một tiến trình (các tập tin mô tả XML, các tập tin wsdl...).
- Process state: lưu các trạng thái của tiến trình, có một số trạng thái như PROCESS_LOADED, PROCESS_RUNNING, PROCESS_SUSPENDED, PROCESS_COMPLETE, PROCESS_FAULT.
- Queues: hàng đợi để quản lý các đối tượng trong quá trình thực thi, như là đối tượng Alarm (đối tượng timer), Invoke (đối tượng gọi một dịch vụ), Reply (đối tượng phản hồi để trả kết quả xử lý về cho đối tượng gọi), InboundReceive (các đối tượng thông điệp được gửi đến tiến trình), MessageReceiver (các đối tượng chờ nhận kết quả yêu cầu.)
- Alarms: quản lý các đối tượng đang được chờ kích hoạt thực hiện bởi một biến cố thời gian (sau bao lâu, hay là cho đến một thời điểm nào đó...).

7.3.1.1.1 Khởi động engine (Engine Startup)

Một đối tượng EngineFactory sẽ quản lý việc tạo và khởi động một BpelEngine. Nhiệm vụ của BpelEngine rất phức tạp vì phải giám sát và thực hiện rất nhiều vấn đề. Và việc phân chia trách nhiệm trong BpelEngine được thực hiện qua việc sử dụng các đối tượng quản lý như là queue manager, alarm manager, timer manager, work manager, process manager... Đoạn mã giả sau sẽ cho thấy quá trình engine factory tạo một BpelEngine như thế nào:

```
// Initialize the work manager
initializeWorkManager();
// Initialize the timer manager
initializeTimerManager();
// create the managers
IProcessManager processManager = createProcessManager();
IQueueManager queueManager      = createQueueManager();
IAlarmManager alarmManager      = createAlarmManager();
ILockManager lockManager        = createLockManager();

sEngine=createNewEngine(queueManager,processManager, alarmManager,
lockManager);
```

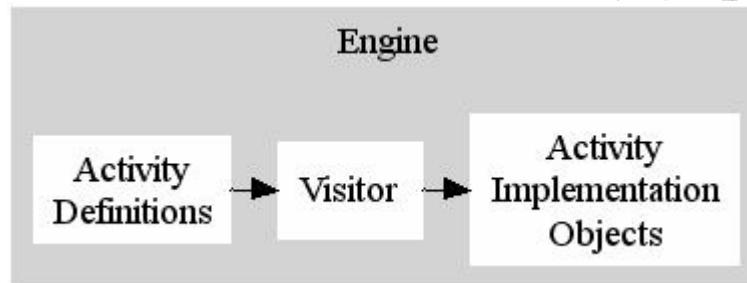
Thông tin cấu hình của engine được lưu trong một tập tin “EngineConfig.xml”. Tập tin này không chỉ chứa các thông số cấu hình như kích thước cache, giới hạn các thread,... mà còn lưu các thông tin về các thành phần thực thi của các factory, manager. Với cách thiết kế này, đã hỗ trợ cho engine có tính linh hoạt cao, dễ thay đổi khi có yêu cầu.

7.3.1.1.2 Tạo tiến trình

Một process deployment provider sẽ đọc các thông tin cần thiết để triển khai một tiến trình (Process Deployment Descriptor - .pdd) và một process deployment manager sẽ xử lý quá trình tạo tiến trình.

Khi một engine đọc một định nghĩa tiến trình, nó sẽ tạo các đối tượng gọi là activity definition. Đây là các đối tượng được xây dựng nhằm “mô hình hóa” tiến trình. Các đối tượng activity definition này chứa tất cả các thông tin cần thiết để khởi tạo một đối tượng thực thi tương ứng (activity implementation object). Ta có thể coi các

activity definition giống như các lớp, và các activity implementation như các đối tượng - thể hiện của lớp.



Hình 7-6 – Tạo các đối tượng activity implementation.

Engine tạo các đối tượng activity implementation bằng cách sử dụng mẫu Visitor để visit từng đối tượng activity definition và tạo các đối tượng activity implementation tương ứng.

7.3.1.1.3 Xử lý dữ liệu

Tất cả những đối tượng biến đều thực thi thành phần giao tiếp IVariable. Thành phần giao tiếp này cung cấp khả năng để lấy thông tin về kiểu và dữ liệu của biến đó. Một biến có thể có những kiểu sau: Xml schema simple type, Xml schema Element hay message type (được định nghĩa trong các tập tin WSDL).

7.3.1.1.4 Lượng giá các biểu thức

Tất cả các xử lý và liên kết đều cho phép sử dụng các biểu thức trong một số thuộc tính của nó. Những biểu thức này đòi hỏi phải có một phương pháp nhất quán để thực thi và mô tả ngữ cảnh thực thi, tức là phải hỗ trợ truy cập các giá trị biến cần thiết để thực hiện lượng giá biểu thức.

Ngôn ngữ mô tả các biểu thức này mặc định là XPath 1.0. Tuy nhiên ngoài các hàm xử lý chuẩn của XPath, các biểu thức được quyền sử dụng các hàm mở rộng được định nghĩa thêm (như là bpws:getVariableData...).

7.3.1.2 Tiến trình nghiệp vụ

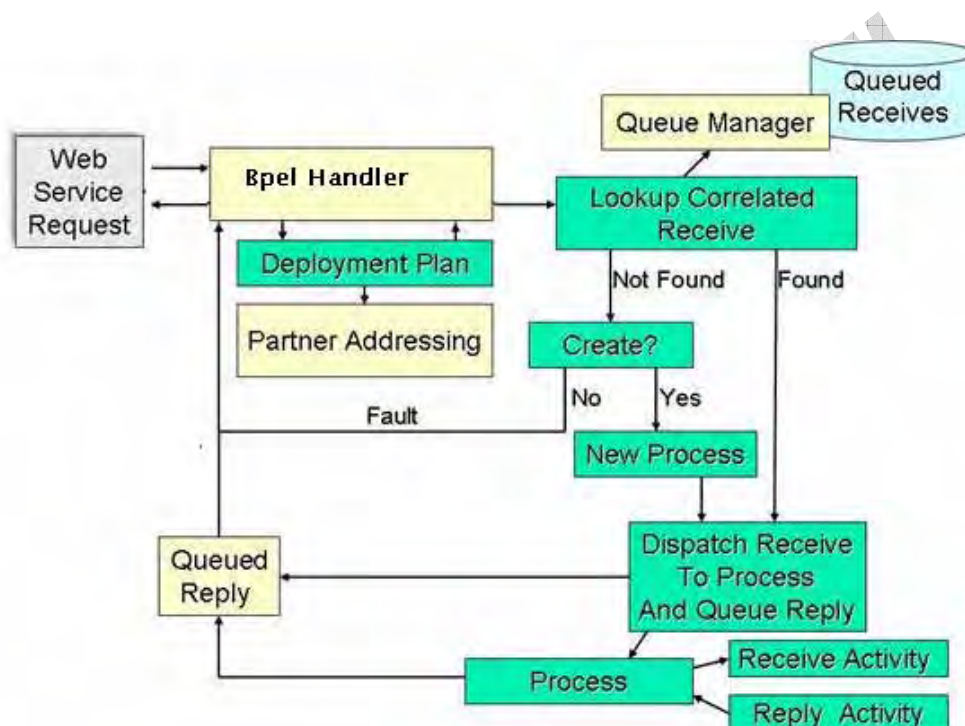
Một tiến trình nghiệp vụ bao gồm các đối tượng sau:

- Partner links: mô tả mối quan hệ giữa hai dịch vụ (bản thân tiến trình cũng có thể coi như là một dịch vụ).
- Partners: các thực thể tham gia vào tiến trình (các dịch vụ và đối tượng sử dụng tiến trình).
- Variables: là các đối tượng để lưu dữ liệu trong quá trình xử lý.
- Correlation sets: một tập các dữ liệu mà nhận diện duy nhất một thể hiện tiến trình. Tại các thời điểm khác nhau của tiến trình, các correlation sets khác nhau vẫn có thể dùng để đặc trưng cho cùng một tiến trình.
- Fault handlers: mô tả cách xử lý của tiến trình khi có lỗi xảy ra.
- Compensation handlers: mô tả cách quay ngược một xử lý đã được thực hiện và hoàn thành.
- Event handlers: xử lý khi có các thông điệp gửi đến hay khi có một biến cố thời gian.
- Activity: các xử lý, một xử lý có thể chứa các xử lý khác.

Điều phối yêu cầu

Mỗi tiến trình phải có ít nhất một xử lý kích hoạt. Một thể hiện của tiến trình được tạo ra khi một trong các xử lý kích hoạt được kích hoạt, hoặc là bởi các thông điệp được gửi đến, hoặc là bởi các biến cố thời gian (Alarm) của <pick>.

Engine sẽ chuyển các thông điệp gửi đến đúng thể hiện của tiến trình. Nếu có dữ liệu correlation, engine sẽ cố gắng tìm đúng thể hiện có dữ liệu correlation phù hợp và chuyển thông điệp cho thể hiện đó. Nếu thông điệp đó không có dữ liệu correlation và engine tìm thấy một xử lý kích hoạt thích hợp thì nó sẽ tạo thể hiện của tiến trình đó.



Hình 7-7 – Sơ đồ điều phối thông điệp của engine

Hàng đợi các đối tượng chờ nhận thông điệp

Hàng đợi này sẽ chứa các xử lý chờ nhận thông điệp hiện đang được thực thi của tất cả các thể hiện tiến trình. Các xử lý chờ nhận thông điệp này đều chứa một xử lý <onMessage> như một phần của <pick> hay của một trình xử lý sự kiện (event handler). Một hành động chờ nhận thông điệp được gọi là đang thực thi khi nó được “cha” của nó (scope, flow, sequence...) đưa vào hàng đợi nhưng vẫn chưa nhận được thông điệp mà nó cần.

Hàng đợi này cũng chứa những thông điệp từ bên ngoài gửi đến nhưng chưa tìm thấy một xử lý chờ nhận thông điệp thích hợp để chuyển đến. Các thông điệp này sẽ được xếp trong hàng đợi cho đến khi hết thời gian cho phép (time-out). Thời gian time-out này do engine qui định trong tập tin cấu hình (*UnmatchedReceiveTimeout*).

Nếu một tiến trình đưa vào hàng đợi một xử lý giống như là receive, thì xử lý này sẽ đợi cho đến khi dữ liệu thích hợp gửi đến hay cho đến khi tiến trình đó kết thúc (có thể là do lỗi hay do một xử lý <terminate>). Các xử lý <picks> thì hơi khác: <onMessage> hay <onAlarm> đầu tiên nào mà được kích hoạt thì ngay lập tức trạng

thái của các <onMessage> hay <onAlarm> khác đều được gán là DEAD_PATH. Hành động này sẽ xóa chúng ra khỏi hàng đợi. Các trình xử lý sự kiện cũng sẽ tự động xoá các đối tượng lắng nghe ra khỏi hàng đợi ngay khi scope mà chứa chúng hoàn thành.

7.3.1.3 Các xử lý

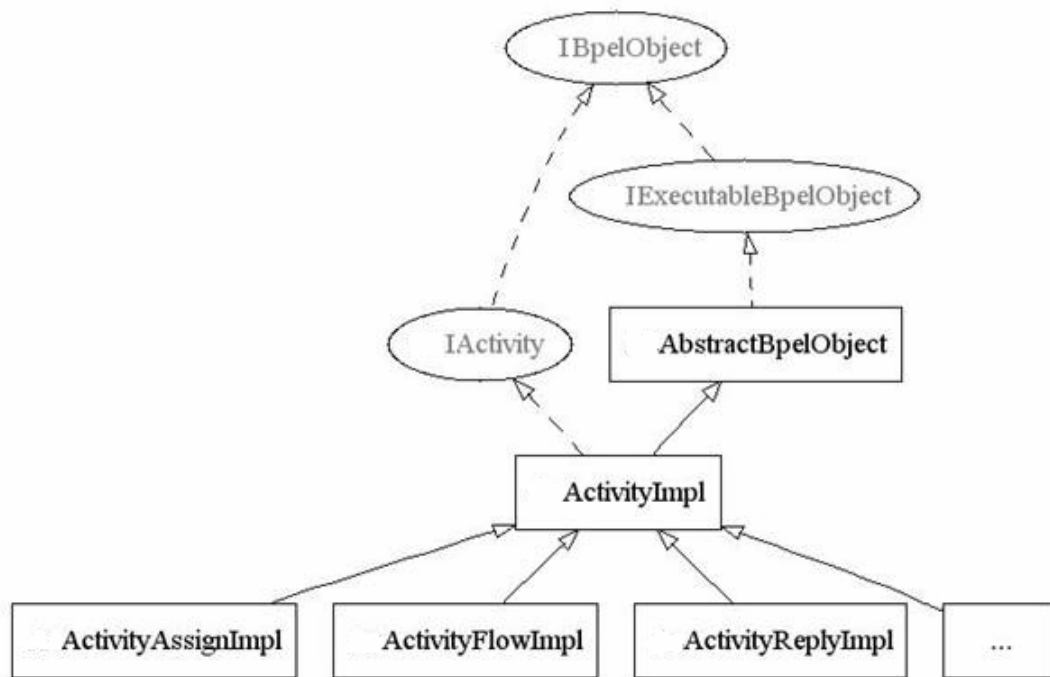
Bao gồm các xử lý cơ bản (receive, reply, invoke, assign, throw, wait, empty); các xử lý có cấu trúc (sequence, switch, while, pick, flow); và một số các xử lý đặc biệt khác (scope, compensate, terminate).

Trạng thái

Mỗi xử lý đều có trạng thái. Có các trạng thái sau:

- INACTIVE: đây là trạng thái đầu tiên của mọi xử lý khi tiến trình bắt đầu.
- READY_TO_EXECUTE: các xử lý này được đưa vào hàng đợi và đã đủ điều kiện thực thi.
- EXECUTING: đang thực thi.
- FINISHED: đã hoàn thành và không có lỗi.
- FAULTED: đã hoàn thành nhưng có lỗi
- DEAD_PATH: không được đưa vào “kế hoạch thực thi” (execution path). Khi một xử lý có trạng thái DEAD_PATH thì tất cả các xử lý con đều có trạng thái giống như thế.
- QUEUED_BY_PARENT: đang đợi để thực thi.
- TERMINATE: đã bị kết thúc.
- UNKNOWN: trạng thái của xử lý không xác định. Nếu một xử lý có trạng thái UNKNOWN thì tất cả các xử lý con đều có trạng thái INACTIVE..

Sơ đồ phân cấp của các xử lý



Hình 7-8 – Sơ đồ phân cấp của các xử lý.

7.3.1.3.1 Các phương thức hay được sử dụng

- `isReadyToExecute()`:
 - ▶ Trả về “true” nếu như xử lý thỏa điều kiện để thực thi thông qua việc kiểm tra các link đầu vào của xử lý.
- `execute()`
 - ▶ Đây là phương thức thực thi chính của các xử lý.
- `objectCompleted()`
 - ▶ Được gọi bởi xử lý khi chúng đã thực thi xong. Phương thức này sẽ xử lý các link đầu ra (hay còn gọi là source link) và thông báo cho tiến trình biết rằng nó đã hoàn thành.
- `terminate()`:
 - ▶ Được gọi bởi một tiến trình khi có lỗi xảy ra trong quá trình xử lý và tiến trình cần phải kết thúc.

7.3.1.4 Các loại tập tin

- .bpel: đây là tập tin định nghĩa chính của tiến trình.
- .pdd: đây là tập tin mô tả các thông tin liên quan đến quá trình triển khai một tiến trình, bao gồm các thông tin về partner link và các tập tin WSDL cần thiết.
- wsdlCatalog.xml: cung cấp cơ chế để quản lý toàn bộ các tập tin WSDL có liên quan trong tiến trình.

7.3.2 Các bước triển khai một business process trong BpelEngine

- Khởi động engine
- Tạo một thư mục có tên là tên của tiến trình và có phần mở rộng là .bpel. Bên trong thư mục này có chứa một tập tin mô tả thông tin triển khai tiến trình (.pdd) và các thư mục con có cấu trúc như sau:
 - ▶ Bpel: trong thư mục này sẽ chứa tập tin định nghĩa tiến trình (.bpel)
 - ▶ Wsdl: chứa tất cả các tập tin wsdl cần dùng trong tiến trình.
 - ▶ Partners: chứa các tập tin định nghĩa partner (option)
 - ▶ META-INF: chứa file wsdlCatalog.xml
- Chép toàn bộ thư mục trên vào thư mục deploy của engine. Engine sẽ tự động triển khai tiến trình trên.

Sử dụng tiến trình như một web service bình thường.

Chương 8

THÀNH PHẦN BPEL DESIGNER CỦA SOA SUITE

✍ Chương 8 sẽ giới thiệu về thành phần thứ ba của SOASuite, bộ công cụ “BpelDesigner” cung cấp môi trường trực quan hỗ trợ người dùng xây dựng, thiết kế các tiến trình nghiệp vụ.

8.1 Giới thiệu

BPEL Designer là một thành phần trong bộ SOA SUITE. BPEL Designer giúp cho người sử dụng định nghĩa ra các tiến trình (process) theo đúng chuẩn BPEL một cách dễ dàng, trực quan và nhanh chóng. Designer cung cấp cho người sử dụng một môi trường phát triển tích hợp IDE có thể hoạt động online hoặc offline.

8.2 Chức năng

8.2.1 Tạo mới, chỉnh sửa, thiết kế một tiến trình

Người dùng có thể tạo mới một tiến trình, hoặc mở những tiến trình đã có lên để chỉnh sửa và lưu lại. BPEL Designer hỗ trợ ngôn ngữ BPEL, cho phép người sử dụng thiết kế dạng kéo thả trực quan. Ngoài ra, người sử dụng vẫn có thể vừa chỉnh sửa mã nguồn vừa chỉnh sửa trên mô hình, khi chỉnh sửa một trong hai góc nhìn (dạng text, dạng trực quan), BPEL Designer sẽ tự động cập nhật nội dung tiến trình.

Để hỗ trợ thêm cho quá trình thiết kế BPEL Designer còn cung cấp cho người sử dụng thành phần UDDI service browser dùng để tìm kiếm các dịch vụ trên UDDI.

8.2.2 Chức năng kết xuất tiến trình ra file ảnh

Khi có được bản thiết kế vừa ý, người sử dụng có thể triển kết xuất hình ảnh trực quan của tiến trình ra định dạng file ảnh bmp. Chức năng này giúp tạo thuận lợi cho việc in ấn mô tả tiến trình nghiệp vụ.

8.2.3 Chức năng triển khai một tiến trình mới lên server

Sau khi có được bản thiết kế vừa ý, người sử dụng có thể triển khai tiến trình trực tiếp lên server hoặc đơn giản hơn là dùng chức năng triển khai của BPEL Designer. Với chức năng triển khai tự động của BPEL Designer, người sử dụng chỉ việc quan tâm đến thiết kế tiến trình, việc triển khai tiến trình bây giờ chỉ là thao tác bấm chọn đơn giản. Ngoài ra, người dùng có thể xem nội dung log file từ BPEL Designer để xem tiến trình có được triển khai thành công hay không, có bị lỗi thiết kế gì không.

8.3 Thiết kế cài đặt

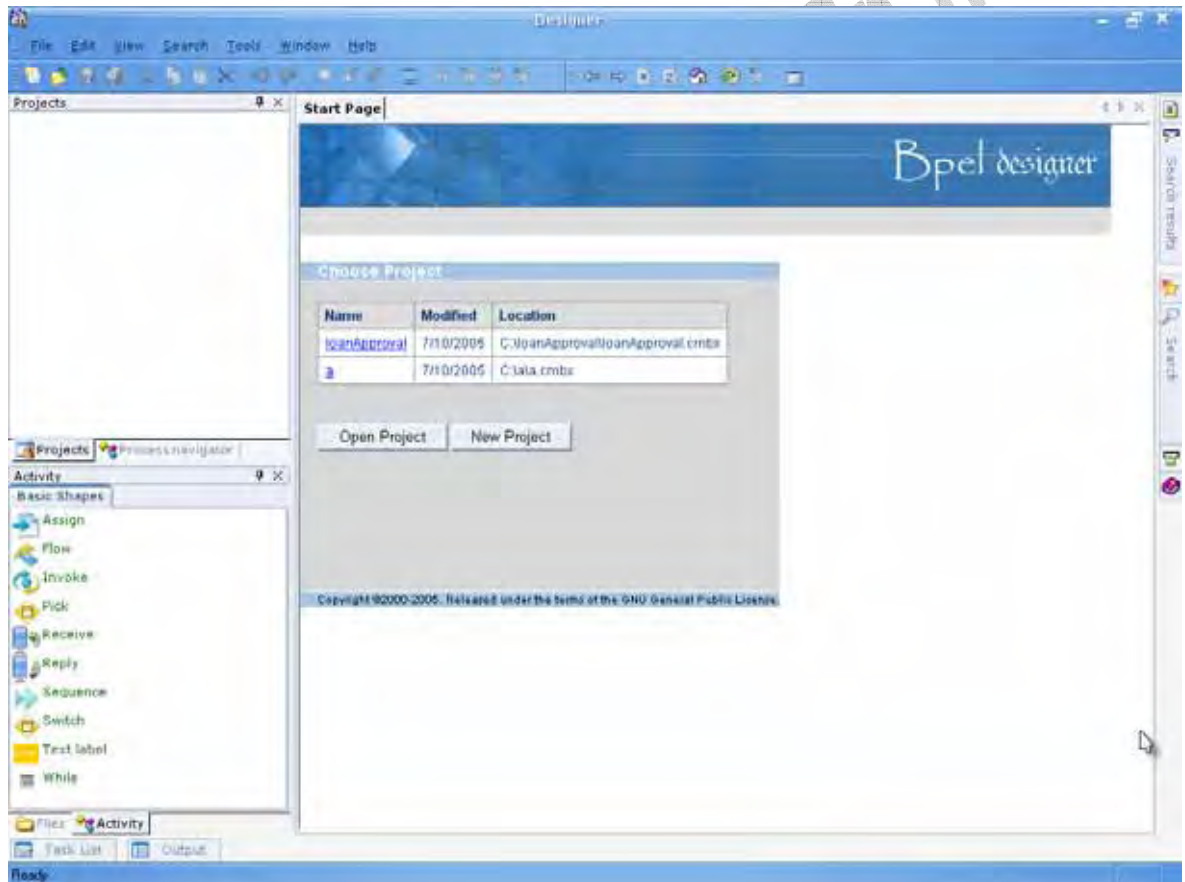
8.3.1 Cấu trúc chương trình

Thành phần BPEL designer có lõi dựa trên kiến trúc nền SharpDevelop và thêm vào các plug-in hỗ trợ BPEL. Các plug-in được viết theo đúng chuẩn và kết hợp với nhau sao cho chúng hoạt động một cách hiệu quả.

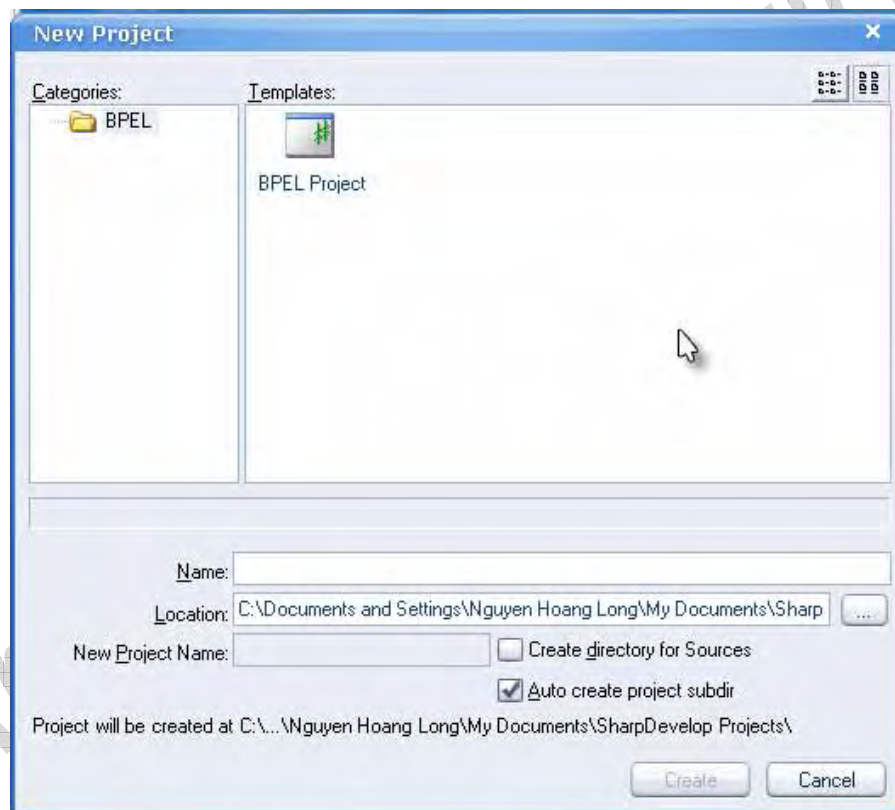
- Plug-in BPELBackendBinding : đây là plug-in chịu trách nhiệm phân tích nội dung cấu trúc BPEL thành các đối tượng, ngoài ra plug-in này còn chịu trách nhiệm xử lý quản lý các project BPEL.
- Plug-in BPELDisplayBinding : đây là plug-in hỗ trợ giao diện kéo-thả trực quan, cho phép chỉnh sửa trực tiếp trên mô hình. Đây cũng là plug-in chính cung cấp các form thao tác, quản lý, chỉnh sửa, cập nhật khi thiết kế các tiến trình BPEL.
- Plug-in BPELScout : plug-in này dùng để tạo cây quan sát và theo dõi thông tin các đối tượng bên trong tiến trình hiện hành.
- Plug-in BPELToolPad: plug-in này dùng cung cấp các xử lý thành phần của BPEL, cho phép kéo đối tượng xử lý vào màn hình thiết kế. Plug-in này được xây dựng theo kỹ thuật plug-in động, tiện lợi cho việc thêm bớt ,chỉnh sửa các xử lý sau này.

8.3.2 Giao diện chương trình

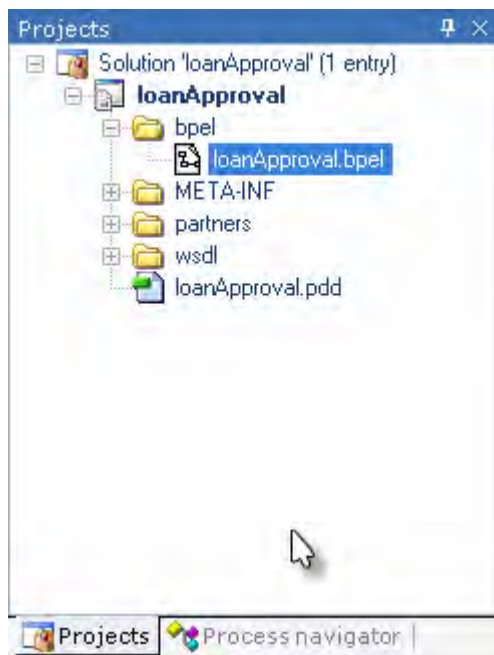
8.3.2.1 Màn hình chính



8.3.2.2 Màn hình chọn tạo mới một project

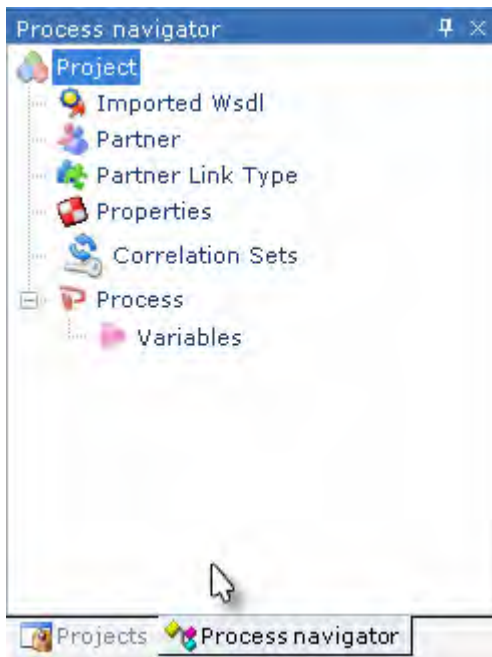


8.3.2.3 Cửa sổ project



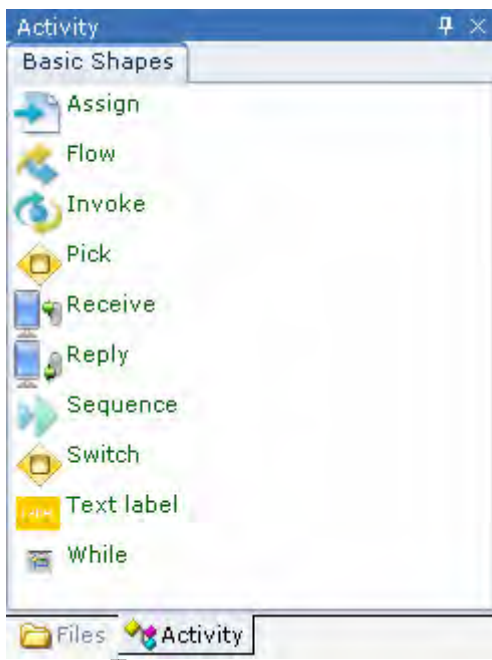
Đây là cửa sổ thao tác cho phép quản lý các file trong project BPEL hiện hành. Có thể truy cập đến cửa sổ này qua phím tắt Ctrl-Alt-L.

8.3.2.4 Cửa sổ process



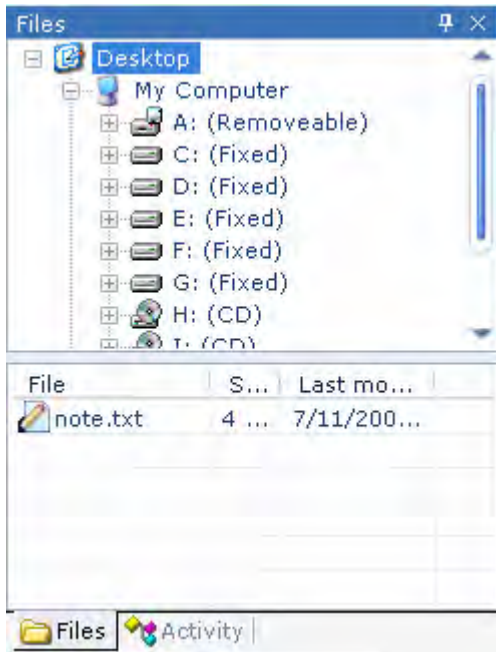
Cửa sổ thao tác cho phép xem, theo dõi các đối tượng BPEL của project BPEL hiện hành, nó còn biểu diễn cấu trúc phân cấp lồng nhau của các xử lý. Cửa sổ này tự động mỗi khi có bất kì đối tượng nào thay đổi. Có thể truy cập đến cửa sổ này qua phím tắt Ctrl-Alt-B.

8.3.2.5 Cửa sổ chọn xử lý



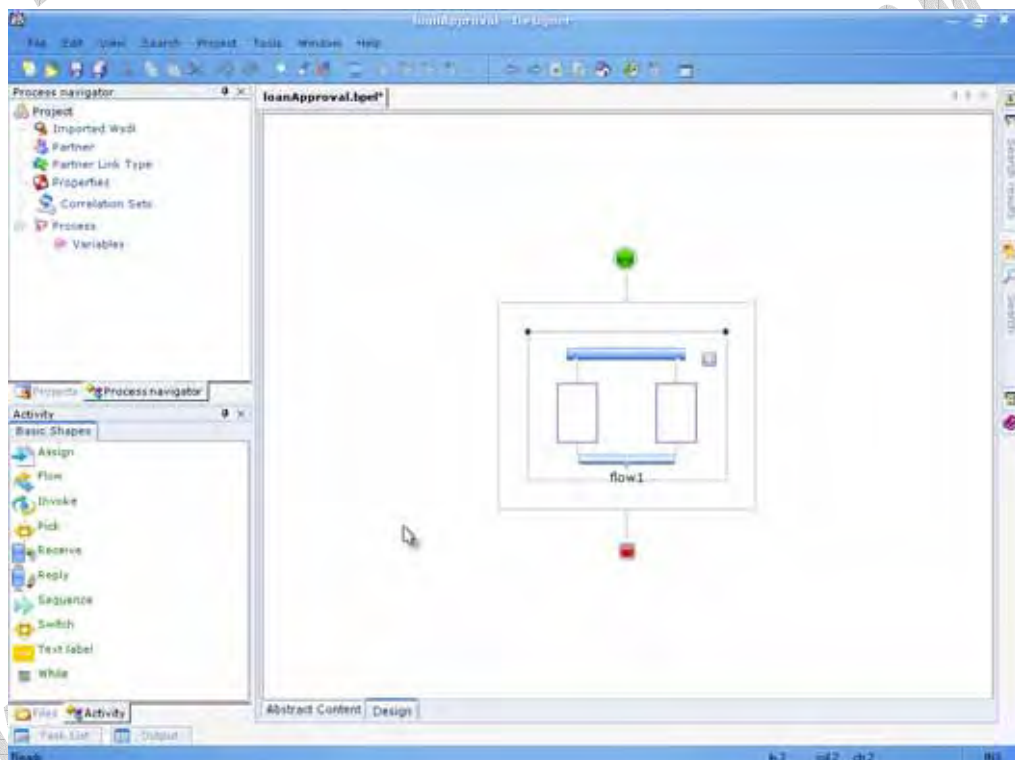
Đây là cửa sổ hiện danh sách các xử lý được hỗ trợ. Cửa sổ này cho phép kéo từng nút trực tiếp qua màn hình thiết kế. Các xử lý được phân loại theo từng tab trong trường hợp có nhiều phiên bản khác nhau hoặc có nhiều xử lý thuộc nhóm khác nhau. Có thể truy cập đến cửa sổ này qua phím tắt Ctrl-Alt-T.

8.3.2.6 Cửa sổ duyệt file



Cửa sổ này dùng để duyệt file trên máy. Nhấn double-click vào file để xem nội dung file (nếu là định dạng file được hỗ trợ). Có thể truy cập đến cửa sổ này qua phím tắt Ctrl-Alt-F

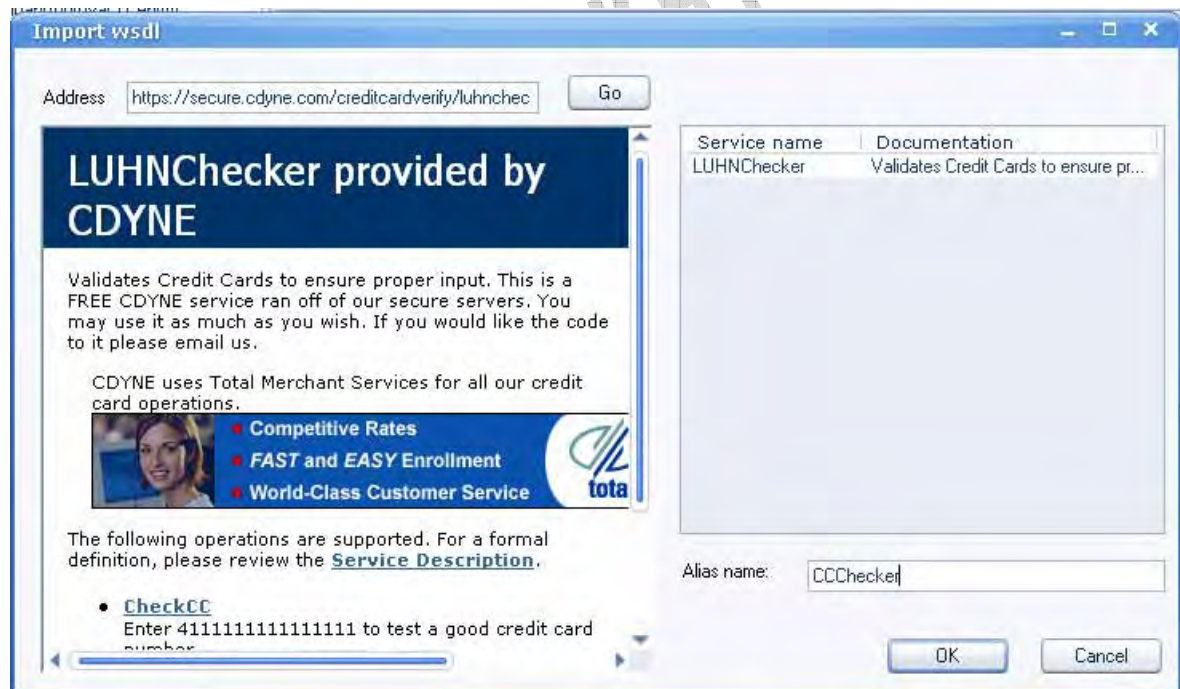
8.3.2.7 Vùng nhìn thiết kế process dạng kéo thả trực quan



Ta sẽ làm việc chính trên vùng nhìn này. Có hai tab tượng trưng cho hai vùng nhìn khác nhau: thiết kế trực quan-soạn thảo text. Tab “Design” chỉ hiện thi khi người

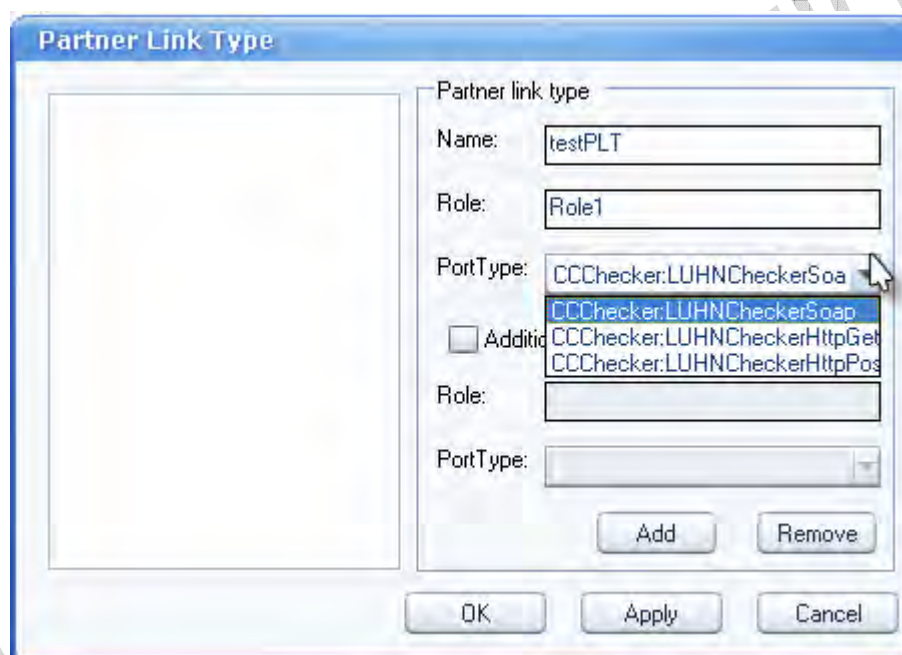
dùng chọn đúng file BPEL. Với vùng nhìn thiết kế trực quan người dùng có thể chọn, kéo thả, xoá, chỉnh sửa, phóng to, thu nhỏ một đối tượng bất kì. Người dùng có thể thoải mái chỉnh sửa ở một trong hai vùng nhìn, khi chuyển qua vùng nhìn còn lại thông tin sẽ được tự động cập nhật.

8.3.2.8 Cửa sổ chọn wsdl



Cửa sổ này dùng để chọn một dịch vụ trên mạng, trước tiên nhập địa chỉ dịch vụ vào thanh address, nhấn nút “Go” để tự động dò tìm dịch vụ, nếu tìm thấy dịch vụ thì danh sách các dịch vụ có trong địa chỉ vừa nhập sẽ hiện ở phần danh mục dịch vụ bên phải. Trước khi hoàn tất thao tác ta cần nhập tên đại diện ngắn gọn cho file wsdl ở textbox “Alias name”.

8.3.2.9 Màn hình thiết kế partner link type



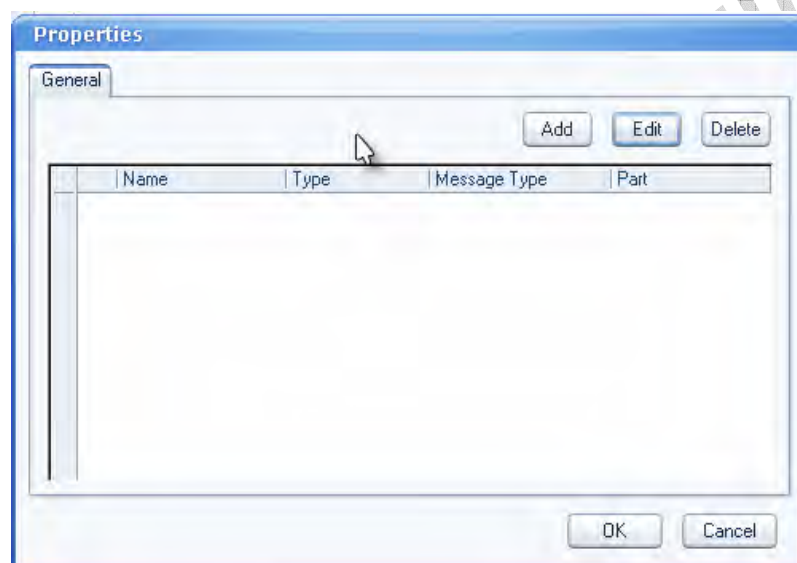
Màn hình này dùng để thiết kế các partner link type. Danh sách các partner link type hiện có sẽ hiện ở danh sách lên bên trái, mỗi khi chọn một partner link type nào trong danh sách thì thông tin bên phải sẽ tự động cập nhật. Nhấn nút add/remove dùng để thêm và xóa một partner link type trong danh sách.

8.3.2.10 Màn hình thiết kế partner link

Màn hình này dùng để thiết kế các partner link. Danh sách các partner link hiện có sẽ hiện ở danh sách bên trái, mỗi khi chọn một partner link nào trong danh sách thì thông tin bên phải sẽ tự động cập nhật. Nhấn nút add/remove dùng để thêm và xóa một partner link trong danh sách. Mỗi partner link thuộc về một partner link type. Partner link có thể là dạng partner cục bộ (inbound request) hoặc hướng ngoại (outbound request) hoặc cả hai.

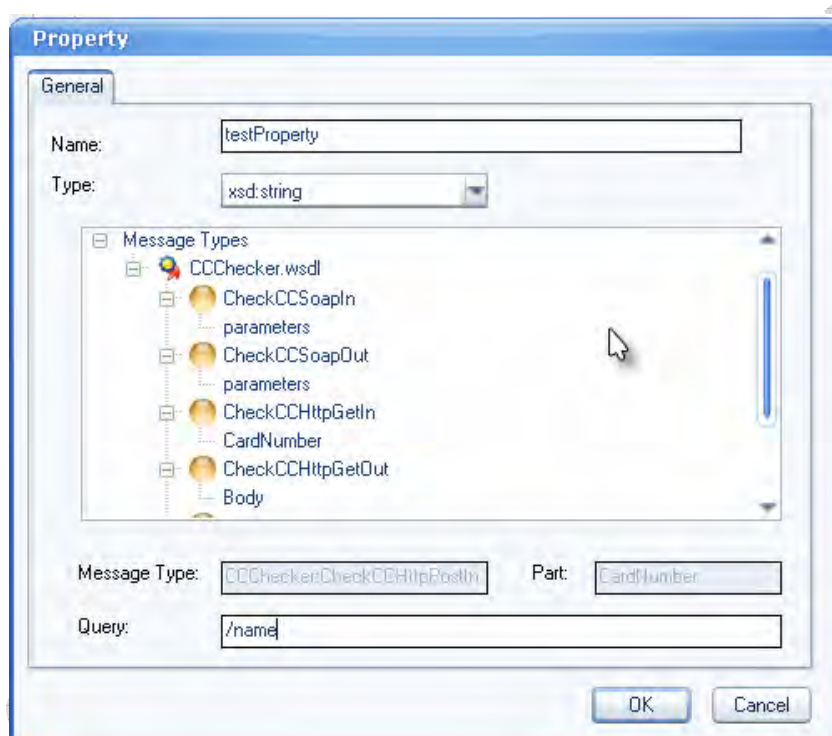
Với dịch vụ nào có dạng inbound thì ta phải chỉ định service name và chọn role tương ứng với partner link type mà partner link đó thuộc về. Trường allow role dùng để chỉ định role nào được phép truy cập đến nó, nếu để trống nghĩa là tất cả role được phép truy cập đến nó. Với dịch vụ nào có dạng outbound thì chỉ cần chọn những dịch vụ được import từ trước, tự động các thông tin liên quan sẽ được điền.

8.3.2.11 Màn hình quản lý các property



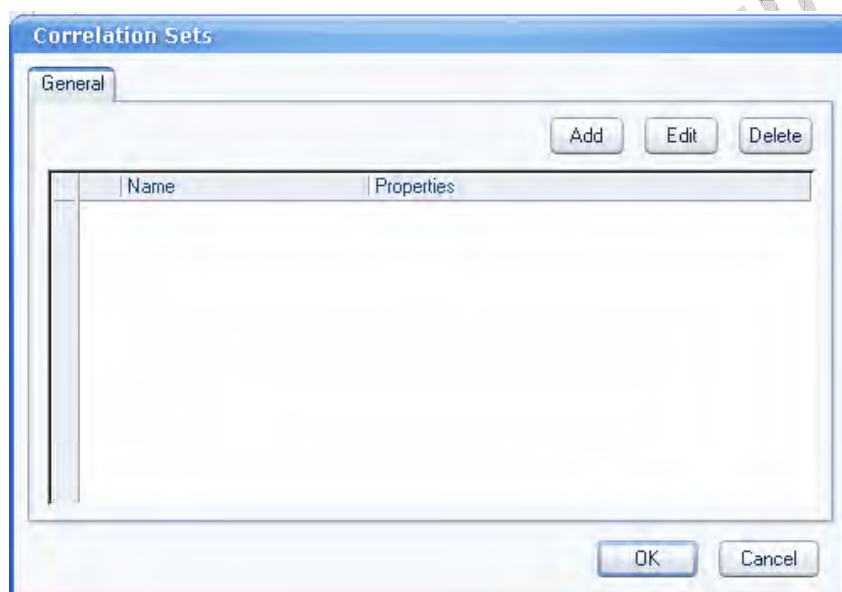
Màn hình này dùng để quản lý, cung cấp các property cho chức năng correlation set

8.3.2.12 Màn hình thêm mới/chỉnh sửa một property



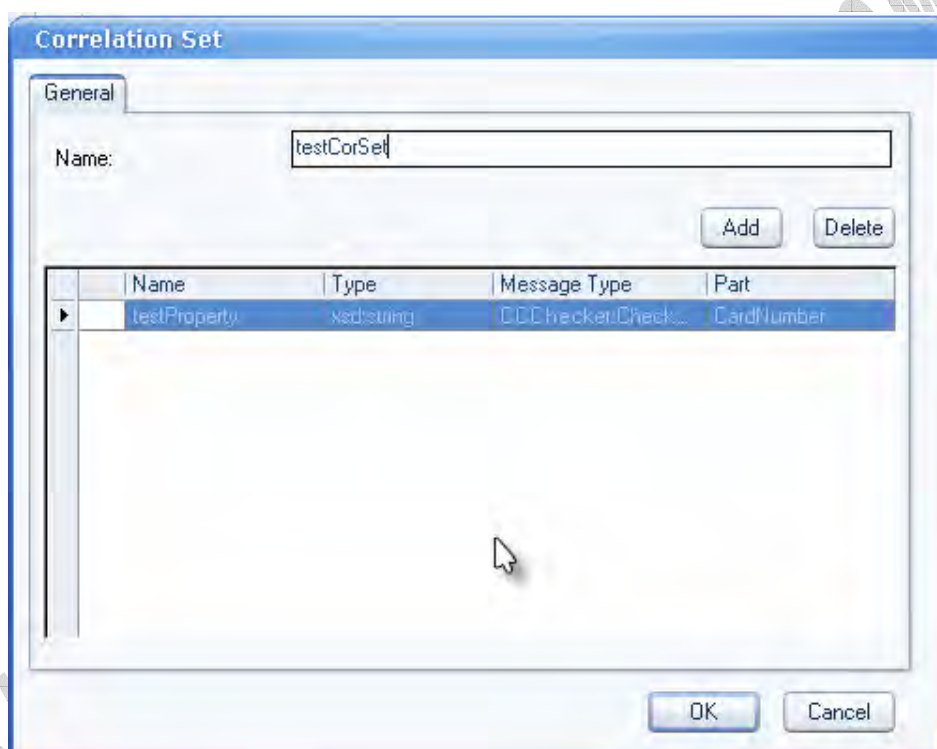
Màn hình này dùng để thêm mới, chỉnh sửa một property. Cây danh mục sẽ tự động liệt kê tất cả các kiểu message của từng file wsdl tương ứng được import vào project. Trường query là một biểu thức XPath dùng để chọn điều kiện biểu thức thích hợp.

8.3.2.13 Màn hình quản lý các correlation set



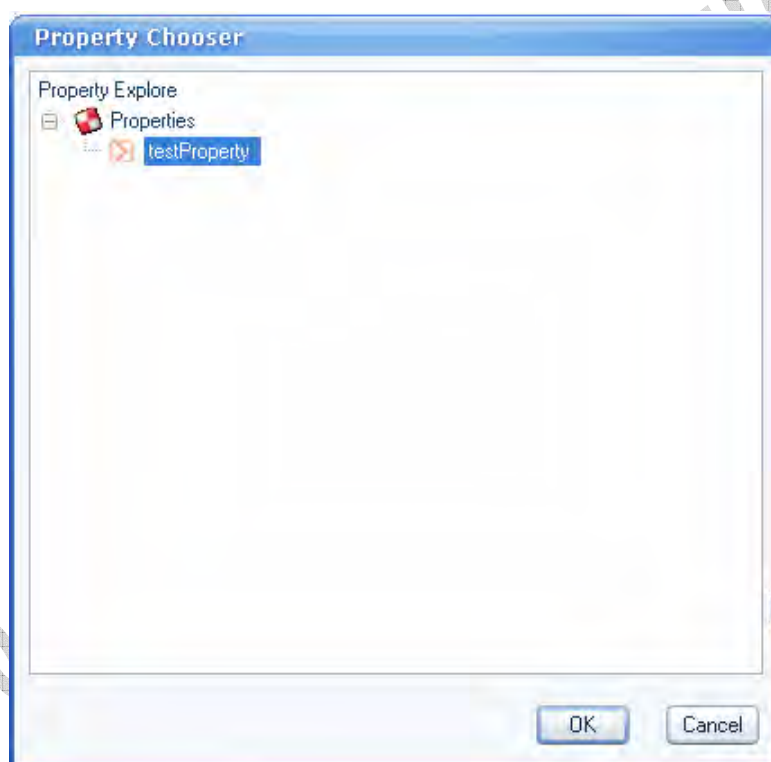
Màn hình này dùng để quản lý các correlation set

8.3.2.14 Màn hình thiết kế một correlation set

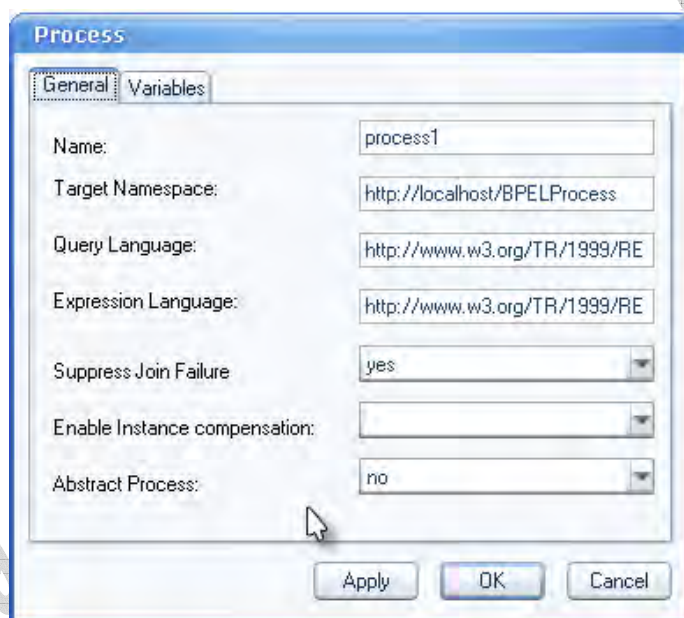


Màn hình này dùng để thiết kế, chỉnh sửa một correlation set. Một correlationset có nhiều property, ta có thể thêm hoặc xóa bằng nút Add/Remove

8.3.2.15 Màn hình chọn một property để thêm vào correlation set

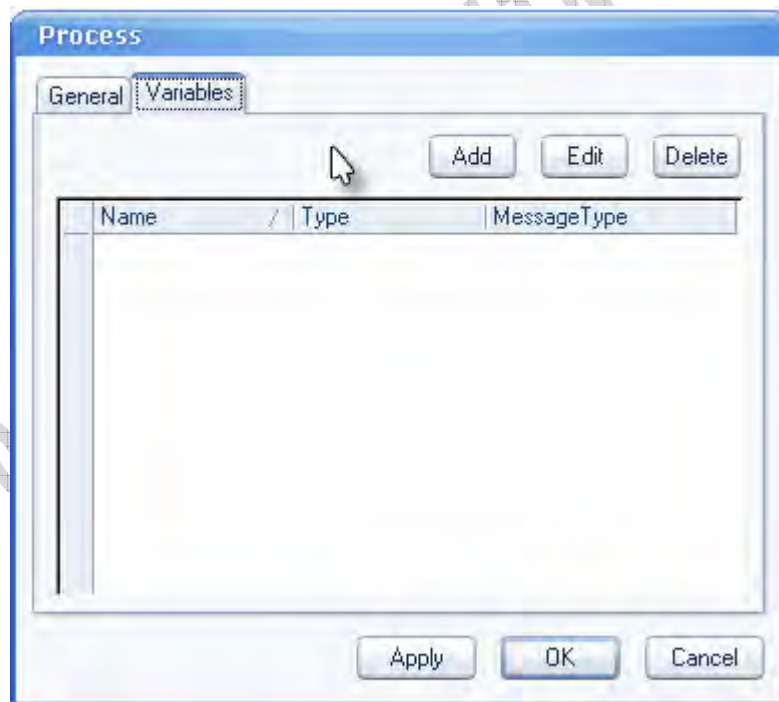


8.3.2.16 Màn hình chỉnh sửa thông tin process

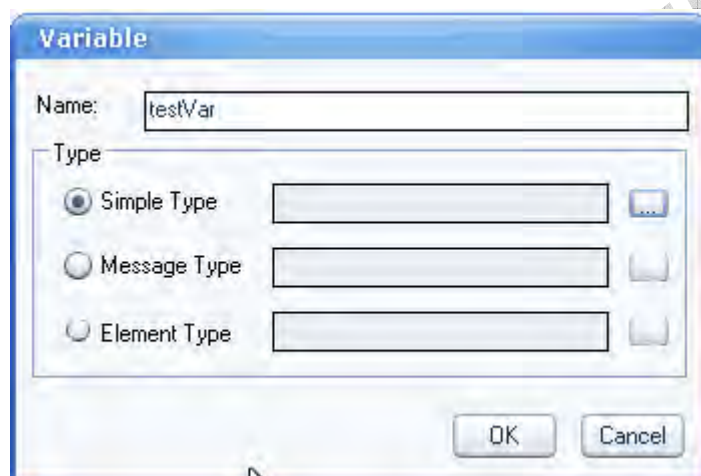


Màn hình này dùng để chỉnh sửa thông tin tiến trình. Mục Suppress Join Failure dùng để báo là sẽ không xử lý nếu có lỗi phát sinh. Mục Abstract Project mang ý nghĩa một tiến trình có được thực thi (tạo ra một thể hiện) mỗi khi có yêu cầu đến hay không.

8.3.2.17 Màn hình quản lý các biến variable

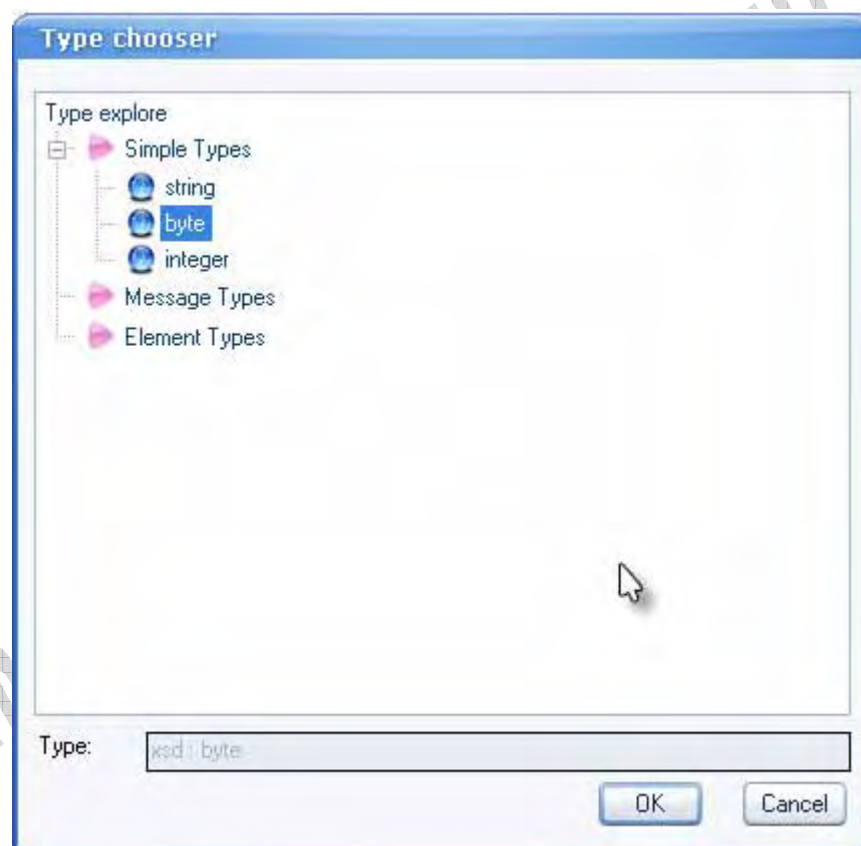


8.3.2.18 Màn hình thêm mới/chỉnh sửa một biến

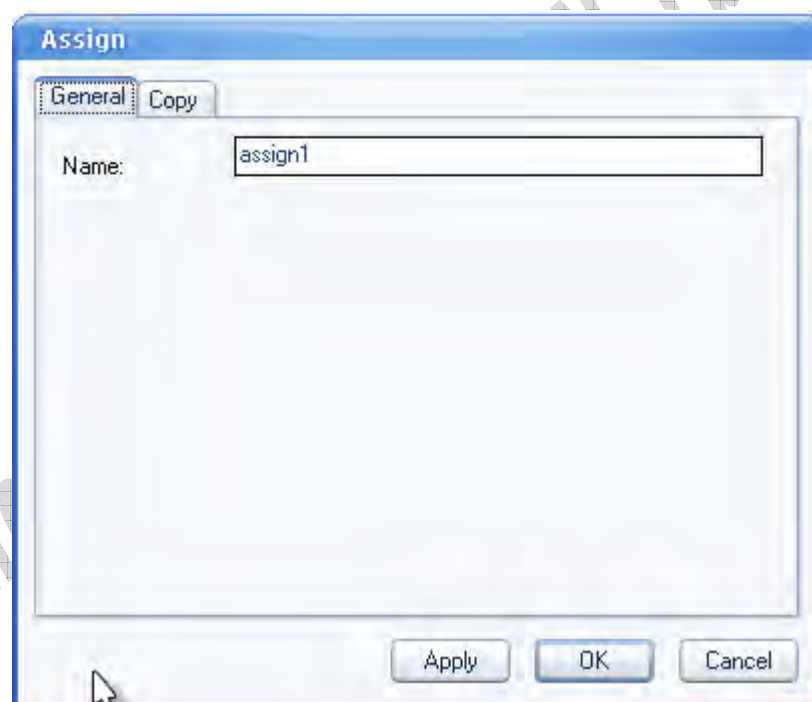


Màn hình này dùng để thêm mới/chỉnh sửa thông tin một biến variable. Một variable ngoài tên gọi sẽ thuộc về 1 trong 3 dạng Simple Type , Message Type hoặc Element Type. Nhấn vào nút bên cạnh để mở cửa sổ chọn kiểu cho variable.

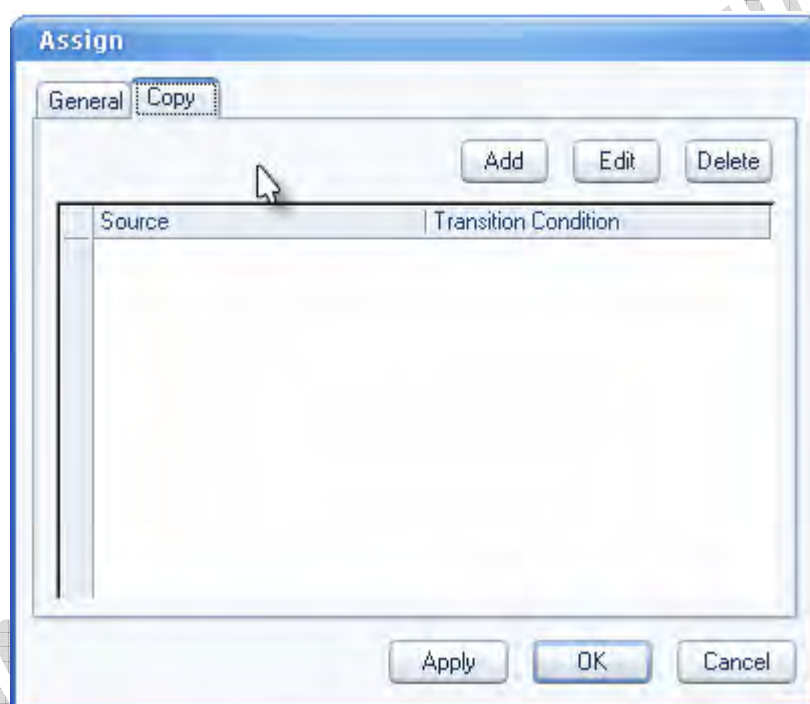
8.3.2.19 Màn hình chọn kiểu chọn biến



8.3.2.20 Màn hình chỉnh sửa thông tin xử lý assign

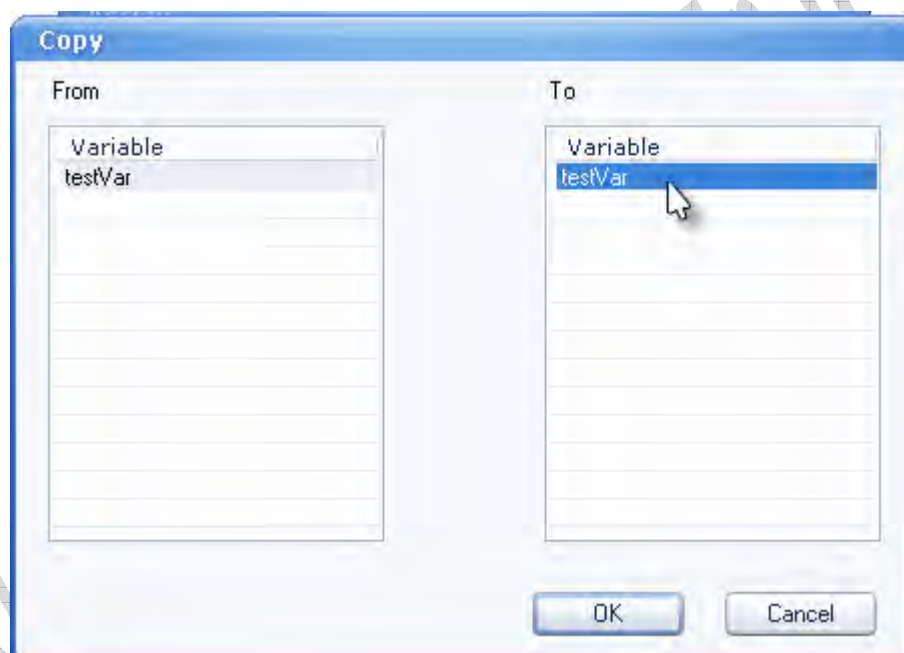


8.3.2.21 Màn hình quản lý các thành phần gán biến của xử lý assign



Màn hình này dùng để quản lý các thành phần copy của assign

8.3.2.22 Màn hình thêm mới/chỉnh sửa một thành phần gán biến của xử lý assign



Màn hình này dùng chọn hình thức copy từ biến nào sang biến nào.

8.3.2.23 Màn hình chỉnh sửa thông tin xử lý invoke

Invoke

Name:

Partner Link:

Port Type:

Operation:

Input Variable:

Output Variable:

8.3.2.24 Màn hình chỉnh sửa thông tin xử lý receive

Receive

Name:

Create Instance:

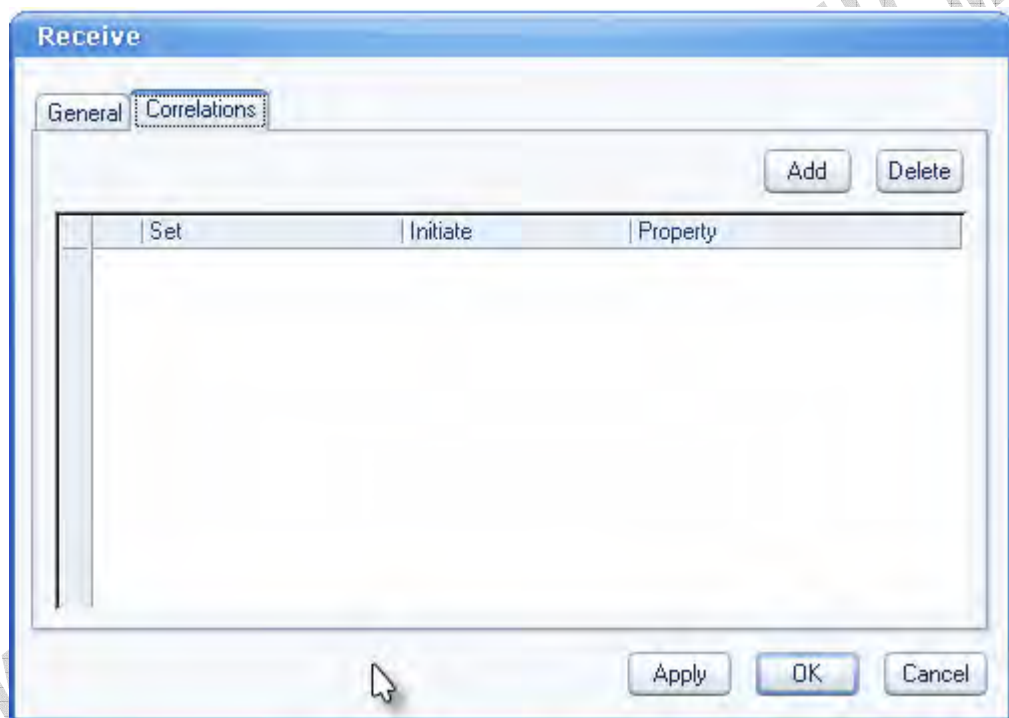
Partner Link:

Port Type:

Operation:

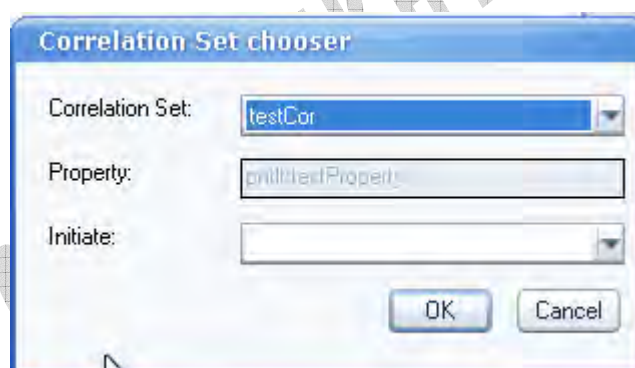
Variable:

8.3.2.25 Màn hình quản lý correlation set của xử lý receive

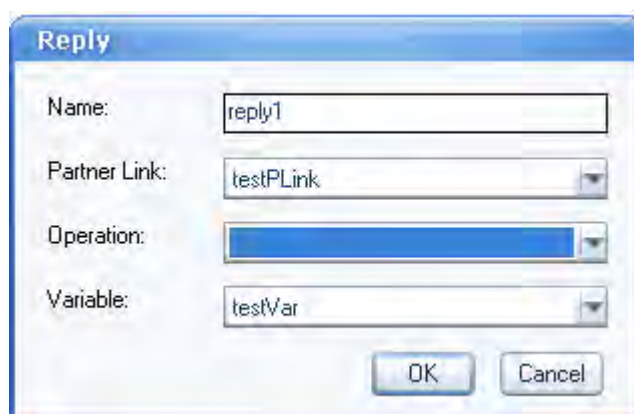


Màn hình này dùng để quản lý các correlation set của xử lý receive. Các correlation set tượng trưng cho định danh duy nhất của tiến trình, để mỗi khi có yêu cầu gửi đến thì BPEL server sẽ chuyển đến đúng thể hiện tiến trình mà client cần trao đổi thông tin

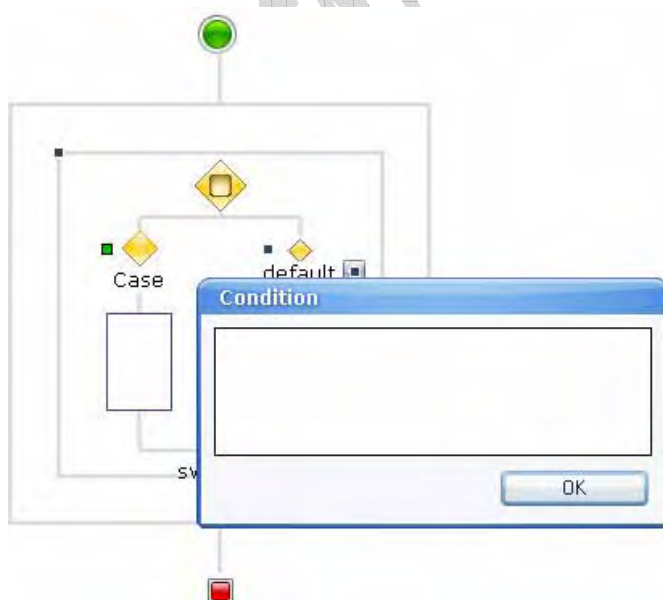
8.3.2.26 Màn hình chọn thêm mới correlation set cho xử lý receive



8.3.2.27 Màn hình chỉnh sửa thông tin xử lý reply

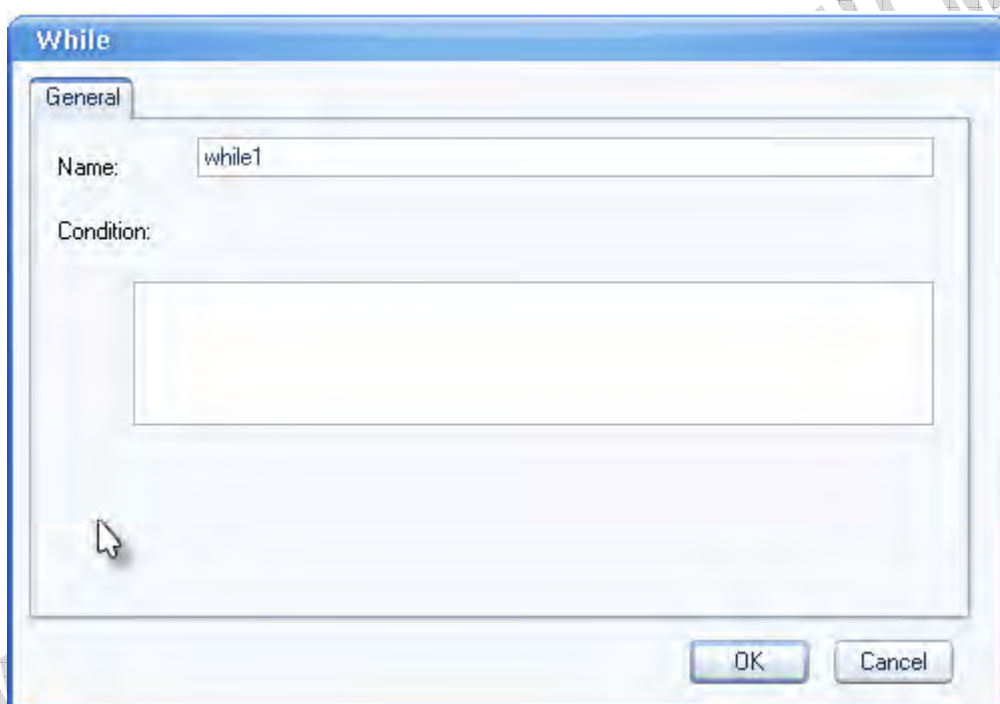


8.3.2.28 Cửa sổ thiết kế điều kiện cho một nhánh điều kiện của xử lý switch



Điều kiện bên trong nhánh phải là một biểu thức XPath với namespace là bpws. Ví dụ bpws:getAttribute("a") > 100.

8.3.2.29 Màn hình chỉnh sửa thông tin của xử lý while

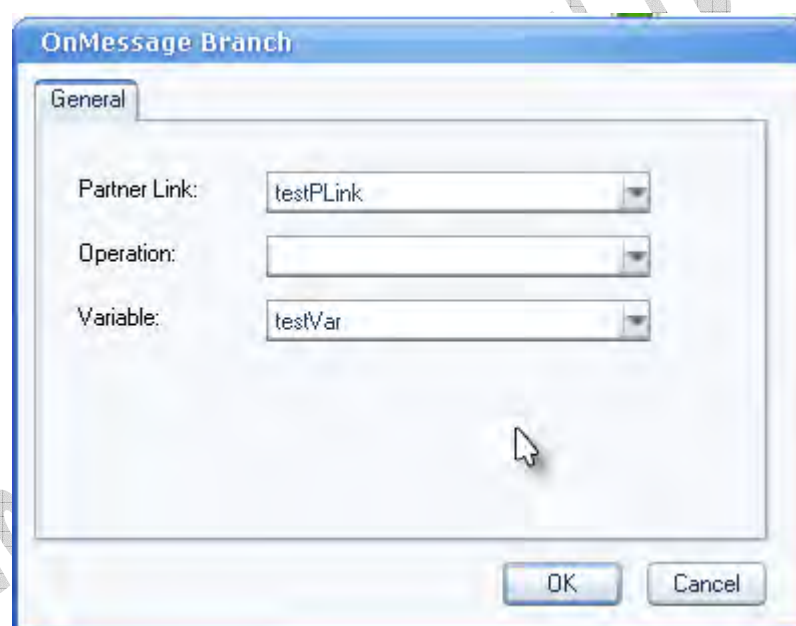


The 'While' dialog box is shown with the following details:

- Title:** While
- Tab:** General
- Name:** while1
- Condition:** (Empty text area)
- Buttons:** OK, Cancel

Màn hình này dùng để chỉnh sửa thông tin xử lý while, biểu thức bên trong condition là một biểu thức luận lý (có thể kết hợp sử dụng biểu thức XPath)

8.3.2.30 Màn hình chỉnh sửa thông tin một nhánh OnMessage của xử lý pick



The 'OnMessage Branch' dialog box is shown with the following details:

- Title:** OnMessage Branch
- Tab:** General
- Partner Link:** testPLink
- Operation:** (Empty dropdown menu)
- Variable:** testVar
- Buttons:** OK, Cancel

8.3.2.31 Màn hình chỉnh sửa thông tin một nhánh OnAlarm của xử lý pick

Màn hình này dùng để chỉnh sửa thông tin một nhánh onAlarm của xử lý pick. Điều kiện kích hoạt nhánh onAlarm thuộc về một trong 4 dạng sau: kích hoạt sau một khoảng thời gian (ví dụ sau 3 giờ 20 phút 10 giây chạy), kích hoạt dựa trên biểu thức XPath, kích hoạt khi đến đúng thời điểm nào đó (ví dụ thứ Hai, ngày 11, tháng 7, năm 2005), và cuối cùng là kích hoạt nếu một biểu thức XPath thỏa.

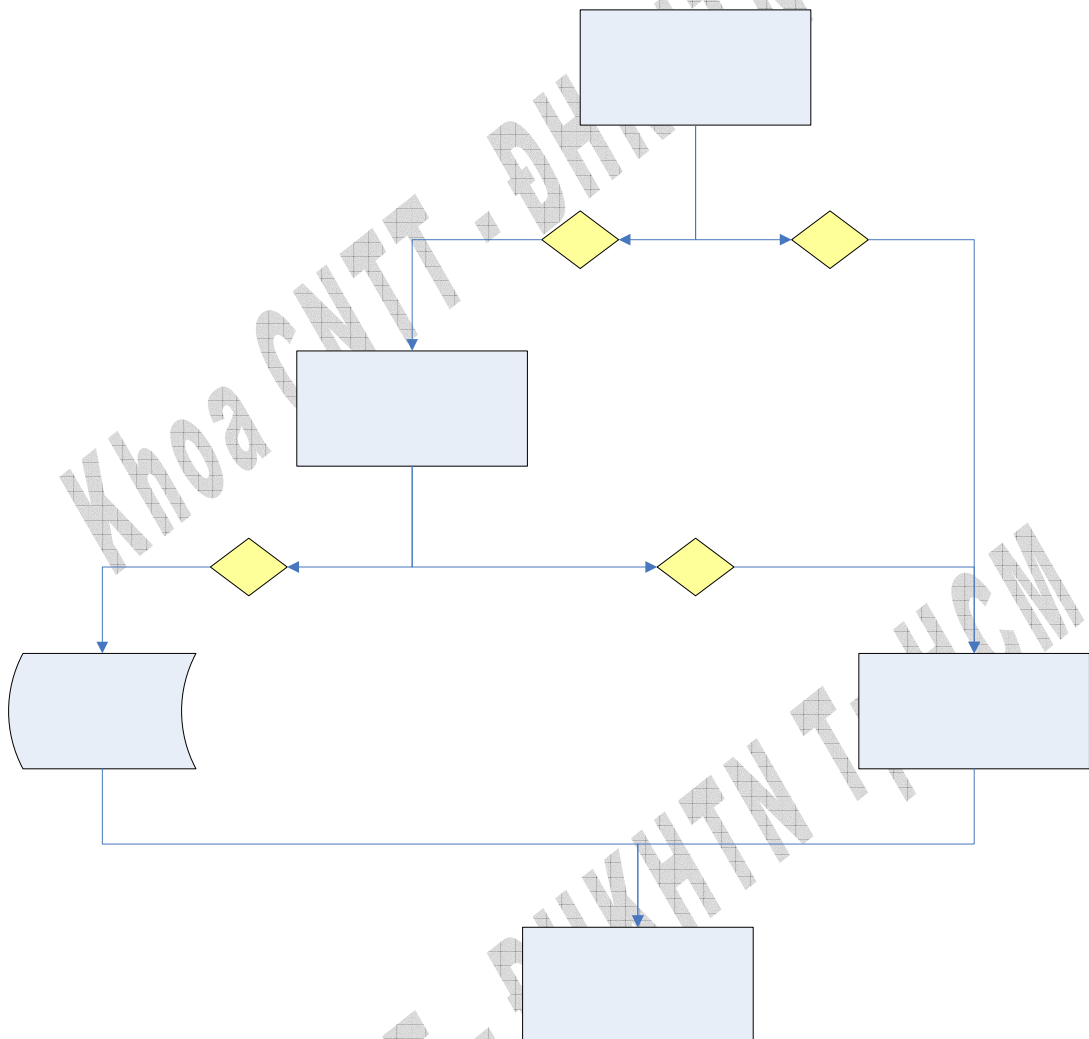
8.4 Hướng dẫn sử dụng

8.4.1 Thiết kế một tiến trình

Thiết kế một tiến trình bên trong BPEL Designer tương đối đơn giản. Một tiến trình cơ bản cần trải qua các bước import các service bên ngoài vào, thiết kế partner link, thiết kế luồng xử lý và thu thập kết quả trả về.

Ta sẽ thiết kế một process BPEL ví dụ mẫu có 3 partner: một partner cho người sử dụng và hai Web Service sử dụng bởi tiến trình. Đây là tiến trình ví dụ này về dịch vụ cho vay trên mạng được mô tả trong đặc tả ngôn ngữ BPEL 1.1.

Hình sau minh họa quy trình xử lý của tiến trình



Và đây là mô tả dạng mã giả

```

String processLoanRequest(message) {
    if (message.amount < 1000 &
        "low".equals(assessor.risk(message)))
        return "approved";
    else
        return approver.approve(message);
}
  
```

Với `assessor.risk` và `approver.approve` là những phương thức triệu gọi từ các Web Service bên ngoài.

Ta sẽ bắt đầu với tiến trình và những xử lý thành phần, sau đó liên kết các xử lý lại với nhau bằng những mối liên kết. Cuối cùng ta sẽ điền tất cả thông tin chi tiết cần thiết cho tiến trình BPEL để nó có thể trao đổi với bên ngoài.

Ta bắt đầu tạo mới một project BPEL với menu File/New/Project và chọn loại project là BPEL, tên project là loanApproval, chọn đường dẫn thư mục chứa project. Designer sẽ tạo ra một project theo đúng dạng BPEL như sau

```
<?xml version="1.0" encoding="utf-8"?>
<process name="process1" suppressJoinFailure="yes"
  targetNamespace="http://localhost/BPELProcess"
  queryLanguage="http://www.w3.org/TR/1999/REC-xpath-19991116"
  expressionLanguage="http://www.w3.org/TR/1999/REC-xpath-19991116"
  abstractProcess="no"
  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
</process>
```

Tiến trình mặc định chỉ chứa một xử lý cấp cao nhất nên muốn có nhiều xử lý hoạt động bên trong tiến trình ta phải thêm các xử lý cấu trúc có khả năng xử lý nhiều xử lý con bên trong như (flow, sequence, switch, ...). Thông thường ta chọn dùng xử lý flow là phần tử con của tiến trình. Có hai cách để thêm một xử lý vào tiến trình, hoặc là dùng chức năng kéo thả của designer hoặc là chỉnh sửa trực tiếp trên màn hình soạn thảo code, tự động designer sẽ cập nhật nội dung vùng nhìn còn lại.

```
<?xml version="1.0" encoding="utf-8"?>
<process name="process1" suppressJoinFailure="yes"
  targetNamespace="http://localhost/BPELProcess"
  queryLanguage="http://www.w3.org/TR/1999/REC-xpath-19991116"
  expressionLanguage="http://www.w3.org/TR/1999/REC-xpath-19991116"
  abstractProcess="no"
  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <flow name="flow1"></flow>
</process>
```

Bây giờ ta do ta cần sử dụng các chức năng của Web Service bên ngoài nên phải import chúng vào hệ thống. Để import ta chọn chức năng **project/WSDL** và chọn đến địa chỉ Web Service tương ứng, ngoài ra cần phải nhập tên đại diện cho Web Service.

Sau khi import xong ta cần thiết kế các partner link và partner link type để các xử lý có thể trao đổi liên lạc với nhau. Trước tiên phải tạo partner link type tương ứng với các Web Service vừa import bằng cách sử dụng chức năng menu **Project/Partner Link Type**. Dựa trên các partner link type vừa tạo ta sẽ thiết kế các Partner Link tương ứng cho các xử lý trong tiến trình bằng cách sử dụng chức năng menu **Project/Partner Link**.

Trước khi thiết kế những phần sau, ta cần định nghĩa một số biến (variable). Ở đây tiến trình cần ba biến: một cho thông điệp nhận từ client, một cho risk assessment message và một cho thông điệp trả về.

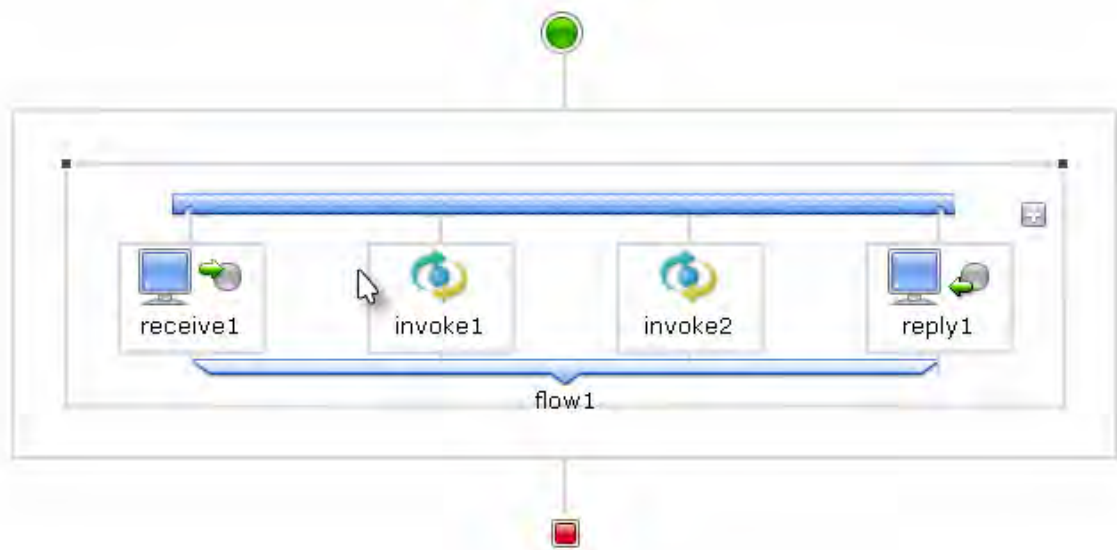
```
<?xml version="1.0" encoding="utf-8"?>
<variables>
  <variable messageType="lns:creditInformationMessage"
    name="request" />
  <variable messageType="lns:riskAssessmentMessage"
    name="risk" />
  <variable messageType="lns:approvalMessage"
    name="approval" />
</variables>
```

Để nhận một thông điệp từ một client và trả về thông tin cần thiết, ta phải tạo xử lý **receive** và **reply**.

```
<?xml version="1.0" encoding="utf-8"?>
<process name="process1" suppressJoinFailure="yes"
  targetNamespace="http://localhost/BPELProcess"
  queryLanguage="http://www.w3.org/TR/1999/REC-xpath-19991116"
  expressionLanguage="http://www.w3.org/TR/1999/REC-xpath-19991116"
  abstractProcess="no"
  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <flow name="flow1">
    <receive createInstance="yes" operation="request"
      partnerLink="customer" portType="lns:loanServicePT"
      variable="request">
    </receive>

    <reply operation="request" partnerLink="customer"
      portType="lns:loanServicePT" variable="approval">
    </reply>
  </flow>
</process>
```

Để sử dụng những dịch vụ bên ngoài ta cần sử dụng xử lý **invoke** để triệu gọi chúng. Trong tiến trình này ta cần 2 xử lý **invoke**.



```
<?xml version="1.0" encoding="utf-8"?>
<invoke inputVariable="request" name="InvokeAssessor"
  operation="check" outputVariable="risk"
  partnerLink="assessor" portType="lns:riskAssessmentPT">
</invoke>
<invoke inputVariable="request" name="InvokeApprover"
  operation="approve" outputVariable="approval"
  partnerLink="approver" portType="lns:loanApprovalPT">
</invoke>
```

Cuối cùng, để liên kết các xử lý theo trình tự ta sẽ tạo các link trong **flow**.

```
<links>
  <link name="receive-to-assess"/>
  <link name="receive-to-approval"/>
  <link name="assess-to-setMessage"/>
  <link name="assess-to-approval"/>
  <link name="setMessage-to-reply"/>
  <link name="approval-to-reply"/>
</links>
```

```
<receive createInstance="yes" operation="request"
  partnerLink="customer" portType="lns:loanServicePT"
  variable="request">
  <source linkName="receive-to-assess"
    transitionCondition=
      "bpws:getVariableData('request','amount') < 10000"/>
  <source linkName="receive-to-approval"
    transitionCondition=
      "bpws:getVariableData('request','amount') >= 10000"/>
</receive>
```

```

<invoke inputVariable="request" name="invokeAssessor"
  operation="check" outputVariable="riskAssessment"
  partnerLink="assessor" portType="asns:riskAssessmentPT">
  <target linkName="receive-to-assess"/>
</invoke>

```

Ngoài ra để gán giá trị các biến ta cần phải tạo một xử lý **assign**

```

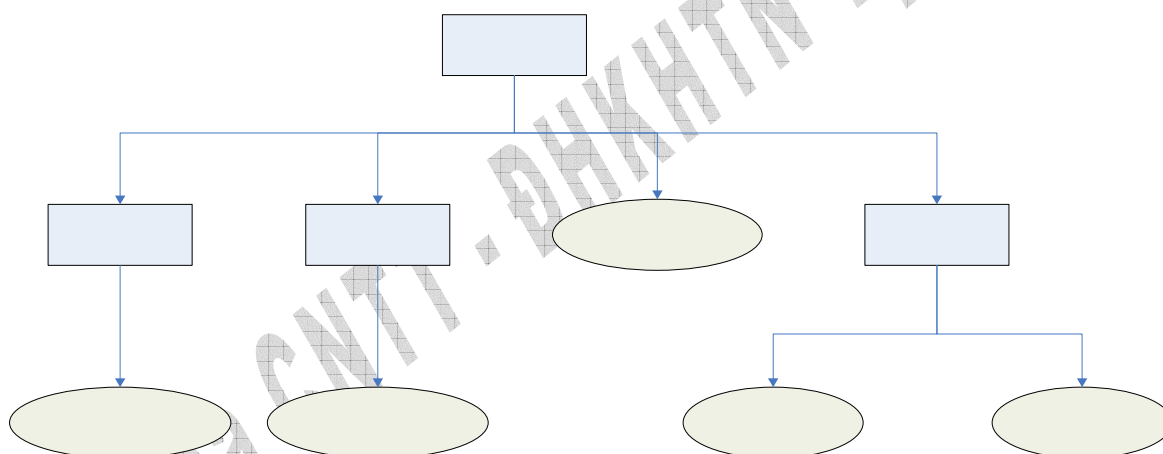
<assign>
  <target linkName="assess-to-setMessage"/>
  <source linkName="setMessage-to-reply"/>
  <copy>
    <from expression="'yes'"/>
    <to part="accept" variable="approval"/>
  </copy>
</assign>

```

Đến đây thì tiến trình đã được thiết kế xong.

8.4.2 Triển khai một tiến trình

Sau khi thiết kế xong tiến trình, ta có thể triển khai tiến trình lên server theo 2 cách: triển khai lên server một cách thủ công – bằng cách chép các file cần thiết lên server để server tự động tìm và triển khai, cách thứ hai là dùng chức năng triển khai tự động của designer. Để thực hiện chức năng triển khai tự động, mở project BPEL lên và chọn chức năng menu **Project / Deploy** hoặc từ toolbar. Cấu trúc thư mục deploy cần theo đúng định dạng sau



Hình 8-1 – Cấu trúc thư mục dùng triển khai một tiến trình BPEL

Định dạng các file WSDL catalog (wsdlCatalog.xml), file định nghĩa partner (.pdef) và file mô tả cấu trúc triển khai (.pdd) được mô tả trong phụ lục Phụ lục C). Thư mục partners là tùy chọn nếu ta có định nghĩa file pdef.

Chương 9

ỨNG DỤNG SOA ĐỂ THIẾT KẾ MỘT SỐ TIẾN TRÌNH

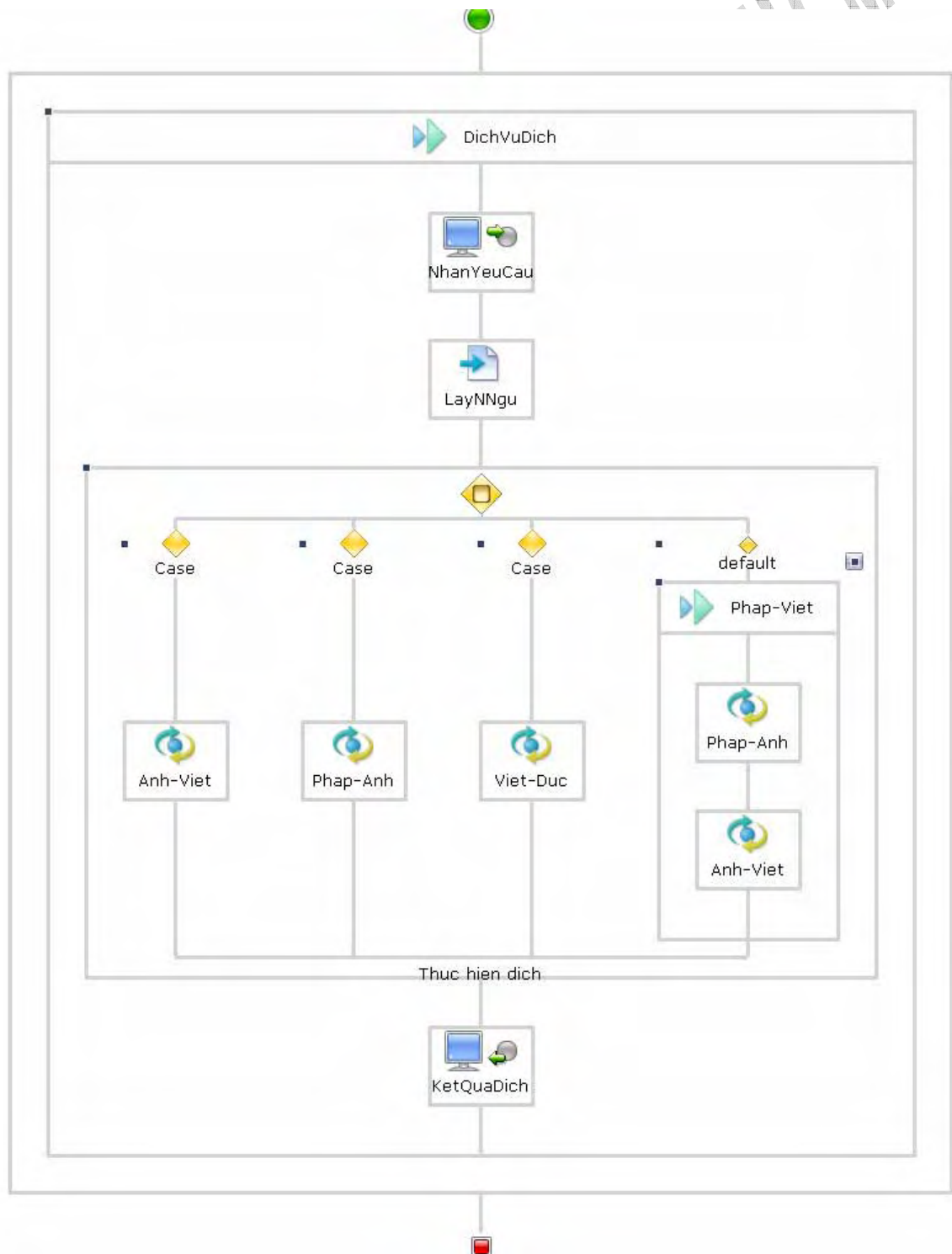
✍ Chương 9 sẽ giới thiệu một số mẫu tiến trình được thiết kế bằng bộ công cụ BpelDesigner.

9.1 Tiến trình dịch tự động đa ngôn ngữ

9.1.1 Mô tả

Mục tiêu xây dựng tiến trình này là nhằm hỗ trợ dịch tự động từ ngôn ngữ này sang ngôn ngữ khác. Giả sử ta có 2 module là dịch “Anh-Việt” và module dịch “Việt-Campuchia” và 2 module đó hoạt động độc lập với nhau. Với 2 module đó, về lý thuyết ta có thể kết hợp theo nhiều dạng “Anh-Việt-Campuchia” để cho ra một số kết quả mới như dịch “Anh-Campuchia”. Dùng kỹ thuật thông thường thì ta phải viết chương trình từ đầu để kết hợp các module lại với nhau. Việc gì sẽ xảy ra nếu có 5,6,7...và rất nhiều module và cần kết hợp chúng để cho ra các dạng dịch chưa được hỗ trợ nhưng có khả năng kết hợp qua trung gian? Sửa lại từ mã nguồn? Hoặc đơn giản là một module nào đó bị hỏng, được thay thế bằng một module khác có chức năng tương tự, cũng phải sửa mã nguồn? Khó khăn nằm ở mối ràng buộc giữa những module đó. Với SOA, ta chỉ đơn giản cung cấp những module đó thành dạng dịch vụ và kết hợp với nhau theo ý muốn là xong. Việc hỗ trợ một hình thức dịch mới có khả năng kết hợp từ nhiều dịch vụ cơ sở được thực hiện rất nhanh chóng và dễ dàng.

9.1.2 Sơ đồ



Hình 9-1 – Tiến trình dịch tự động

9.1.3 Mô tả luồng xử lý

Đầu tiên xử lý “NhanYeuCau” sẽ nhận yêu cầu từ người sử dụng. Thông tin đầu vào của thông điệp gửi đến xử lý “NhanYeuCau” gồm hai phần: loại ngôn ngữ cần dịch (ví dụ “Anh-Phap”) và chuỗi cần dịch.

Xử lý “NhanYeuCau” sẽ gửi toàn bộ thông điệp đó đến xử lý “LayNNgu” (là một xử lý dạng assign) để tách 2 thành phần thông tin ra. Rồi chuyển phân loại ngôn ngữ cần dịch cho xử lý “Thực hiện dịch” để xử lý này chọn nhánh xử lý thích hợp.

Mỗi nhánh của xử lý “Thực hiện dịch” nhận thông tin đầu vào là chuỗi cần dịch sau một loạt xử lý tuần tự sẽ cho ra đoạn văn bản ở ngôn ngữ cần dịch. Ở giai đoạn này ta có thể tự do phối hợp các dịch vụ dịch tự động với nhau. Giả sử ta chỉ có dịch vụ dịch “Pháp – Anh” và “Anh – Việt”, ta sẽ kết hợp 2 dịch vụ này theo trình tự để có kết quả dịch “Pháp – Việt”, tương tự và mở rộng cho các cách kết hợp các với dịch vụ khác.

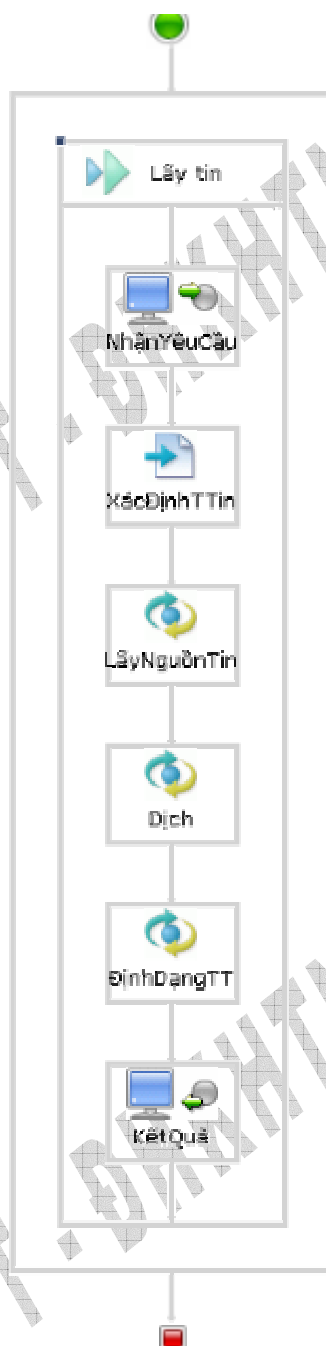
Sau khi có kết quả dịch tiến trình sẽ tự động trả kết quả về cho người sử dụng.

9.2 Tiến trình thu thập thông tin từ bên ngoài

9.2.1 Mô tả

Việc thu thập thông tin tự động từ những nguồn tin trên thế giới rất quan trọng. Đa phần những nguồn tin trên thế giới không cung cấp dạng Web Service và chúng ở nhiều ngôn ngữ khác nhau. Ta sẽ thiết kế một tiến trình có thể lấy thông tin một trang web bất kỳ, qua một bộ lọc, dịch ngôn ngữ đó thành tiếng việt nếu có thể và xuất ra nhiều định dạng khác nhau (dạng WAP, dạng Web, dạng RSS, dạng XML, dạng text đơn thuần...). Tiến trình này tái sử dụng lại tiến trình dịch tự động đa ngôn ngữ ở trên.

9.2.2 Sơ đồ



Hình 9-2 – Tiến trình thu thập thông tin và dịch tự động

9.2.3 Mô tả luồng xử lý

Đầu tiên xử lý “NhanYeuCau” sẽ nhận yêu cầu từ người sử dụng. Thông tin đầu vào của thông điệp gửi đến xử lý “NhanYeuCau” gồm bốn phần: *nguồn tin* (một địa chỉ

web bất kì), loại *ngôn ngữ cần dịch* (ví dụ “Anh-Phap”) và *địa chỉ file XSLT đầu vào* cho bộ dịch, *địa chỉ file XSLT đầu ra* cho kết xuất ngôn ngữ tùy theo định dạng.

Xử lý “NhanYeuCau” sẽ gửi toàn bộ thông điệp đó đến xử lý “XacDinhTT” (là một xử lý dạng assign) để phân tích các yêu cầu thông tin. Rồi chuyển phần *nguồn tin* và *địa chỉ file XSLT đầu vào* cho xử lý “LấyThôngTin”. Xử lý “LấyThôngTin” có nhiệm vụ lấy toàn bộ dữ liệu trang web đó về và chuyển đổi thông tin bằng địa chỉ XSLT được cung cấp trước khi cung cấp nội dung cần dịch cho xử lý “Dịch”.

Xử lý “Dịch” thực chất là triệu gọi lại dịch vụ “Dịch tự động đa ngôn ngữ” đã thiết kế ở trên. Điều này minh họa khả năng kết hợp và tái sử dụng dịch vụ của BPEL.

Xử lý “Định dạng TT” sẽ được triệu gọi sau khi dịch xong, nó có nhiệm vụ nhận kết quả dịch và chuyển đổi dữ liệu về định dạng thích hợp nhờ file XSLT được cung cấp cho tiến trình ban đầu. Đến đây ta đã có bản tin từ nguồn bên ngoài và được dịch qua ngôn ngữ mình cần.

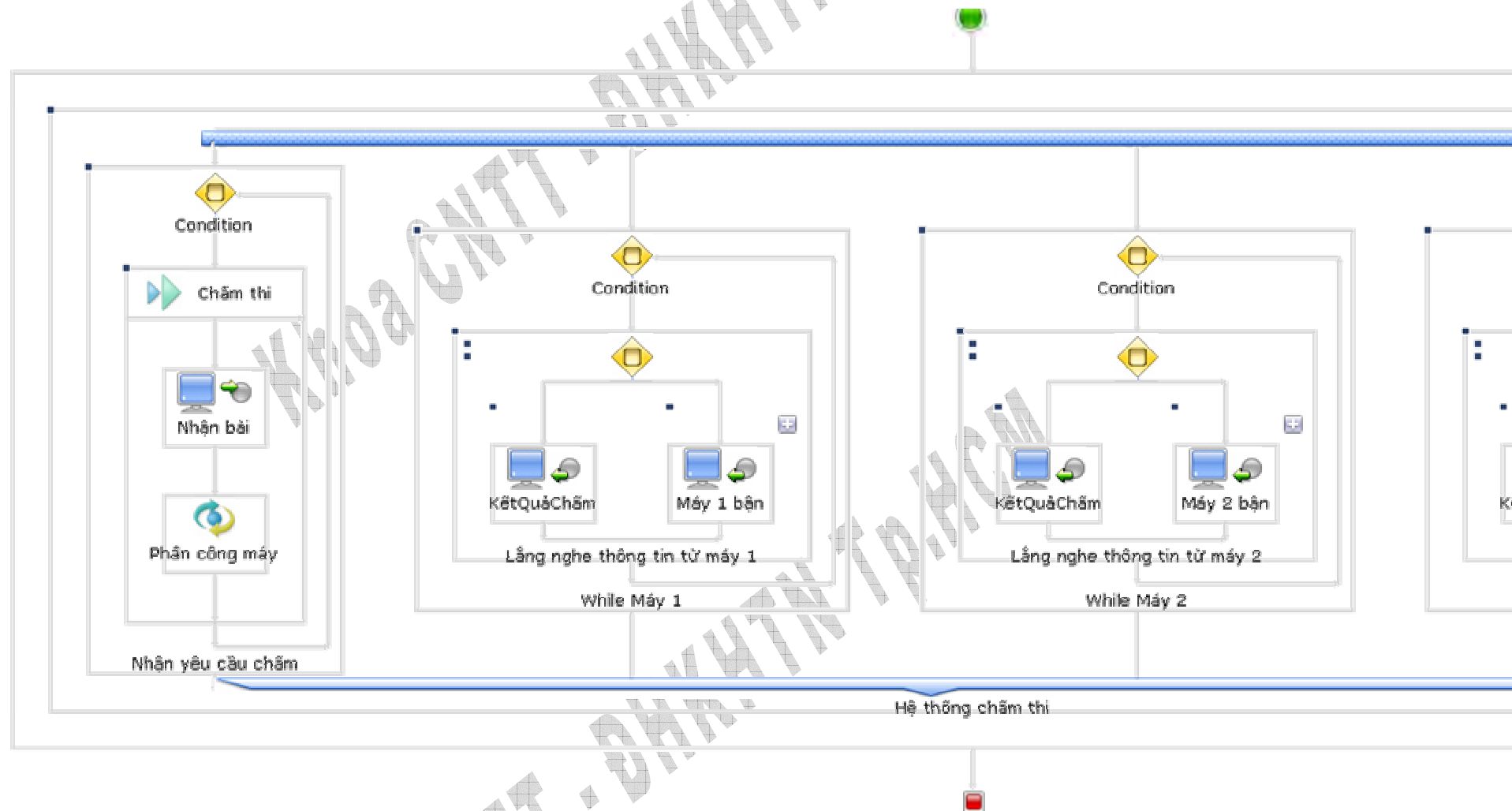
9.3 Tiến trình chấm thi tự động qua mạng

9.3.1 Mô tả

Việc chấm thi ở mỗi kì thi luôn là công việc cực nhọc. Nay công việc đó có phần đỡ nặng nhọc hơn khi đã có một số chương trình chấm thi tự động các bài thi lập trình tin học. Tuy nhiên, các chương trình này lại chạy trên máy đơn và mỗi lần chỉ xử lý được một bài thi. Làm cách nào cho phép nhiều thí sinh nộp bài lên mạng cùng lúc và có kết quả thi ngay sau đó. Công việc này đòi hỏi phải đồng bộ và xử lý song song trên nhiều máy để tăng tốc độ xử lý. Với các kỹ thuật cũ thì sử dụng lại những chương trình chấm thi tự động này cho chúng kết hợp với nhau, xử lý song song là điều hết sức khó khăn. Ta phải thiết kế thành phần giao tiếp qua mạng cho chương trình, đồng bộ và song song, tệ hơn, có thể ta phải viết lại từ đầu !!!

Với SOA thì việc giải bài toán trên trở nên dễ dàng, thậm chí có thể cho thay đổi số lượng máy, địa chỉ máy mà vẫn đảm bảo yêu cầu xử lý trong một thời gian ngắn. Trong bài này, ta giả sử có 3 máy xử lý chấm bài thi cùng lúc.

9.3.2 Sơ đồ



Hình 9-3 – Tiến trình chấm thi tự động

9.3.3 Mô tả luồng xử lý

Mỗi bài thi gửi đến tiến trình chứa thông tin về bài thi: mã bài thi, mã thí sinh, mã môn thi, mã hội đồng, mã kì ,v.v... Ở đây ta sẽ thiết kế xử lý “Nhận bài” (xử lý được xử lý đầu tiên do 3 luồng song song còn lại đều phải chờ) nhận bài thi và chuyển xuống cho xử lý “phân công máy”.

“Phân công máy” là một xử lý phức , nó làm một loạt công việc như sau: xác định máy còn trống, cập nhật danh sách biến trạng thái của các máy và gửi bài thi cho máy trống chấm. Nếu không có máy nào trống thì gửi kết quả lỗi bận về cho người sử dụng.

Ta có 3 luồng song song tương ứng với 3 máy chấm thi cùng lúc. Đây là xử lý song song nên mỗi khi chấm xong máy đó sẽ gửi kết quả trả về cho người sử dụng (kết quả tập trung về một nơi).

Thí sinh có thể xem điểm thông qua trang web được cập nhật thường xuyên.

Chương 10

Kết luận

Chương 10 sẽ trình bày một số kết luận và hướng phát triển của đề tài.

10.1 Một số kết quả đạt được

Qua tìm hiểu và nghiên cứu đề tài, chúng em đã nắm được các cơ sở lý thuyết về kiến trúc hướng dịch vụ (SOA), bao gồm các khái niệm, các tính chất, và các nguyên tắc thiết kế, cũng như các bước cần thực thi khi xây dựng hệ thống SOA. Rõ ràng là có rất nhiều thách thức, nhiều vấn đề cần quan tâm trong việc ứng dụng giải pháp SOA trong một dự án cụ thể. Thế nhưng những khó khăn này không làm cản trở sự quan tâm của các tổ chức, các nhà doanh nghiệp và các nhà quản lý hệ thống bởi vì chúng quá nhỏ bé so với những giá trị thiết thực mà một hệ thống SOA đem lại nếu được triển khai thành công.

SOA thật sự là một kiến trúc “lý tưởng” cho các hệ thống quản lý của các tổ chức, các doanh nghiệp. Với kết cấu mở, linh hoạt, khả năng dễ mở rộng và tính liên kết cao làm cho hệ thống SOA thật sự có “sức đề kháng” cao đối với những rủi ro về sự thay đổi xảy ra trong môi trường hoạt động nghiệp vụ của các tổ chức. Thay đổi là yếu tố không thể thiếu trong các hoạt động nghiệp vụ nhằm nâng cao hiệu quả và chất lượng phục, đặc biệt là trong môi trường cạnh tranh ngày nay. Một hệ thống SOA khi đó có thể dễ dàng tùy biến để đáp ứng nhanh chóng các yêu cầu thay đổi với mức chi phí thấp hơn rất nhiều so với các giải pháp trước đây. ***SOA làm được điều này bởi 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.***

Để hỗ trợ việc thiết kế và triển khai một hệ thống SOA phần nào được dễ dàng và hiệu quả hơn, bộ công cụ **SOASuite** được xây dựng với 3 thành phần sau:

- Hệ thống quản lý dịch vụ **ServiceBus** tạo ra môi trường giao tiếp giữa các dịch vụ thông qua cơ chế trao đổi thông điệp. ServiceBus, với cơ chế quản lý linh hoạt và dễ mở rộng dựa trên nguồn cơ sở tri thức, còn cho phép tùy biến các chức năng xử lý của các dịch vụ thông qua cơ chế sử dụng các bộ lọc. Ngoài ra, các dịch vụ này còn có thể được ghép vào hay lấy ra serviceBUS một cách dễ dàng bằng cách thay đổi thông tin tri thức của dịch vụ.
- **BpelEngine** cung cấp môi trường triển khai, thực thi và quản lý các tiến trình nghiệp vụ. Ngoài ra, BpelEngine còn cung cấp chức năng các tiến trình ra bên ngoài như những web service. Mỗi tiến trình có thể xem như là sự kết hợp có qui trình của các dịch vụ. Như vậy, với BpelEngine ta có thể hiện thực hóa ý tưởng: *mỗi hệ thống, mỗi phần mềm, mỗi chức năng có thể coi như một dịch vụ; và các dịch vụ này có thể lắp ghép, tùy biến để cho ra các dịch vụ tích hợp theo những yêu cầu khác nhau.*
- Bộ công cụ hỗ trợ thiết kế BpelDesigner cung cấp một môi trường tiện dụng và hiệu quả trong việc thiết kế và xây dựng các tiến trình nghiệp vụ thông qua các thao tác đơn giản **kéo và thả**.

Với những kết quả tìm hiểu được, chúng em tin rằng SOA thực sự là một giải pháp tốt để giải quyết các vấn đề khó khăn mà các giải pháp trước chưa xử lý được, và sẽ trở thành nền tảng mang tính chiến lược của các tổ chức, doanh nghiệp trong tương lai.

10.2 Hướng phát triển

Về mặt lý thuyết, sẽ nghiên cứu sâu hơn về vấn đề bảo mật trong hệ thống SOA, xem xét các giải pháp cụ thể trong vấn đề dịch vụ hóa và tích hợp mở rộng cho các loại hệ thống xây dựng dựa trên các công nghệ trước như DCOM, CORBA, J2EE... Ngoài ra, có thể tìm hiểu thêm về cách kết hợp giữa SOA và “Tính toán lưới” (Grid

Computing) để nâng cao hiệu suất hoạt động của các hệ thống SOA, vì khi đó các hệ thống SOA sẽ kế thừa được sức mạnh tính toán của mô hình “Tính toán lưới”.

Về mặt ứng dụng, sẽ nghiên cứu để xây dựng thêm các thành phần hỗ trợ vấn đề bảo mật cho môi trường tương tác của các dịch vụ. Thêm vào đó, tăng cường chức năng cho BpelEngine để lưu lại trạng thái của các tiến trình trong bộ nhớ phụ, hiện tại thì toàn bộ thông tin này chỉ được lưu trong bộ nhớ chính. Điều này thật sự cần thiết trong việc làm giảm hậu quả nếu có sự cố xảy ra trong khi các tiến trình đang thực hiện.

Tài liệu tham khảo

Tiếng Anh

- [1] IBM Red Book Team (2004), *Pattern: Implementing an SOA using Enterprise Service Bus*.
- [2] Carl-Fredrik Sorensen (2004), *A Comparison of Distributed Object Technologies*
- [3] Ludwig Wittgenstein (2002), *Distributed Application Architecture*
- [4] Sriram Krishnan (2004), *An Architecture for Checkpointing and Migration of Distributed Components on the Grid*
- [5] ZapThink foundation report (2003), *The “Pros and Cons” of Web Services*
- [6] IBM Red Book Team (2004), *Pattern: Service-Oriented Architecture and Web Services*.
- [7] ZapThink foundation report (2003), *Service-Oriented Architecture : Why and How*
- [8] BEA Technical Paper (2004), *IT Transformation To Service-Oriented Architecture*
- [9] Web Service Group (2004), *Web Services Coordination*
- [10] Web Service Group (2004), *Web Services Business Activity Framework*
- [11] Web Service Group (2004), *Web Services Atomic Transaction*
- [12] Software Engineer IBM (2002), *Business Process with BPEL4WS: Learning BPEL4WS*
- [13] Lauri Jaakkola (2005), *Applying Service-Oriented Architecture to Geographically Distributed Industrial Information System*
- [14] Ed Ort (2005), *Service-Oriented Architecture and Web Services: Concepts, Technologies and Tools*
- [15] Hartwig Gunzer (2002), *Introduction to Web Services*
- [16] Lawrence Wilkes (2005), *ROI the Costs and Benefits of Web Services and Service-Oriented Architecture*
- [17] BEA (2004), *Service-Oriented Architecture Solution Accelerator Guide*
- [18] JWSA (2004), *Service-Oriented Architecture*
- [19] Software Maker (2004), *SOA*
- [20] Steven Wilkes (2004), *Loosen Up*
- [21] *Cross Platform Intergration : Key technologies Involved*

- [22] Eric Newcomer, Greg Lomow ,*Understanding SOA with Web Services*.
- [23] A Meta Group White paper, *Intersection of Web services and Security Management: A Service-Oriented Security Architecture*
- [24] Adam Zilinskas, Morris Brown, Brian Loomis, *Securing B2B Xml Web services with WSE*
- [25] Martin Bjurhager, *Security Systems for Business Transactions based on Web Services*
- [26] Defense Information Systems Agency (DISA) , *A Security architecture for net-centric enterprise services*
- [27] *Web service Security*.
- [28] Jan Poortinga, *Oracle Integration*
- [29] Microsoft MSDN, *Enterprise Application Integration Overview*
- [30] *Analysis of Middleware Integration - Forces Affecting the Adoption of Web Services and Other Integration Techniques*
- [31] IconMedialab, *Service Oriented Architecture & Web Services*
- [32] IBM, *Migrating to a service-oriented architecture, Part 2*
- [33] Nick Kall, *Service-Oriented Security Architecture: Part 1*
- [34] XML-Based Integration with XAware, *Integration Technologies*
- [35] W3C, *Web Service Choreography Interface (WSCI) 1.0*
- [36] H.M. ter Hofstede, *Pattern Based Analysis of BPEL4WS*
- [37] *Business Process Execution Language for Web Services*
- [38] Technical Staff of Systinet Corporation, *Web Services: the Right Way to Implement a Service-Oriented Architecture*
- [39] Yefim V. Natis, *Service-Oriented Architecture Scenario*
- [40] Systinet, *A Practical Guide to SOA for IT Management*

Phụ lục A

ĐẶC TẢ NGÔN NGỮ BPEL V1.1**A.1 Định nghĩa một tiến trình nghiệp vụ (business process)****A.1.1 Cấu trúc của một tiến trình nghiệp vụ:**

Cấu trúc cơ bản của một tiến trình như sau:

Một tiến trình được thể hiện bằng một tài liệu XML có nút gốc là `<process>`.

Các thuộc tính của nút này bao gồm:

- *queryLanguage*: chỉ định ngôn ngữ truy vấn XML sẽ được dùng trong việc chọn các nút ở các thao tác gán, định nghĩa property... Giá trị mặc định cho thuộc tính này là XPath1.0.
- *expressionLanguage*: chỉ định ngôn ngữ biểu thức dùng trong tiến trình. Giá trị mặc định là XPath1.0
- *suppressJoinFailure*: chỉ định bỏ qua hay xử lý lỗi joinFailure cho tất cả các xử lý trong tiến trình. Giá trị này có thể được thay đổi tại mỗi xử lý. Giá trị mặc định là “no”.
- *enableInstanceCompensation*: Giá trị mặc định là “no”.
- *abstractProcess*: chỉ định tiến trình có được định nghĩa là trừu tượng hay không? Giá trị mặc định là “no” (đây là một tiến trình thực thi).

`<activity>` có thể là một trong các xử lý sau: `<receive>`, `<reply>`, `<invoke>`, `<assign>`, `<throw>`, `<terminate>`, `<wait>`, `<empty>`, `<sequence>`, `<switch>`, `<while>`, `<pick>`, `<flow>`, `<scope>`, `<compensate>`.

```
<process name="ncname" targetNamespace="uri" queryLanguage="anyURI"?
  expressionLanguage="anyURI"? suppressJoinFailure="yes|no"?
  enableInstanceCompensation="yes|no"? abstractProcess="yes|no"?
  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/">
  <partnerLinks>?
    <!-- Note: At least one role must be specified. -->
    <partnerLink name="ncname" partnerLinkType="qname"
      myRole="ncname"? partnerRole="ncname"?>+
    </partnerLink>
```

```

</partnerLinks>
<partners>?
  <partner name="ncname">+
    <partnerLink name="ncname"/>+
  </partner>
</partners>
<variables>?
  <variable name="ncname" messageType="qname"?
    type="qname"? element="qname"?/>+
</variables>
<correlationSets>?
  <correlationSet name="ncname" properties="qname-list"/>+
</correlationSets>
<faultHandlers>?
<!-- Note: There must be at least one fault handler or default. -->
  <catch faultName="qname"? faultVariable="ncname"?*>
    activity
  </catch>
  <catchAll>?
    activity
  </catchAll>
</faultHandlers>
<compensationHandler>?
  activity
</compensationHandler>
<eventHandlers>?
<!-- Note: There must be at least one onMessage or onAlarm handler -->
  <onMessage partnerLink="ncname" portType="qname"
    operation="ncname" variable="ncname"?>
    <correlations>?
      <correlation set="ncname" initiate="yes|no"?>+
    </correlations>
    activity
  </onMessage>
  <onAlarm for="duration-expr"? until="deadline-expr"?*>
    activity
  </onAlarm>
</eventHandlers>

  activity
</process>

```

<receive> sẽ cho phép tiến trình dừng thực thi và chờ nhận một thông điệp thích hợp.

```

<receive partnerLink="ncname" portType="qname" operation="ncname"
variable="ncname"? createInstance="yes|no"? standard-attributes>
  standard-elements
  <correlations>?
    <correlation set="ncname" initiate="yes|no"?>+
  </correlations>
</receive>

```

<reply> cho phép tiến trình nghiệp vụ gửi thông điệp trả lời cho thông điệp đã nhận trước đó bởi **<receive>**. Sử dụng kết hợp **<receive>** và **<reply>** thì tương ứng với một request-response operation định nghĩa trong file WSDL.

```
<reply partnerLink="ncname" portType="qname" operation="ncname"
variable="ncname"? faultName="qname"? standard-attributes>
  standard-elements
  <correlations>?
    <correlation set="ncname" initiate="yes|no"?>+
  </correlations>
</reply>
```

<invoke> cho phép tiến trình gọi một operation của một portType cung cấp bởi một partner

```
<invoke partnerLink="ncname" portType="qname" operation="ncname"
inputVariable="ncname"? outputVariable="ncname"? standard-attributes>
  standard-elements
  <correlations>?
    <correlation set="ncname" initiate="yes|no"?
      pattern="in|out|out-in"/>+
  </correlations>
  <catch faultName="qname" faultVariable="ncname"?>*
    activity
  </catch>
  <catchAll>?
    activity
  </catchAll>
  <compensationHandler>?
    activity
  </compensationHandler>
</invoke>
```

<assign> cho phép thực hiện gán giá trị vào một biến. Ta có thể thực hiện nhiều phép gán trong xử lý này.

```
<assign standard-attributes>
  standard-elements
  <copy>+
    from-spec
    to-spec
  </copy>
</assign>
```

<throw> cho phép ném lỗi từ bên trong một tiến trình .

```
<throw faultName="qname" faultVariable="ncname"? standard-attributes>
  standard-elements
</throw>
```

<wait> cho phép dừng xử lý trong bao lâu hay tới một thời điểm nào đó

```
<wait (for="duration-expr" | until="deadline-expr") standard-attributes>
  standard-elements
</wait>
```

<empty> cho phép chèn một xử lý “không làm gì cả” vào tiến trình, điều này cần thiết khi cần phải đồng bộ những xử lý đang được xử lý cùng lúc.

```
<empty standard-attributes>
  standard-elements
</empty>
```

<sequence> cho phép chỉ định các xử lý cần được thực hiện tuần tự

```
<sequence standard-attributes>
  standard-elements
  activity+
</sequence>
```

<flow> cho phép chỉ định các xử lý cần được xử lý song song

```
<flow standard-attributes>
  standard-elements
  <links>?
    <link name="ncname">+
  </links>
  activity+
</flow>
```

<switch> cho phép chọn một trong nhiều nhánh xử lý để thực hiện

```
<switch standard-attributes>
  standard-elements
  <case condition="bool-expr">+
    activity
  </case>
  <otherwise>?
    activity
  </otherwise>
</switch>
```

<while> cho phép lặp lại quá trình thực hiện một xử lý trong khi điều kiện lặp còn được thỏa.

```
<while condition="bool-expr" standard-attributes>
  standard-elements
  activity
</while>
```

<pick> cho phép tiến trình dừng và chờ cho đến khi nhận được một thông điệp thích hợp hay cho đến khi hết giờ (time-out). Khi nhận được thông điệp thích hợp, thì nhánh xử lý tương ứng sẽ được thực hiện và xử lý **<pick>** kết thúc (nghĩa là với xử lý này chỉ nhiều nhất một nhánh xử lý được thực hiện).

```
<pick createInstance="yes|no"? standard-attributes>
  standard-elements
  <onMessage partnerLink="ncname" portType="qname" operation="ncname"
variable="ncname"?>+
    <correlations?>
      <correlation set="ncname" initiate="yes|no"?>+
    </correlations>
    activity
  </onMessage>
  <onAlarm (for="duration-expr" | until="deadline-expr")>*
    activity
  </onAlarm>
</pick>
```

<scope> cho phép chỉ ra một xử lý cần được thực hiện với những biến, fault handler, compensation handler riêng định nghĩa bên trong nó.

```
<scope variableAccessSerializable="yes|no" standard-attributes>
  standard-elements
  <variables?>
    ... see above under <process> for syntax ...
  </variables>
  <correlationSets?>
    ... see above under <process> for syntax ...
  </correlationSets>
  <faultHandlers?>
    ... see above under <process> for syntax ...
  </faultHandlers>
  <compensationHandler?>
    ... see above under <process> for syntax ...
  </compensationHandler>
  <eventHandlers?>
    ...
  </eventHandlers>
  activity
</scope>
```

<compensate> cho phép gọi thực hiện một compensation handler của một scope đã được xử lý xong. Xử lý này chỉ có thể gọi từ một fault handler hay một compensation handler khác.

```
<compensate scope="ncname"? standard-attributes>
  standard-elements
</compensate>
```

Các standard-attribute được sử dụng trong các xử lý ở trên là:

```
name="ncname"?
joinCondition="bool-expr"?
suppressJoinFailure="yes|no"?
```

giá trị mặc định suppressJoinFailure=no

Các standard-element được sử dụng trong các xử lý trên là

```
<target linkName="ncname"/>*
<source linkName="ncname" transitionCondition="bool-expr"?/>*
```

giá trị mặc định transitionCondition=true

A.1.2 Chu kỳ sống của một tiến trình nghiệp vụ

Mô hình tương tác được trực tiếp hỗ trợ bởi WSDL về cơ bản là giao tiếp client/server phi trạng thái với giao tiếp đồng bộ hoặc bất đồng bộ không tương quan (uncorrelated asynchronous). Ngôn ngữ BPEL4WS, được xây dựng dựa trên WSDL bằng cách xem mọi tương tác ra bên ngoài của một tiến trình nghiệp vụ đều thông qua các phương thức của Web service. Tuy nhiên, BPEL4WS cho phép xây dựng các tiến trình nghiệp vụ với các tương tác trạng thái (stateful) và thời gian xử lý lâu. Trong mỗi tương tác đều có một bắt đầu và một kết thúc.

Ví dụ như, trong một hoạt động cung cấp hàng, tiến trình nghiệp vụ xử lý bán hàng sẽ bắt đầu quá trình tương tác bằng việc nhận một đơn đặt hàng thông qua một thông điệp đầu vào, và sau đó trả về cho người mua một xác nhận nếu đơn đặt hàng đó có thể đáp ứng. Sau đó, tiến trình này thực hiện các xử lý của mình, và nó có thể gửi tiếp cho khách hàng một vài thông tin khác như biên nhận, hóa đơn. Như vậy, tiến trình đòi hỏi cần phải nhớ trạng thái của từng phiên giao dịch như thế. Điều này là cần thiết vì có thể một khách hàng có thể cùng lúc thực hiện nhiều phiên giao dịch với cùng một nhà cung cấp.

Từ định nghĩa của một tiến trình có thể tạo ra nhiều thể hiện của tiến trình. Quá trình tạo một thể hiện của tiến trình luôn luôn được thực hiện “ngầm” (implicit). Những xử lý có khả năng nhận thông điệp (như receive và pick) có thể được chỉ định là chúng là “điểm nút” (entry point) để tạo một thể hiện mới của tiến trình. Điều này được thực hiện bằng cách đặt giá trị của thuộc tính “createInstance” là “yes”. Khi một thông điệp được nhận bởi một xử lý như thế, thì một thể hiện của tiến trình sẽ được tạo ra nếu như nó chưa thật sự tồn tại.

Như vậy, để được khởi tạo thì mỗi tiến trình cần phải chứa ít nhất một xử lý “khởi động”. Xét về mặt xử lý thì đây phải là xử lý đầu tiên được thực hiện trong qui trình xử lý của tiến trình.

Nếu có nhiều xử lý “khởi động” được dùng để khởi tạo process, thì các correlation sets không nhất thiết phải được sử dụng. Ví dụ như một xử lý <pick> với nhiều nhánh “onMessage”, thì mỗi nhánh có thể sử dụng một “correlation sets” khác nhau, hay là không sử dụng cũng được.

Một thể hiện của tiến trình được kết thúc theo một trong các cách sau:

- Khi xử lý định nghĩa phần xử lý của tiến trình đã thực thi xong (một tiến trình chỉ có một xử lý). Đây là sự kết thúc bình thường.
- Khi mà một lỗi (fault) được ném ra tới phạm vi của tiến trình (không quan tâm lỗi này có được xử lý hay không). Đây là trường hợp kết thúc không bình thường (ngay cả khi lỗi đó đã được xử lý). Một compensation handler không bao giờ được xét đến trong một scope mà kết thúc không bình thường như thế.
- Khi thể hiện đó được yêu cầu kết thúc bằng xử lý <terminate>. Đây cũng là trường hợp kết thúc không bình thường.

A.2 Partner, Partner Link Type, và Partner Link

A.2.1 Partner

Vì là một ngôn ngữ dùng để kết hợp nhiều dịch vụ lại với nhau để tạo một dịch vụ tổng hợp hoặc tiến trình, nên trong qui trình xử lý của tiến trình có gọi đến các dịch vụ và/hoặc nhận lời gọi từ các client (đối tượng sử dụng dịch vụ cung cấp bởi tiến trình).

Partner hoặc là các dịch vụ mà tiến trình gọi (invoked partner), hoặc là các đối tượng mà sẽ gọi tiến trình (client partner).

- Loại partner thứ nhất (invoked partners) rất rõ ràng. Xử lý <invoke> sẽ chỉ ra partner cần gọi, bao gồm operation nào và của portType nào?
- Loại partner thứ hai (client partners) thì không dễ hiểu như thế. Tại sao process lại đối xử với client như là partner? Có hai lý do giải thích điều này:
 - Lý do thứ nhất: có những lúc tiến trình cần gọi những phương thức của các client partner. Đây trường hợp giao tiếp bất đồng bộ. Ví dụ như, client yêu cầu sử dụng dịch vụ của tiến trình, sau khi tiến trình xử lý yêu cầu xong, nó sẽ gọi một phương thức của client partner để thông báo kết quả. Khi này không có sự phân biệt giữa một client partner và một invoked partner.
 - Lý do thứ hai: dịch vụ cung cấp bởi tiến trình có thể được sử dụng bởi nhiều client. Bên cạnh đó, tiến trình có thể muốn có phân biệt với những đối tượng dùng khác nhau này. Ví dụ như, với những đối tượng sử dụng khác nhau thì nó sẽ có những cách “đối xử” khác nhau.

Vì thế, partner có thể là:

- Những dịch vụ mà chỉ để tiến trình gọi (pure invoked partners)
- Những dịch vụ mà chỉ gọi tiến trình (pure client partners)
- Những dịch vụ mà gọi tiến trình và được tiến trình gọi.

Hai trường hợp đầu đã rõ ràng. Ta sẽ xem xét mối quan hệ giữa tiến trình và dịch vụ trong trường hợp thứ 3:

- Tiến trình gọi dịch vụ nghĩa là dịch vụ sẽ cung cấp portType (PT1) và tiến trình gọi operation của portType đó. Dịch vụ gọi tiến trình nghĩa là tiến trình sẽ cung cấp portType (PT2) cho service.
- Nếu đứng trên góc nhìn của tiến trình, thì ta nói rằng “tiến trình cần PT1 của dịch vụ và cung cấp PT2 cho dịch vụ”. Và nếu đứng trên góc nhìn của dịch vụ, thì phát biểu sẽ ngược lại.

A.2.2 Partner Link Type

Thay vì định nghĩa mối liên hệ giữa tiến trình và dịch vụ từ góc nhìn của mỗi đối tượng, Partner link type sẽ biểu diễn mối quan hệ này một cách “khách quan” hơn. Partner link type biểu diễn mối quan hệ này bằng cách định nghĩa các “role” mà các đối tượng sẽ thể hiện trong quá trình tương tác. Mỗi role sẽ có một portType. Đây sẽ là portType mà mỗi bên cung cấp trong quá trình tương tác.

```
<definitions name="ncname" targetNamespace="uri"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
...
  <plnk:partnerLinkType name="ncname">
    <plnk:role name="ncname">
      <plnk:portType name="qname" />
    </plnk:role>
    <plnk:role name="ncname">?
      <plnk:portType name="qname" />
    </plnk:role>
  </plnk:partnerLinkType>
...
</definitions>
```

Trong những trường hợp “pure client partner” và “pure invoke partner” thì partner link type chỉ có một role

A.2.3 Partner Link

Trong BPEL4WS, các partner được biểu diễn bởi partner link. Một partner link được định nghĩa bằng cách đặt cho nó một cái tên, chỉ ra tên của partner link type, và sau

đó chỉ ra “role” mà tiến trình sẽ tham gia và “role” mà partner sẽ tham gia trong partner link type đó.

```
<partnerLinks>
  <partnerLink name="ncname" partnerLinkType="qname" myRole="ncname"?
partnerRole="ncname"?>+
  </partnerLink>
</partnerLinks>
```

Role của bản thân tiến trình nghiệp vụ được chỉ định bởi thuộc tính “myRole” và role của partner được chỉ định bởi thuộc tính “partnerRole”. Trong trường hợp partner link type chỉ có một role thì các thuộc tính trên có thể được bỏ qua.

A.3 Xử lý dữ liệu

Mô hình tương tác của tiến trình nghiệp vụ là có trạng thái (stateful). Thông tin trạng thái bao gồm các thông điệp nhận và gửi, cùng với các dữ liệu liên quan khác như giá trị time-out. Việc duy trì trạng thái của tiến trình đòi hỏi phải sử dụng các variable (biến).

Các dữ liệu của các thông tin trạng thái này cần được rút trích và kết hợp theo nhiều cách để điều khiển luồng xử lý của tiến trình. Các thao tác trên dữ liệu này cần các “biểu thức dữ liệu” (data expression). Ngoài ra, các thông tin trạng thái này cần được thay đổi, cập nhật trong quá trình thực thi của tiến trình nên cần sự hỗ trợ của các phép gán (assignment).

BPEL4WS hỗ trợ các tính năng này trên các kiểu dữ liệu XML và những kiểu message type được định nghĩa trong WSDL.

A.3.1 Biểu thức

BPEL4WS sử dụng rất nhiều kiểu biểu thức (expression). Các loại biểu thức được sử dụng bao gồm:

- Biểu thức cho giá trị luận lý (Boolean-value expression). Loại biểu thức này được sử dụng trong transitionCondition, joinCondition, while và switch-cases.
- Biểu thức cho giá trị thời gian tối hạn (dùng trong thuộc tính “until” của onAlarm và wait)

- Biểu thức cho giá trị khoảng thời gian (dùng trong thuộc tính “for” của onAlarm và wait).
- Các biểu thức tổng quát (dùng trong các phép gán.):
 - ▶ Các biểu thức dạng này theo cú pháp của XPath1.0. Kết quả trả về sẽ là một trong các kiểu giá trị của XPath (string, number hay boolean).
 - ▶ Được dùng giá trị kiểu số với các toán tử <, <=, =, !=, >=, >
 - ▶ Có thể thực hiện các phép tính số học với các giá trị số nguyên.
 - ▶ Đối với giá trị chuỗi thì chỉ được sử dụng các phép so sánh =, !=

BPEL4WS cung cấp cơ chế mở rộng trong việc lựa chọn ngôn ngữ trong các biểu thức này. Loại ngôn ngữ được chỉ định thông qua thuộc tính “expressionLanguage” của <process>. Ngôn ngữ XPath1.0 là ngôn ngữ mặc định dùng trong các biểu thức của BPEL4WS.

BPEL4WS cung cấp nhiều hàm mở rộng bên cạnh các hàm chuẩn của XPath để cho phép các biểu thức XPath có thể truy cập thông tin từ tiến trình. Các mở rộng được định nghĩa trong namespace chuẩn của BPEL4WS <http://schema.xmlsoap.org/ws/2003/03/business-process/> và tiền tố “bpws:” được sử dụng kết hợp với namespace này.

Mọi qualified name - QName (là những tên có dạng prefix:localname) được sử dụng trong các biểu thức XPath đều được phân giải bằng cách sử dụng các namespace được khai báo trong đặc tả tiến trình.

Các hàm sau đây được định nghĩa bởi đặc tả BPEL4WS:

```
bpws:getVariableProperty('variableName', propertyName')
```

- Hàm này trích dữ liệu cho property từ một variable. Đối số đầu tiên là tên của biến chứa dữ liệu và đối số thứ hai là QName của property cần lấy dữ liệu. Giá trị trả về của hàm là một nodeset chỉ chứa một nút là giá trị của property.

```
bpws:getLinkStatus ('linkName')
```

- Hàm này trả về giá trị kiểu Boolean cho biết tình trạng của link. Nếu tình trạng của link đã được kích hoạt thì trả về giá trị true, ngược lại trả về false. Biểu thức này chỉ được sử dụng trong joinCondition. Giá trị của linkName phải tham chiếu đến tên của một trong các incoming link (<target>) của xử lý gắn với joinCondition.

A.3.2 Variable (biến)

Quá trình tương tác của tiến trình nghiệp vụ với các partners là có trạng thái, trong đó các thông điệp được trao đổi qua lại. Trạng thái của một tiến trình bao gồm:

- Các thông điệp được trao đổi
- Các dữ liệu trung gian dùng trong quá trình xử lý và trong việc tạo các thông điệp để gửi cho partner.

Các biến cung cấp cách thức lưu trữ các thông điệp sử dụng trong quá trình trao đổi giữa tiến trình và các partner. Các biến cũng hỗ trợ lưu trữ các dữ liệu trạng thái liên quan đến tiến trình và không bao giờ được trao đổi giữa partners.

Kiểu của mỗi biến có thể là:

- Kiểu message type được định nghĩa trong WSDL.
- Kiểu lược đồ Xml đơn giản (int, string, bool,)
- Kiểu thành phần lược đồ Xml.

Cú pháp khai báo các biến này như sau:

```
<variables>
  <variable name="ncname" messageType="qname"? type="qname"?
  element="qname"?/>+
</variables>
```

- Tên của biến phải duy nhất trong phạm vi khai báo của nó. Nếu một biến cục bộ có cùng tên và cùng kiểu với một biến khác đã được khai báo ở phạm vi bên ngoài, thì biến cục bộ sẽ chỉ được dùng trong các phép gán cục bộ. BPEL4WS không cho phép khai báo các biến cùng tên nhưng khác kiểu bên trong phạm vi khai báo.

- Các thuộc tính messageType, type hay element được dùng để chỉ ra kiểu của biến. Chính xác là chỉ duy nhất một trong các thuộc tính này được sử dụng.
 - ▶ messageType tham chiếu đến một kiểu messageType định nghĩa trong WSDL
 - ▶ type tham chiếu đến kiểu lược đồ Xml đơn giản.
 - ▶ element tham chiếu đến thành phần lược đồ Xml.

Một ví dụ về khai báo một biến với kiểu messageType được khai báo trong tài liệu WSDL với targetNamespace= “<http://example.com/orders>”

```
<variable xmlns:ORD="http://example.com/orders"
          name="orderDetails" messageType="ORD:orderDetails"/>
```

Các biến có kiểu message type có thể được dùng như các biến đầu vào và ra của các xử lý như invoke, receive và reply. Khi gọi một service trả về thông điệp lỗi thì tạo ra một fault trong scope hiện hành, và biến fault trong fault handler sẽ được gán với giá trị là thông điệp lỗi nhận được.

Mỗi biến được khai báo trong một scope và được gọi là thuộc về scope đó. Các biến được khai báo trong scope của tiến trình được gọi là biến toàn cục (global variable). Mỗi biến chỉ có thể được hiểu trong scope mà nó khai báo và trong tất cả các scope lồng bên trong scope mà nó thuộc về. Vì thế các biến toàn cục được hiểu trong suốt tiến trình. Ta có thể “che dấu” một biến ở scope bên ngoài bằng cách khai báo một biến trùng tên (trùng kiểu) ở scope bên trong.

Các biến toàn cục ở trạng thái chưa được khởi tạo khi bắt đầu tiến trình. Một biến cục bộ ở trạng thái chưa được khởi tạo khi bắt đầu scope mà nó thuộc về. Có nhiều cách để khởi tạo một biến bằng các phép gán hay bằng cơ chế nhận thông điệp. Các biến cũng có thể được khởi tạo từng phần bằng các phép gán property hay khi chỉ thực hiện gán một số part của biến (một messageType có nhiều part).

A.4 Phép gán

Sao chép dữ liệu từ một biến này sang một biến khác là một công việc thường làm bên trong một tiến trình. Xử lý <assign> có thể được dùng để thực hiện việc này. Ngoài ra nó còn cho phép khởi tạo và chèn thêm dữ liệu mới bằng cách sử dụng các biểu thức. Các biểu thức được sử dụng chủ yếu từ nhu cầu muốn thực hiện các tính toán đơn giản (như là tăng tuần tự một số). Cuối cùng, xử lý này cũng có thể được dùng để sao chép địa chỉ của service giữa những partner link.

Một xử lý <assign> có thể chứa một hay nhiều phép gán.

```
<assign standard-attributes>
  standard-elements
  <copy>+
    from-spec
    to-spec
  </copy>
</assign>
```

Xử lý <assign> các kiểu giá trị có kiểu tương thích nhau từ “from-spec” đến “to-spec”.

- Nguồn phải là một trong các dạng sau:

```
<from variable="ncname" part="ncname"?/>
<from partnerLink="ncname" endpointReference="myRole|partnerRole"/>
<from variable="ncname" property="qname"/>
<from expression="general-expr"/>
<from> ... literal value ... </from>
```

- Đích phải là một trong các dạng sau:

```
<to variable="ncname" part="ncname"?/>
<to partnerLink="ncname"/>
<to variable="ncname" property="qname"/>
```

Ở dạng đầu tiên của nguồn và đích:

- Nếu kiểu của biến là WSDL messageType thì thuộc tính tùy chọn “part” có thể được dùng để chỉ ra tên của part bên trong biến đó.
- Nếu kiểu của biến là Xml Schema simple type hay element, thì thuộc tính “part” không được dùng.

Ở dạng thứ hai của nguồn và đích cho phép thao tác trên địa chỉ dịch vụ của các partner link. Giá trị của thuộc tính “partnerLink” là tên của partner link được khai báo trong tiến trình.

- Đối với nguồn, ta phải chỉ định role là “myRole” (địa chỉ của tiến trình sẽ là nguồn) hay “partnerRole” (địa chỉ của partner link sẽ là nguồn).
- Đối với đích, phép gán chỉ có thể thực hiện cho “partnerRole”, do đó không cần phải chỉ định giá trị của role.

Ở dạng thứ ba cho phép thực hiện các thao tác trên các message property.

Ở dạng thứ tư của nguồn cho phép tiến trình thực hiện các tính toán đơn giản trên property và biến.

Ở dạng thứ năm cho phép gán một giá trị hằng cho đích. Kiểu của giá trị hằng phải cùng kiểu với đích. Kiểu của giá trị hằng có thể được khai báo “bên trong” với giá trị bằng cách sử dụng cơ chế kiểu thể hiện của lược đồ Xml (ví dụ: xsi:type).

Kiểm trong phép gán phải tương thích

- Một phép gán hợp lệ khi dữ liệu được tham chiếu bởi nguồn và đích phải có kiểu tương thích nhau. Trong một số trường hợp đặc biệt, kiểu của nguồn và đích là Xml schema type hay element và ràng buộc là giá trị của nguồn phải “nằm trong” type hay element của đích thì kiểu của nguồn và đích khác nhau. Cụ thể, kiểu của nguồn có thể là kiểu con (subtype) của đích. Trong trường hợp các biến được định nghĩa bằng cách tham chiếu đến một element, thì source và destination phải cùng một element.

Ví dụ về phép gán

Ví dụ giả sử rằng complexType sau được định nghĩa trong namespace

<http://tempuri.org/bpws/example>

```
<complexType name="tAddress">
  <sequence>
    <element name="number" type="xsd:int"/>
    <element name="street" type="xsd:string"/>
    <element name="city" type="xsd:string"/>
    <element name="phone"/>
  </sequence>
</complexType>
```

```

        <complexType>
          <sequence>
            <element name="areacode" type="xsd:int"/>
            <element name="exchange" type="xsd:int"/>
            <element name="number" type="xsd:int"/>
          </sequence>
        </complexType>
      </element>
    </sequence>
  </complexType>
<element name = "address" type = "tAddress"/>

```

Và giả sử rằng các messageType sau được định nghĩa trong cùng namespace

```

<message name="person" xmlns:x="http://tempuri.org/bpws/example">
  <part name="full-name" type="xsd:string"/>
  <part name="address" element="x:address"/>
</message>

```

Ta có các biến được khai báo như sau:

```

<variable name="c1" messageType="x:person"/>
<variable name="c2" messageType="x:person"/>
<variable name="c3" element="x:address"/>

```

Ta sẽ thực hiện sao chép một biến sang một biến khác, đồng thời cũng sao chép một part của biến này sang một biến khác có kiểu tương thích.

```

<assign>
  <copy>
    <from variable="c1"/>
    <to variable="c2"/>
  </copy>
  <copy>
    <from variable="c1" part = "address"/>
    <to variable="c3"/>
  </copy>
</assign>

```

A.5 Các xử lý cơ bản

A.5.1 Các thuộc tính cơ sở của mỗi xử lý

Mỗi xử lý có một số thuộc tính cơ sở tùy chọn: tên, joinCondition và suppressionJoinFailure. Phần giải thích về hai thuộc tính joinCondition và suppressionJoinFailure sẽ được nói rõ trong phần nói về flow.

```

name="ncname"?
joinCondition="bool-expr"?

```

```
suppressJoinFailure="yes|no"?
```

Giá trị mặc định của `suppressJoinFailure` là “no”.

Giá trị mặc định của `joinCondition` là biểu thức thực hiện các phép OR của tình trạng các link đầu vào của xử lý này.

A.5.2 Các thành phần cơ sở của mỗi xử lý

Mỗi xử lý trong BPEL có hai thành phần cơ sở tùy chọn là `<source>` và `<target>`. Các thành phần này được dùng để thiết lập các quan hệ đồng bộ, các ràng buộc về trình tự xử lý thông qua việc sử dụng các link (Xem phần flow).

- Mỗi link được xác định bởi một tên và được định nghĩa một cách độc lập. Tên của link được sử dụng như là giá trị của thuộc tính “`linkName`” của thành phần `<source>` và `<target>`.
- Một xử lý có thể tự khai báo như là *source* của một hay nhiều link bằng cách chứa trong nó một hay nhiều thành phần `<source>` (trường hợp chứa nhiều thì các `<source>` phải có tên link khác nhau).
- Tương tự như thế, mỗi xử lý **có thể** tự khai báo nó như là *target* của một hay nhiều link bằng cách chứa trong nó một hay nhiều `<target>` (trường hợp chứa nhiều thì các `<target>` phải có tên link khác nhau).
- Mỗi `<source>` có thể khai báo thêm thuộc tính tùy chọn là `transitionCondition` với giá trị là một biểu thức bool. Thuộc tính này sử dụng như là một điều kiện để quyết định xem link này đủ kiện kích hoạt hay chưa? Giá trị mặc định của thuộc tính này là “true”(Khi một xử lý kết thúc bình thường thì các link sẽ coi như được kích hoạt nếu đủ điều kiện).

```
<source linkName="ncname" transitionCondition="bool-expr"?/>*
<target linkName="ncname"/>*
```

A.5.3 Gọi một phương thức của Web Service (Invoke)

Các web service được cung cấp bởi các partner và được dùng để thực hiện các tác vụ trong một tiến trình nghiệp vụ. Gọi một phương thức của một web service được thực

hiện bởi xử lý <invoke>. Các phương thức này có thể thuộc dạng request/response đồng đồng bộ hay one-way bất đồng bộ.

Một lời gọi bất đồng bộ chỉ cần biến đầu vào của phương thức bởi vì nó không phải chờ để nhận kết quả trả về của phương thức. Trong khi một lời gọi đồng bộ yêu cầu phải chỉ định cả hai biến vào và ra (in và out).

Một hay nhiều correlation sets có thể được dùng để “định danh” một thể hiện của tiến trình với một dịch vụ có tính trạng thái ở phía partner.

Trong trường hợp của một lời gọi đồng bộ, phương thức có thể trả về một thông điệp lỗi. Thông điệp lỗi này sẽ làm phát sinh một lỗi trong phần xử lý của tiến trình. Nếu lỗi này không được xử lý ở scope hiện hành (bên trong xử lý invoke) thì sẽ được ném ra cho scope chứa xử lý này. Một thông điệp lỗi của web service sẽ được thể hiện trong BPEL4WS target namespace của portType chứa phương thức kèm với tên của lỗi.

Cuối cùng, một xử lý có thể được liên kết với một xử lý khác đóng vai trò như là phần xử lý của một compensation handler.

```
<invoke partnerLink="ncname" portType="qname" operation="ncname"
inputVariable="ncname"? outputVariable="ncname"? standard-attributes>
  standard-elements
  <correlations>?
    <correlation set="ncname" initiate="yes|no"?
pattern="in|out|out-in"/>+
  </correlations>
  <catch faultName="qname" faultVariable="ncname"?*>
    activity
  </catch>
  <catchAll>?
    activity
  </catchAll>
  <compensationHandler>?
    activity
  </compensationHandler>
</invoke>
```

Một ví dụ về xử lý <invoke> với một compensation handler bên trong

```
<invoke partnerLink="Seller" portType="SP:Purchasing"
operation="SyncPurchase" inputVariable="sendPO"
outputVariable="getResponse">
  <compensationHandler>
    <invoke partnerLink="Seller" portType="SP:Purchasing"
operation="CancelPurchase" inputVariable="getResponse"
outputVariable="getConfirmation">
    </compensationHandler>
  </invoke>
```

A.5.4 Cung cấp các phương thức của Web Service

Một tiến trình nghiệp vụ cung cấp dịch vụ cho các partner của nó thông qua các xử lý <receive> và các <reply> tương ứng. Một <receive> chỉ ra partner link mà nó muốn nhận thông điệp từ đó, portType và operation mà các partner sẽ gọi. Thêm vào đó, nó cũng chỉ ra một biến dùng để chứa phần dữ liệu của thông điệp nhận được.

Ngoài ra, các xử lý <receive> còn có vai trò trong chu kỳ sống của một tiến trình nghiệp vụ. Cách duy nhất để khởi tạo một thể hiện của tiến trình đó là dùng xử lý <receive> (hay <pick>) với thuộc tính createInstance có giá trị “yes”. Giá trị mặc định của thuộc tính này là “no”. Xử lý <receive> dạng này phải được xử lý đầu tiên trong tiến trình.

BPEL4WS cũng cho phép khai báo cùng lúc nhiều xử lý khởi tạo như thế trong tiến trình. Trường hợp này được sử dụng khi có nhiều thông điệp có khả năng để khởi tạo tiến trình, nhưng ta không biết trước được thứ tự đến của các thông điệp. Điều lưu ý đó là tất cả các xử lý khởi động này phải sử dụng cùng một correlation sets. Môi trường thực thi process cũng phải đảm bảo rằng chỉ có duy nhất một xử lý khởi động được kích hoạt mà thôi, và các thông điệp đến sau đó (với cùng correlation sets) sẽ được chuyển đến cho thể hiện đã được khởi tạo từ trước.

```
<receive partnerLink="ncname" portType="qname" operation="ncname"
variable="ncname"? createInstance="yes|no"? standard-attributes>
  standard-elements
  <correlations?>
    <correlation set="ncname" initiate="yes|no"?>+
  </correlations>
</receive>
```

Xử lý <reply> được sử dụng để gửi phản hồi cho một yêu cầu đã được chấp nhận trước đó bởi xử lý <receive>. Những phản hồi này chỉ có ý nghĩa trong các tương tác đồng bộ. Một phản hồi trong tương tác bất đồng bộ luôn luôn được gửi thông qua việc gọi một phương thức one-way của partner link.

Xử lý <reply> có thể chỉ ra biến mà sẽ chứa dữ liệu cần được gửi. Điều lưu ý là một <reply> luôn phải được đặt sau một <receive> có cùng partner link, portType và operation.

```
<reply partnerLink="ncname" portType="qname" operation="ncname"
variable="ncname"? faultName="qname"? standard-attributes>
  standard-elements
  <correlations>?
    <correlation set="ncname" initiate="yes|no"?>+
  </correlations>
</reply>
```

Lưu ý: một <reply> có thể có hai dạng.

- Nếu kết quả phản hồi là bình thường, thì thuộc tính “faultName” không được sử dụng và thuộc tính “variable” khi đó sẽ tham chiếu đến một biến với kiểu message type tương ứng với thông điệp cần trả về.
- Ngược lại, nếu kết quả phản hồi nhằm thông báo lỗi, thì thuộc tính “faultName” sẽ được dùng và thuộc tính “variable” sẽ tham chiếu đến biến kiểu message type tương ứng với thông điệp lỗi.

A.5.5 Cập nhật nội dung của biến

Nội dung các biến được cập nhật thông qua thao tác gán.

A.5.6 Báo lỗi (signaling fault)

Xử lý <throw> có thể được dùng khi tiến trình muốn thông báo một lỗi ra cho bên ngoài. Mỗi lỗi đều cần phải có một QName. Xử lý <throw> sẽ chỉ ra tên của lỗi cần thông báo, và có thể tùy chọn để chỉ ra thêm một biến chứa dữ liệu nhằm cung cấp thông tin chi tiết hơn về lỗi đó. Một trình xử lý lỗi (fault handler) có thể sử dụng các thông tin này để phân tích, xử lý lỗi và nếu cần thì sẽ gửi các thông điệp lỗi đến cho các dịch vụ khác.

BPEL4WS không đòi hỏi tên của lỗi phải được khai báo trước khi chúng được sử dụng trong xử lý <throw>. Lỗi có thể là lỗi do vi phạm các nguyên tắc xử lý của tiến trình hay là lỗi logic trong quá trình thực thi tiến trình. Khi đó, ta có thể thực hiện xử lý <throw> với việc sử dụng một QName thích hợp để làm tên lỗi, và cung cấp thêm thông tin chi tiết (nếu cần) bằng cách sử dụng một biến.

```
<throw faultName="qname" faultVariable="ncname"? standard-attributes>
  standard-elements
</throw>
```

Một ví dụ đơn giản về activity <throw>

```
<throw xmlns:FLT="http://example.com/faults" faultName="FLT:OutOfStock"/>
```

A.5.7 Waiting

Xử lý <wait> cho phép tiến trình tạm dừng xử lý trong một khoảng thời gian bao lâu (duration) hay là cho đến một thời điểm nào đó (deadline)

```
<wait (for="duration-expr" | until="deadline-expr") standard-attributes>
  standard-elements
</wait>
```

Trường hợp sử dụng của xử lý này đó là khi có nhu cầu cần thực thi một xử lý tại một thời điểm nào đó.

```
<sequence>
  <wait until="'2002-12-24T18:00+01:00'"/>
  <invoke partnerLink="CallServer" portType="AutomaticPhoneCall"
    operation="TextToSpeech" inputVariable="seasonalGreeting">
  </invoke>
</sequence>
```

A.5.8 Không làm gì cả

Ta cũng có khi có nhu cầu sử dụng một xử lý để “không làm gì cả”. Ví dụ như khi một lỗi cần được bắt và ta không muốn xử lý, khi này xử lý <empty> sẽ được dùng.

Cú pháp như sau:

```
<empty standard-attributes>
  standard-elements
</empty>
```

A.6 Các xử lý có cấu trúc (structure activity)

Các xử lý có cấu trúc sẽ qui định thứ tự thực hiện của một tập các xử lý chứa trong nó. Chúng mô tả cách một tiến trình nghiệp vụ được tạo ra bằng cách liên kết các xử lý cơ bản mà nó cần thực hiện theo các qui trình xử lý nhằm thể hiện các luồng điều khiển, luồng dữ liệu, xử lý lỗi và các sự kiện, xử lý các quá trình trao đổi thông điệp.

Các xử lý có cấu trúc của BPEL4WS bao gồm:

- Để điều khiển các xử lý thực thi một cách tuần tự ta có: <sequence>, <switch> và <while>.
- Để điều khiển các xử lý thực thi một cách song song và đồng bộ ta có <flow>
- Để điều khiển các xử lý dựa vào các sự kiện từ bên ngoài ta có <pick>.

Các xử lý có cấu trúc có thể được kết hợp và sử dụng lồng nhau một tùy ý.

A.6.1 Sequence

Xử lý <sequence> có thể chứa một hay nhiều xử lý mà sẽ được thực thi một cách tuần tự. Thứ tự thực hiện sẽ giống như thứ tự được khai báo bên trong <sequence>.

Xử lý <sequence> sẽ hoàn thành khi xử lý cuối cùng hoàn thành.

```
<sequence standard-attributes>
  standard-elements
  activity+
</sequence>
```

Một ví dụ cho xử lý <sequence>

```
<sequence>
  <receive name="receiveOrder" partnerLink="ns:shoppingLink"
portType="ns:ShopPortType" operation="ns:placeOrder" variable="order"
createInstance="yes"/>
  <assign>
    <copy>
      <from variable="order" part="product"/>
      <to variable="invoiceData" part="product"/>
    </copy>
  </assign>
  <assign>
    <copy>
      <from variable="order" part="client"/>
```



```

        <to variable="invoiceData" part="client"/>
    </copy>
</assign>
<invoke name="invokeInvoice" partnerLink="ns:invoicingLink"
portType="ns:InvoicePortType" operation="ns:doInvoice"
inputVariable="invoiceData" outputVariable="bill"/>
    <reply name="sendBill" partnerLink="ns:shoppingLink"
portType="ns:BuyerPortType" operation="ns:receiveBill" variable="bill"/>
</sequence>

```

A.6.2 Switch

Xử lý có cấu trúc <switch> hỗ trợ những xử lý theo điều kiện. Những xử lý này ta cũng rất thường hay gặp. <switch> bao gồm một tập có thứ tự của một hay nhiều nhánh điều kiện được biểu diễn bởi các <case>, và sau cùng có thể là một nhánh <otherwise> (có thể có hoặc không).

Các nhánh <case> được xét duyệt theo thứ tự nó được thể hiện trong <switch>, khi có nhánh <case> đầu tiên thỏa điều kiện, thì xử lý kèm theo sẽ được thực hiện và sau đó xử lý <switch> coi như kết thúc (các nhánh còn lại không được quan tâm nữa).

Trường hợp không có nhánh <case> nào thỏa điều kiện, thì xử lý trong nhánh <otherwise> (nếu có khai báo) sẽ được thực hiện. Theo mặc định thì một nhánh <otherwise> với xử lý kèm theo <empty> sẽ luôn tồn tại. Xử lý <switch> kết thúc khi xử lý được thực hiện (ở nhánh được chọn) hoàn thành.

```

<switch standard-attributes>
  standard-elements
  <case condition="bool-expr">+
    activity
  </case>
  <otherwise>?
    activity
  </otherwise>
</switch>

```

Một ví dụ về <switch>

```

<!-- Select the best offer and construct the TravelResponse -->
<switch>

  <case condition="bpws:getVariableData('FlightResponseAA',
    'confirmationData','/confirmationData/Price')
    <= bpws:getVariableData('FlightResponseDA',
    'confirmationData','/confirmationData/Price')">

    <!-- Select American Airlines -->
    <assign>

```

```

        <copy>
        <from variable="FlightResponseAA" />
        <to variable="TravelResponse" />
        </copy>
    </assign>
</case>

<otherwise>
    <!-- Select Delta Airlines -->
    <assign>
        <copy>
        <from variable="FlightResponseDA" />
        <to variable="TravelResponse" />
        </copy>
    </assign>
</otherwise>
</switch>

```

A.6.3 While

Xử lý <while> hỗ trợ lặp lại việc thực thi tuần tự một xử lý. Quá trình lặp này sẽ dừng lại cho tới khi điều kiện lặp không còn thỏa. (biểu thức bool không còn cho giá trị true)

```

<while condition="bool-expr" standard-attributes>
    standard-elements
    activity
</while>

```

Một ví dụ cho <while>

```

<while condition="bpws:getVariableData('itemsShipped') &lt;
bpws:getVariableProperty('shipRequest','props:itemsTotal')">
    <sequence>
        <assign>
            <copy>
            <from opaque="yes"/>
            <to variable="shipNotice" property="props:itemsCount"/>
            </copy>
        </assign>
        <invoke partnerLink="customer"
            portType="sns:shippingServiceCustomerPT"
            operation="shippingNotice" inputVariable="shipNotice">
            <correlations>
                <correlation set="shipOrder" pattern="out"/>
            </correlations>
        </invoke>
        <assign>
            <copy>
            <from expression="bpws:getVariableData('itemsShipped')+
bpws:getVariableProperty('shipNotice','props:itemsCount')"/>
            <to variable="itemsShipped"/>
            </copy>

```

```

        </assign>
    </sequence>
</while>

```

A.6.4 Pick

Xử lý <pick> sẽ lắng nghe để chờ nhận sự xuất hiện của một trong các sự kiện và sau đó sẽ thực thi xử lý gắn với sự kiện đó. Sự xuất hiện của sự kiện này là loại trừ nhau. Nếu nhiều hơn một sự kiện xảy ra thì việc chọn xử lý nào để thực hiện tùy thuộc vào sự kiện nào xảy ra trước.

Cấu trúc của <pick> bao gồm một tập các nhánh có dạng event/activity, và chính xác là chỉ có một nhánh sẽ được chọn căn cứ vào sự xuất hiện của sự kiện gắn với nó, các sự kiện xuất hiện sau đó sẽ không còn được <pick> quan tâm.

Các sự kiện có thể phát sinh dưới hình thức những thông điệp được gửi đến tiến trình hay do một biến cố thời gian. Một dạng đặc biệt của xử lý <pick> được sử dụng khi cần khởi tạo một thể hiện của tiến trình thông qua việc nhận một số các thông điệp. Trong trường hợp này, thuộc tính createInstance của <pick> sẽ được gán giá trị “yes” (giá trị mặc định là “no”). Khi đó, mọi sự kiện trong <pick> phải ở dạng sinh ra do nhận được các thông điệp, nghĩa là không cho phép xuất hiện biến cố thời gian.

Mỗi xử lý <pick> phải có ít nhất một sự kiện <onMessage>. Ngữ nghĩa của <onMessage> giống hệt xử lý <receive>

```

<pick createInstance="yes|no"? standard-attributes>
  standard-elements
  <onMessage partnerLink="ncname" portType="qname" operation="ncname"
variable="ncname"?>+
    <correlations?>
      <correlation set="ncname" initiate="yes|no"?>+
    </correlations>
    activity
  </onMessage>
  <onAlarm (for="duration-expr" | until="deadline-expr")>*
    activity
  </onAlarm>
</pick>

```

Xử lý <pick> hoàn thành khi xử lý được thực hiện (ở nhánh được chọn hoàn thành). Sau đây là một ví dụ về <pick>

```

<pick>
  <onMessage partnerLink="buyer" portType="orderEntry"
    operation="inputLineItem" variable="lineItem">
    <!-- activity to add line item to order -->
  </onMessage>

  <onMessage partnerLink="buyer" portType="orderEntry"
    operation="orderComplete" variable="completionDetail">
    <!-- activity to perform order completion -->
  </onMessage>

  <!-- set an alarm to go after 3 days and 10 hours -->
  <onAlarm for="'P3DT10H'">
    <!-- handle timeout for order completion -->
  </onAlarm>
</pick>

```

A.6.5 Flow

Xử lý flow hỗ trợ thực thi song song và đồng bộ các xử lý.

```

<flow standard-attributes>
  standard-elements
  <links>?
    <link name="ncname">+
  </links>
  activity+
</flow>

```

Các thuộc tính chuẩn và thành phần chuẩn cho các xử lý chứa bên trong <flow> đặc biệt quan trọng bởi vì những thuộc tính và thành phần này được thiết kế chủ yếu là cung cấp các ngữ nghĩa cần thiết để mô tả những quy định, ràng buộc liên quan đến vấn đề song song và đồng bộ.

Xử lý <flow> hoàn thành khi tất cả xử lý bên trong đều hoàn thành. Một xử lý bên trong cũng có thể coi là hoàn thành khi nó bị bỏ qua, không được kích hoạt do điều kiện kích hoạt của nó không thỏa.

Một ví dụ đơn giản trong đó <flow> chứa hai activity <invoke>. Hai xử lý này sẽ được kích hoạt khởi động cùng lúc ngay khi <flow> bắt đầu. Và <flow> kết thúc sau cả hai khi lời gọi trên nhận được phản hồi (giả sử rằng hai lời gọi trên là request/response).

```

<flow>
  <sequence>
    <invoke partnerLink="AmericanAirlines"

```

```

        portType="aln:FlightAvailabilityPT"
        operation="FlightAvailability"
        inputVariable="FlightDetails" />

        <receive partnerLink="AmericanAirlines"
        portType="aln:FlightCallbackPT"
        operation="FlightTicketCallback"
        variable="FlightResponseAA" />

    </sequence>

    <sequence>
        <invoke partnerLink="DeltaAirlines"
        portType="aln:FlightAvailabilityPT"
        operation="FlightAvailability"
        inputVariable="FlightDetails" />

        <receive partnerLink="DeltaAirlines"
        portType="aln:FlightCallbackPT"
        operation="FlightTicketCallback"
        variable="FlightResponseDA" />
    </sequence>
</flow>

```

Tổng quát hơn, một xử lý <flow> ngoài khả năng hỗ trợ thực hiện cùng lúc các xử lý bên trong, nó còn có thể điều khiển đồng bộ hóa giữa những xử lý thông qua các ràng buộc đồng bộ (synchronization dependencies). <link> được dùng để biểu diễn các mối ràng buộc này. Tất cả các <link> của một <flow> đều phải được khai báo bên trong <flow>.

Một <link> được xác định bởi một tên (name) và thể hiện ràng buộc giữa hai xử lý. Trong đó

- Một xử lý sẽ giữ vai trò là source của link (xử lý đó sẽ chứa <source> với thuộc tính “linkName” là tên của <link>, và còn có thể có thêm một thuộc tính “transitionCondition” với giá trị là một biểu thức bool – mặc định là “true”)
- Xử lý còn lại sẽ giữ vai trò là target của link (xử lý đó sẽ chứa <target> với thuộc tính linkName là tên của <link>). Target activity sẽ không được thực hiện chừng nào tình trạng của link đó chưa được kích hoạt. Tình trạng của một link gọi là được kích hoạt khi source activity của link đó kết thúc và transitionCondition của <source> có giá trị “true”. Ngoài ra, target activity có thể có thêm thuộc tính “joinCondition” (sẽ được trình bày trong phần sau)

Một `<link>` được khai báo trong `<flow>` phải có đúng một source activity và một target activity. Và giữa hai xử lý được ràng buộc bởi nhiều nhất là một link, nghĩa là, với hai `<link>` khác nhau thì không được có cùng source activity và cùng target activity.

```
<!-- Make a concurrent invocation to AA in DA -->
<flow>
  <links>
    <link name="XtoY"/>
    <link name="CtoD"/>
  </links>
  <sequence name="X">
    <source linkName="XtoY"/>
    <invoke name="A" .../>
    <invoke name="B" .../>
  </sequence>
  <sequence name="Y">
    <target linkName="XtoY"/>
    <receive name="C" ...>
      <source linkName="CtoD"/>
    </receive>
    <invoke name="E" .../>
  </sequence>
  <invoke partnerLink="D" ...>
    <target linkName="CtoD"/>
  </invoke>
</flow>
```

Một `<link>` được gọi là “*cross boundary*” khi các source activity hay target activity của link được chứa trong một construct trong khi link được khai báo bên ngoài construct đó (một construct được tạo ra bởi một xử lý có cấu trúc). Ví dụ ở trên có thể minh họa cho trường hợp “cross boundary” (source activity của link ‘CtoD’ ở trong construct tạo ra bởi `<sequence>` trong link đó được khai báo trong construct của `<flow>`). Tuy nhiên tình trạng “cross boundary” này không được xảy ra với xử lý `<while>`, event handler hay compensation handler. Riêng đối với fault handler thì link được phép “cross boundary”, trong đó fault-handler phải chứa source activity và target activity sẽ thuộc scope bao ngoài scope của fault-handler.

Cuối cùng, một điểm nữa cần quan tâm đó là không được dùng link để tạo chu trình giữa hai xử lý, nghĩa là: B chỉ thực thi khi A hoàn thành và A chỉ thực thi khi B hoàn thành.

Ngữ nghĩa của link (link semantic)

Trong các phần còn lại chúng ta sẽ qui ước như sau:

- ▶ Những link mà trong đó A là source activity sẽ được gọi là link đầu ra của A.
- ▶ Những link mà trong đó A là target activity sẽ được gọi là link đầu vào của A.
- ▶ Nếu xử lý X là target và Y là source của link, thì ta nói rằng ‘X phụ thuộc đồng bộ vào Y’.

Mỗi target activity đều có một “join condition” kèm theo (được hiểu ngầm hoặc khai báo tường minh). Các join condition được khai báo tường minh bằng các element <joinCondition> ở bên trong <target> element. Và giá trị của <joinCondition> là một biểu thức bool. Ý nghĩa của joinCondition là được dùng để kiểm tra một xử lý có đủ điều kiện kích hoạt hay không.

Trạng thái “sẵn sàng” của một xử lý sẽ được quyết định bởi các xử lý có cấu trúc chứa nó. Ví dụ, xử lý thứ hai trong một <sequence> sẽ ở trạng thái “sẵn sàng” ngay khi xử lý thứ nhất hoàn thành ; một xử lý bên trong <flow> ở trạng thái “sẵn sàng” ngay khi <flow> được kích hoạt. Một xử lý đã ở trạng thái “sẵn sàng” và có một hay nhiều incoming link, thì nó sẽ được kích hoạt khi trạng thái của tất cả các link đầu vào của nó đều đã được kích hoạt và join condition đi kèm theo được có giá trị “true”.

Trạng thái của một link có thể có một trong ba giá trị sau : được kích hoạt, không được kích hoạt và chưa xác định. Mỗi khi <flow> bắt đầu thực thi, thì tất cả các link đều có trạng thái là “chưa xác định”. Và ta thấy rằng chu kỳ sống của một link bằng đúng chu kỳ sống của xử lý <flow> mà nó được khai báo trong đó.

Vai trò của link trong mối quan hệ đồng bộ giữa hai xử lý sẽ được giải thích một cách rõ hơn . Khi xử lý A hoàn thành, các bước sau sẽ được tiến hành :

- Đánh giá trạng thái cho các link đầu ra của A. Trạng thái kết quả sẽ là “được kích hoạt” hay “không được kích hoạt”. Để xác định tình trạng của mỗi link

thì `transitionCondition` của link đó sẽ được đánh giá. Nếu `transitionCondition=true` thì trạng thái của link là “được kích hoạt”. Ngược lại thì là “không được kích hoạt”. Một điều lưu ý là việc đánh giá `transitionCondition` được thực hiện sau khi xử lý đã hoàn thành. Vì thế, nếu có bất kỳ lỗi nào xảy ra trong quá trình đánh giá thì cũng sẽ không ảnh hưởng đến kết quả trước đó của xử lý, và lỗi này sẽ được xử lý scope chứa xử lý đó. Nếu target activity ở bên ngoài scope của source activity thì trạng thái của link sẽ là “không được kích hoạt”. Một điều quan trọng cần nhớ là, trường hợp bản thân source activity là một `<scope>`, thì “hoàn thành” không nhất thiết là không được có lỗi trong quá trình xử lý. Trong quá trình xử lý của `<scope>` có thể có lỗi phát sinh. Nhưng `<scope>` vẫn được gọi là “hoàn thành” khi lỗi đó được bắt và xử lý thành công bởi một fault handler của scope. Trường hợp `<link> L` với source activity là `<scope> S`, lỗi phát sinh nếu có trong quá trình đánh giá `transitionCondition` sẽ được truyền ra scope bên ngoài S.

- Với mỗi xử lý B mà “phụ thuộc đồng bộ” vào A, sẽ kiểm tra:
 - ▶ B đã ở trạng thái “sẵn sàng” chưa
 - ▶ Trạng thái của tất cả link đầu vào của B đã “được kích hoạt” chưa
- Nếu cả hai điều kiện trên đều thỏa, thì sẽ thực hiện đánh giá `joinCondition` của B. Nếu `joinCondition` có giá trị false thì lỗi “bpws :joinFailure ” sẽ được phát sinh, ngược lại, B sẽ được kích hoạt.

Nếu trong quá trình thực thi của một xử lý có cấu trúc S, nếu một xử lý X bên trong S một không được thực thi (ví dụ như link đầu vào của X “không được kích hoạt”) thì trạng thái của toàn bộ các link đầu ra của X được gán là “không được kích hoạt”.

Trường hợp xử lý `<flow>` chứa bên trong nó một xử lý `<flow>` khác, và `<flow>` bên trong khai báo một số `<link>` trùng tên với các `<link>` đã được định nghĩa trước đó ở `<flow>` bên ngoài. Khi khai báo các `<source>` và `<element>` trong các xử lý mà có tham chiếu đến “linkName” thì tham chiếu sẽ được chuyển đến `<link>` cùng tên nào mà gần xử lý đó nhất.

- Một ví dụ cho trường hợp này

```
<flow name="F1">
  <links>
    <link name="L1"/> <!-- L1 is defined here and ... -->
  </links>
  <sequence name="S1">
    <flow name="F2">
      <links>
        <link name="L1"/> <!-- ... here -->
      </links>
      <sequence name="S2">
        <receive name="R">
          <source linkName="L1"/> <!--This matches
F2.L1 and not F1.L1 -->
        </receive>
        <invoke name="I" .../>
      </sequence>
      ...
    </flow>
    ...
  </sequence>
  ...
</flow>
```

Xử lý Dead-Path_elimination (DPE)

Trong một số trường hợp mà luồng điều khiển rất phức tạp với việc sử dụng rất nhiều `<link>`. Nếu joinCondition của xử lý A có giá trị false thì A sẽ không được thực thi, và một lỗi 'bpws:joinFailure' sẽ được phát sinh. Ngoài ra, ta cần phải thực hiện gán giá trị cho các link đầu ra của A là "không được kích hoạt".

BPEL4WS hỗ trợ vấn đề này thông qua việc sử dụng thuộc tính 'suppressJoinFailure' trong các xử lý. Nếu giá trị của thuộc tính này là 'yes', thì lỗi này sẽ được bắt và xử lý bởi một trình xử lý lỗi đặc biệt (fault handler này sẽ có phần xử lý là một `<empty>` xử lý). Tuy nhiên, bởi vì xử lý (có joinCondition = false) chưa được thực thi, nên các link đầu ra của nó phải được tự động gán trạng thái "không được kích hoạt". Điều này sẽ giúp cho các xử lý liên quan được bỏ qua, tránh tình trạng 'ngõ cụt'. Đây là tình trạng mà các xử lý phải chờ để được kích hoạt trong khi các link đầu vào của chúng ở tình trạng 'không xác định.'

Link và các xử lý có cấu trúc

Link có thể được dùng để liên kết hai xử lý được chứa trong những xử lý có cấu trúc khác nhau. Khi đó, ta cần thận trọng để đảm bảo luồng xử lý của tiến trình được thực hiện đúng theo ý muốn. Phần này sẽ làm rõ các vấn đề cần quan tâm thông qua phân tích một ví dụ.

Mô tả ví dụ:

- ▶ Trong một <flow> chứa 3 xử lý: một <sequence> S và hai <receive> X và Y.
- ▶ Bên trong S: sau khi A hoàn thành, B không được cho đến khi trạng thái của các link đầu vào của nó là X và Y được kích hoạt và joinCondition được đánh giá có giá trị true.
- ▶ Khi X và Y hoàn thành, joinCondition của B được đánh giá.
- ▶ Giả sử rằng biểu thức $P(X,B) \text{ OR } P(Y,B)$ có giá trị false, khi đó lỗi “bpws:joinFailure” được phát sinh. Và bởi vì, giá trị của thuộc tính suppressionJoinFailure là “no” nên xử lý của <flow> bị dừng và cả B và C đều không được thực hiện.
- ▶ Trong trường hợp ngược lại, nếu giá trị của suppressionJoinFailure được gán là “yes”, thì B sẽ bị bỏ qua nhưng C sẽ được thực thi bởi vì lỗi “bpws:joinFailure” sẽ được xử lý trong scope của B.

```
<flow suppressJoinFailure="no">
  <links>
    <link name="XtoB"/>
    <link name="YtoB"/>
  </links>
  <sequence name="S">
    <receive name="A" ...>
      ...
    </receive>
    <receive name="B" ...>
      <targets>
        <target linkName="XtoB"/>
        <target linkName="YtoB"/>
      </targets>
      ...
    </receive>
    <receive name="C" ...>
      ...
    ...
  </sequence>
</flow>
```

```

    </receive>
</sequence>
<receive name="X" ...>
  <sources>
    <source linkName="XtoB">
      <transitionCondition>P(X,B)</transitionCondition>
    </source>
  </sources> ...
</receive>
<receive name="Y" ...>
  <sources>
    <source linkName="YtoB">
      <transitionCondition>P(Y,B)</transitionCondition>
    </source>
  </sources>
  ...
</receive>
</flow>

```

- Sau đó, giả sử ta bổ sung bằng cách liên kết A, B và C với những link (có `transitionCondition="true"`) thay vì để chúng trong một `<sequence>`. Khi này, cả B và C sẽ luôn luôn được thực thi. Bởi vì `transitionCondition` của link “AtoB” luôn bằng “true”, nên `joinCondition` cũng sẽ luôn có giá trị “true”, bất chấp giá trị của biểu thức `P(X,B)` và `P(Y,B)`.

```

<flow suppressJoinFailure="no">
  <links>
    <link name="AtoB"/>
    <link name="BtoC"/>
    <link name="XtoB"/>
    <link name="YtoB"/>
  </links>
  <receive name="A">
    <sources>
      <source linkName="AtoB"/>
    </sources>
  </receive>
  <receive name="B">
    <targets>
      <target linkName="AtoB"/>
      <target linkName="XtoB"/>
      <target linkName="YtoB"/>
    </targets>
    <sources>
      <source linkName="BtoC"/>
    </sources>
  </receive>
  <receive name="C">
    <targets>
      <target linkName="BtoC"/>
    </targets>
  </receive>
</flow>

```

```

</receive>
<receive name="X">
  <sources>
    <source linkName="XtoB">
      <transitionCondition>P(X,B)</transitionCondition>
    </source>
  </sources>
</receive>
<receive name="Y">
  <sources>
    <source linkName="YtoB">
      <transitionCondition>P(Y,B)</transitionCondition>
    </source>
  </sources>
</receive>
</flow>

```

A.7 Scope

Ngữ cảnh xử lý (behavior context) của mỗi xử lý được cung cấp bởi một scope. Một scope có thể cung cấp một fault handler, event handler, compensation handler, data variable, partner link và correlation sets.

Cú pháp khai báo của một scope như sau:

```

<scope variableAccessSerializable="yes|no" standard-attributes>
  standard-elements
  <variables>?
    ... see above under <process> for syntax ...
  </variables>
  <correlationSets>?
    ... see above under <process> for syntax ...
  </correlationSets>
  <faultHandlers>?
    ... see above under <process> for syntax ...
  </faultHandlers>
  <compensationHandler>?
    ... see above under <process> for syntax ...
  </compensationHandler>
  <eventHandlers>?
    ...
  </eventHandlers>
  activity
</scope>

```

Mỗi scope có một xử lý chính dùng để phân xử lý của nó. Xử lý này có thể là một xử lý có cấu trúc phức tạp, với nhiều xử lý khác chứa bên trong nó. Scope được chia sẻ bởi tất cả các xử lý bên trong.

Một ví dụ về scope với xử lý chính là <flow>. Xử lý <flow> sẽ chứa 2 xử lý <invoke> cần xử lý cùng lúc. Một trong hai xử lý <invoke> này có thể nhận một hay nhiều kiểu lỗi phản hồi. Phần fault handler của scope được chia sẻ bởi cả hai <invoke> và có thể được dùng để bắt lỗi nhận về.

```
<scope>
  <faultHandlers>?
  ...
</faultHandlers>

  <flow>
    <invoke partnerLink="Seller" portType="Sell:Purchasing"
      operation="SyncPurchase"
      inputVariable="sendPO"
      outputVariable="getResponse" />
    <invoke partnerLink="Shipper"
      portType="Ship:TransportOrders"
      operation="OrderShipment"
      inputVariable="sendShipOrder"
      outputVariable="shipAck" />

  </flow>
</scope>
```

A.7.1 Xử lý dữ liệu data và Partner Link

Một scope có thể khai báo các biến và partner link mà chỉ có ý nghĩa sử dụng bên trong scope.

A.7.2 Xử lý lỗi trong tiến trình nghiệp vụ

Tiến trình nghiệp vụ thường thực hiện các tương tác trong thời gian dài và theo cơ chế bất đồng bộ. Các tiến trình còn thực hiện thao tác trên những dữ liệu nhạy cảm của các cơ sở dữ liệu bên trong hệ thống. Xử lý lỗi trong môi trường này là một yêu cầu khó khăn nhưng cần thiết.

Xử lý lỗi trong tiến trình nghiệp vụ chủ yếu dựa trên khái niệm “compensation”, nghĩa là xây dựng các xử lý (application-specific) nhằm quay lui lại kết quả của công việc đã được thực hiện trước đó (trong bối cảnh là ta đang thực hiện một công việc lớn – một giao tác- gồm nhiều công việc nhỏ, vì một lý do gì đó mà ta không muốn tiếp tục thực hiện giao tác đó, nên ta sẽ xử lý quay lui lại những phần việc đã được làm).

BPEL4WS hỗ trợ thực thi khái niệm “compensation” thông qua cung cấp khả năng xây dựng các thành phần xử lý lỗi (fault handling) và xử lý compensation (compensation handling).

A.7.3 Compensation handler

Định nghĩa một compensation handler

Một compensation handler đơn giản là một lớp bọc cho phần xử lý quay lui (compensation) ở bên trong

```
<compensationHandler>?
  activity
</compensationHandler>
```

Như đã giải thích ở trên, ta có thể thực hiện gọi phương thức compensation từ trong một xử lý <invoke>, thay vì phải khai báo một <scope>.

- Gọi compensation trực tiếp từ xử lý <invoke>

```
<invoke partnerLink="Seller"
  portType="SP:Purchasing"
  operation="SyncPurchase"
  inputVariable="sendPO"
  outputVariable="getResponse">
  <correlations>
    <correlation set="PurchaseOrder" initiate="yes"
pattern="out"/>
  </correlations>

  <compensationHandler>
    <invoke partnerLink="Seller"
portType="SP:Purchasing"
      operation="CancelPurchase"
      inputVariable="getResponse"
      outputVariable="getConfirmation">
      <correlations>
        <correlation set="PurchaseOrder" pattern="out"/>
      </correlations>
    </invoke>
  </compensationHandler>
</invoke>
```

- Trong ví dụ này, xử lý <invoke> thực hiện xử lý mua hàng, và trong trường hợp cần quay lui (compensate) xử lý mua hàng vừa thực hiện, thì <compensationHandler> sẽ gọi phương thức “cancellation” với

cùng portType, partnerLink, với kết quả đầu ra của lần mua hàng sẽ làm tham số đầu vào cho phương thức hủy này.

- o Gọi compensation từ <scope> (cú pháp chuẩn)

```
<scope>
  <compensationHandler>
    <invoke partnerLink="Seller"
                                     portType="SP:Purchasing"
                                     operation="CancelPurchase"
                                     inputVariable="getResponse"

      outputVariable="getConfirmation">
        <correlations>
          <correlation set="PurchaseOrder" pattern="out" />
        </correlations>
      </invoke>
    </compensationHandler>

    <invoke partnerLink="Seller"
                                     portType="SP:Purchasing"
                                     operation="SyncPurchase"
                                     inputVariable="sendPO"
                                     outputVariable="getResponse">
      <correlations>
        <correlation set="PurchaseOrder" initiate="yes"
pattern="out" />
      </correlations>
    </invoke>
  </scope>
```

Sử dụng một compensation handler

Compensation handler có thể được gọi bằng cách dùng xử lý <compensate>, chỉ ra tên của scope mà chứa compensation handler sẽ được gọi. Một compensation handler của scope chỉ được phép gọi khi scope đã kết thúc một cách bình thường.

Lưu ý rằng trong trường hợp xử lý <invoke> có định nghĩa bên trong nó một compensation handler, thì tên của xử lý này là tên của scope được dùng trong xử lý <compensate>

```
<compensate scope="ncname"? standard-attributes>
  standard-elements
</compensate>
```

Xử lý <compensate> chỉ có thể được gọi trong ngữ cảnh sau:

- Trong một fault handler của scope mà bao ngoài trực tiếp scope của compensation cần được thực thi.
- Trong một compensation handler của scope mà bao ngoài trực tiếp scope của compensation cần được thực thi.

```
<compensate scope="RecordPayment" />
```

A.7.4 Xử lý lỗi

Xử lý lỗi trong BPEL4WS nhằm quay lui lại những xử lý xử lý không thành công (đã phát sinh lỗi) trong một scope. Các lỗi sẽ được bắt và xử lý trong các fault handler. Và ngay cả khi quá trình xử lý lỗi không bị vấn đề gì thì scope chứa xử lý phát sinh lỗi cũng không được gọi là kết thúc thành công. Và các compensation handler của những scope này không bao giờ được thực hiện.

Việc thiết kế các fault handler trong scope sẽ cho phép ta lựa chọn các xử lý lỗi khác nhau thông qua các xử lý <catch>. Mỗi xử lý <catch> sẽ được chỉ định để bắt một lỗi khác nhau, với QName của lỗi và một biến dùng để chứa các thông tin chi tiết. Nếu tên lỗi không được chỉ định, thì <catch> đó sẽ bắt tất cả các lỗi.

Biến để chứa lỗi được chỉ định bằng cách sử dụng thuộc tính “faultVariable” trong <catch>. Biến này là cục bộ của fault handler mà nó được khai báo, và không được sử dụng ở bên ngoài. Ngoài ra, biến này là tùy chọn vì có những lỗi mà không có thông tin kèm theo.

Phản hồi lỗi của xử lý <invoke> là một dạng của lỗi, với tên rõ ràng và phần dữ liệu lỗi sẽ tùy thuộc vào định nghĩa về fault trong WSDL operation. Một dạng lỗi thứ hai là những lỗi được phát sinh do sử dụng xử lý <throw> kèm với tên và thông tin về lỗi. <catchAll> sẽ được sử dụng để bắt những lỗi mà không được quan tâm bởi các <catch> ở trên.

```
<faultHandlers>?
<!-- there must be at least one fault handler or default -->
  <catch faultName="qname"? faultVariable="ncname"?*>
    activity
  </catch>

  <catchAll>?
    activity
```



```

</catchAll>

</faultHandlers>

```

Bởi vì cơ chế linh hoạt trong việc cho phép chỉ định loại fault mà một `<catch>` có thể xử lý, nên sẽ có trường hợp một lỗi chỉ định xử lý bởi nhiều fault handler. Một số nguyên tắc được sử dụng để chọn ra `<catch>` thích hợp xử lý lỗi:

- Nếu lỗi không có thông tin dữ liệu đi kèm, thì xử lý `<catch>` mà có `faultName` trùng với tên của lỗi sẽ được chọn. Nếu không, lỗi sẽ được xử lý bởi `<catchAll>`
- Nếu lỗi có dữ liệu lỗi kèm theo, thì xử lý `<catch>` nào có tên lỗi và kiểu message type của biến lỗi (`faultVariable`) trùng với những thông tin tương ứng của biến lỗi sẽ được chọn để xử lý. Nếu không có `<catch>` nào thỏa, thì lỗi sẽ được xử lý bởi `<catchAll>`.

Nếu không có `<catch>` hay `<catchAll>` được chỉ chọn, thì lỗi sẽ không được bắt bởi scope hiện tại mà sẽ được chuyển ra scope bên ngoài trực tiếp của scope hiện tại. Nếu lỗi được chuyển qua cho tiến trình xử lý, và không tìm thấy fault handler phù hợp thì tiến trình sẽ kết thúc một cách “không bình thường”, giống như khi một xử lý `<terminate>` được gọi.

```

<faulthandlers>
  <catch faultName="x:foo">
    <empty/>
  </catch>

  <catch faultVariable="bar">
    <empty/>
  </catch>

  <catch faultName="x:foo" faultVariable="bar">
    <empty/>
  </catch>

  <catchAll>
    <empty/>
  </catchAll>
</faulthandlers>

```

Scope mà trong đó có xuất hiện lỗi được xem như có kết thúc không bình thường, cho dù lỗi đó có được bắt và xử lý hay không. Và compensation handler của một scope như thế sẽ không bao giờ được gọi.

A.7.5 Xử lý sự kiện

Toàn bộ tiến trình cùng với mỗi scope đều có thể được kết hợp với các event handler. Các event handle sẽ được gọi cùng lúc nếu sự kiện tương ứng xảy ra. Xử lý bên trong một event handler có thể là bất kỳ dạng xử lý nào ngoại trừ xử lý <compensate/> vì xử lý này chỉ được phép gọi trong fault handler và compensation handler.

Có hai loại sự kiện. Loại sự kiện thứ nhất có thể là các thông điệp đến tương ứng với các operation trong WSDL. Loại thứ hai có thể là biến cố về thời gian.

Cú pháp khai báo của event handler như sau:

```
<eventHandlers>?
  <!-- there must be at least one onMessage or
  onAlarm handler -->
  <onMessage partnerLink="ncname"
    portType="qname"
    operation="ncname"
    variable="ncname"?>*
    <correlations>?
      <correlation set="ncname" initiate="yes|no">+
    </correlations>
    activity
  </onMessage>

  <onAlarm for="duration-expr"? until="deadline-expr"?>*
    activity
  </onAlarm>
</eventHandlers>
```

Điều cần lưu ý là, không giống như event handler và compensation handler, event handler được xem như một phần trong xử lý của một scope.

Các sự kiện dạng thông điệp

<onMessage> nhằm lắng nghe loại sự kiện thông điệp, nghĩa là chờ một thông điệp thích hợp được gửi đến. Ý nghĩa và các thuộc tính của <onMessage> và các thuộc tính của nó rất giống với xử lý <receive> (partnerLink, portType, operation, variable, correlation sets). Điểm khác biệt đó là không cho phép tạo các thể hiện tiến

trình. Vì thế các event handler không có thuộc tính “createInstance”. Khi thông điệp chỉ định đến, xử lý tương ứng sẽ được thực thi.

Một ví dụ về cách dùng event handler để hỗ trợ việc kết thúc một tiến trình thông qua một thông điệp từ bên ngoài gửi vào.

```
<process name="orderCar">
  ...
  <eventHandlers>
    <onMessage partnerLink="buyer"
                                     portType="car"
                                     operation="cancel"
                                     variable="cancelDetails">
                                     <terminate/>
    </onMessage>
    ...
  </eventHandlers>
  ...
</process>
```

Các sự kiện dạng biến cố thời gian

<onAlarm> dùng để biểu diễn một sự kiện dạng timeout.

- Thuộc tính “for” chỉ ra khoảng thời gian mà sau đó sự kiện sẽ được chuyển đi.
- Thuộc tính “until” chỉ ra thời điểm mà đến lúc đó sự kiện sẽ được chuyển đi.

Thời gian bắt đầu tính khi scope của nó bắt đầu. onLarm chỉ có một trong hai thuộc tính được chỉ định.

Sự kích hoạt event handler

Event handler gắn với scope thì sẽ được kích hoạt ngay khi scope bắt đầu thực thi. Nếu event handler được gắn với tiến trình, thì sẽ được kích hoạt ngay khi thể hiện của tiến trình được tạo ra. Thể hiện của một tiến trình được tạo ra khi xử lý <receive> đầu tiên (với createInstance= “yes”) nhận được và xử lý thông điệp tương ứng.

Một ví dụ sử dụng event handler để quản lý thời gian sống của thể hiện tiến trình. Đó là ta sẽ nhúng dữ liệu thời gian vào thông điệp dùng để tạo thể hiện tiến trình.

```

<wsdl:definitions
  targetNamespace="http://www.example.com/wsdl/example"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  ...>

  <wsdl:message name="orderDetails">
    <part name="processDuration"
          type="xsd:duration"/>
  </wsdl:message>

</wsdl:definitions>

```

```

<process name="orderCar"
  xmlns:def="http://www.example.com/wsdl/example"
  ...>
  ...
  <eventHandlers>
    <onAlarm for=
      "bpws:getVariableData(orderDetails,processDuration)"
    >
    ...
    </onAlarm>
  </eventHandlers>
  ...
  <variable name="orderDetails"
messageType="def:orderDetails"/>
  </variable>
  ...
  <receive name="getOrder"
    partnerLink="buyer"
    portType="car"
    operation="order"
    variable="orderDetails"
    createInstance="yes"/>
  ...
</process>

```

Xử lý sự kiện tiến trình

- Sự kiện dạng biến cố thời gian

Việc tính thời gian cho một sự kiện kiểu biến cố thời gian bắt đầu ngay khi event handler được kích hoạt. Sự kiện sẽ được phát khi đến thời điểm hay sau khi đã qua một khoảng thời gian được chỉ định trong các thuộc tính “for” và “until”. Sự kiện loại này chỉ được thực hiện nhiều nhất là một lần và chuyển sang trạng thái “không còn hiệu lực” sau khi thực hiện phần xử lý kèm theo.

- Sự kiện dạng thông điệp

Sự kiện này xảy ra khi một thông điệp thích hợp được gửi đến. Khi đó, xử lý tương ứng sẽ được thực hiện. Sự kiện loại này có thể được lắng nghe và xử lý nhiều lần trong khi scope tương ứng vẫn còn hoạt động.

Vô hiệu hoá các sự kiện

Tất cả các event handler gắn với một scope sẽ bị vô hiệu hóa khi quá trình xử lý của scope đó kết thúc một cách bình thường. Một scope sẽ phải đợi cho đến khi tất cả các event handler xử lý xong để thật sự hoàn thành.

Phụ lục B

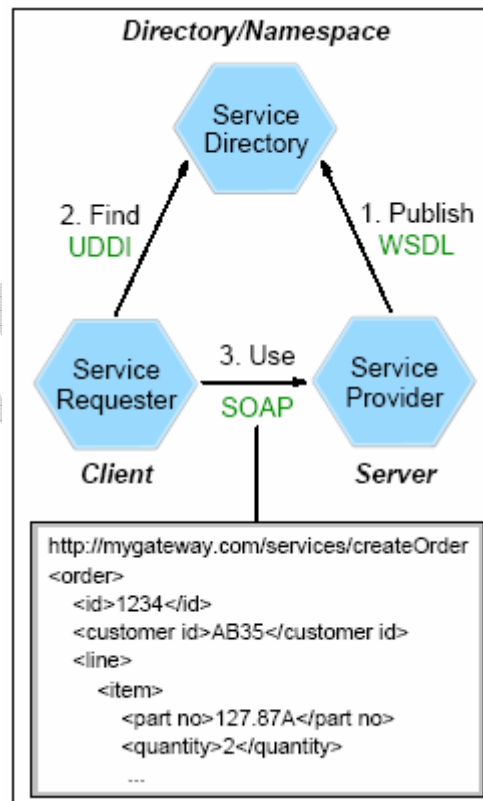
SOA VÀ WEB SERVICES

B.1 Kiến trúc Web services

Web services là một tập các chuẩn đặc tả mở rộng khả năng của các chuẩn có sẵn như XML, URL và HTTP nhằm cung cấp chuẩn truyền thông giữa các hệ thống với nhau. Web services là những thành phần thực thi một số xử lý nghiệp vụ thông qua những dịch vụ và cung cấp những dịch vụ qua mạng, những dịch vụ này có thể được triệu gọi bởi các dịch vụ client bằng cách sử dụng giao thức SOAP trên HTTP. Web services độc lập về ngôn ngữ và độc lập về nền tảng bởi vì nó tách biệt đặc tả ra khỏi cài đặt; nó còn hỗ trợ tích hợp loose coupling giữa các ứng dụng với nhau qua trao đổi các thông điệp đồng bộ và bất đồng bộ thông. Web services dựa trên kiến trúc phân tán trong đó không có bất kỳ dịch vụ xử lý trung tâm nào và tất cả dạng truyền thông đều sử dụng các giao thức chuẩn. Các giao thức không được có bất kỳ ý nghĩa ngầm định nào bên trong mà phải được mô tả rõ ràng.

Một trong những đặc tính quan trọng của mô hình tính toán dựa trên Web services là ở đó cả các client và Web services đều không cần biết cài đặt của nhau. Kiến trúc Web services cung cấp nhiều thành phần cho phép các ứng dụng client tìm kiếm và sử dụng những Web services mình cần một cách động. Web services hứa hẹn mang đến khả năng tạo ra các môi trường phân tán trong đó bất kỳ ứng dụng nào, hoặc bất kỳ component ứng dụng nào cũng đều có thể kết hợp với nhau dễ dàng với tính độc lập nền tảng, và độc lập ngôn ngữ (language-neutral). Web service có thể được sử dụng vào các mục đích đơn giản như đăng nhập vào một trang web hay với mục đích phức tạp như xử lý nghiệp vụ giao dịch vay mượn giữa các công ty với nhau. Điểm khác biệt chính của Web services với các công nghệ phân tán trước đây như Win32, J2EE và CGI là ở sự chuẩn hoá. Web services sử dụng XML, một ngôn ngữ độc lập trong việc biểu diễn dữ liệu, làm ngôn ngữ trao đổi thông tin. Bởi vậy khi được kết hợp với nhau, khả năng tích hợp phần mềm và đa kết nối giữa những mô hình web

services thật đáng kinh ngạc. Thêm vào đó, các chuẩn Web services mới còn hỗ trợ các tính năng cao cấp như hỗ trợ giao dịch, bảo mật, quy trình nghiệp vụ, vân vân...



Mô hình web services dạng đơn giản định nghĩa cách thức tương tác giữa Service Requester, Service Provider, và Service Directory như sau :

Bên sử dụng dịch vụ tìm kiếm các dịch vụ trong một UDDI Service Directory. Chúng sẽ lấy thông tin mô tả WSDL của các Web services cung cấp bởi Service Providers từ trước thông qua Service Directory. Sau khi lấy được mô tả WSDL, bên yêu cầu dịch vụ kết nối đến nhà cung cấp dịch vụ bằng cách triệu gọi các dịch vụ thông qua giao thức SOAP.

Web services cơ bản bao gồm các khái niệm SOAP, WSDL, và UDDI. Chúng ta sẽ lần lượt phân tích trong các phần sau. Cần lưu ý là các chuẩn trên có thể được kết hợp với nhau hoặc dùng độc lập tùy trường hợp cụ thể.

B.2 Các đặt trưng của Web services

B.2.1 Loosely coupled

Loosely coupled nghĩa là các Web Service và chương trình triệu gọi chúng có thể thay đổi độc lập với nhau thay vì phải thiết kế lại những thành phần liên quan ở hai bên mỗi khi có sự thay đổi. Cũng có thể thay đổi Web service interface mà không làm ảnh hưởng đến khả năng tương tác của client với dịch vụ đó. Trong khi đó, với một hệ thống có tính tightly coupled, nghiệp vụ ở phía client và ở server phải ràng buộc chặt chẽ với nhau, mỗi khi có một interface thay đổi thì phía còn lại cũng phải được cập nhật. Xu hướng hiện nay là phát triển các kiến trúc có tính loosely coupled để các hệ thống phần mềm trở nên dễ quản lý hơn, làm cho khâu tích hợp các hệ thống khác nhau cũng trở nên dễ dàng hơn.

B.2.2 Tính đóng gói

Tính đóng gói (Encapsulated) nghĩa là phạm vi bên ngoài của mỗi Web service không được biết đến cài đặt của Web Service đó. Các chức năng chỉ được cung cấp thông qua các interface mà Web Service đó cung cấp. Về bản chất, Web Service trừu tượng hoá cài đặt của nó qua interface, tương tự như cách XML tách biệt thông tin ra khỏi xử lý. Web Service kế thừa tính đóng gói từ những kiến trúc hướng đối tượng dựa trên Java, C++ và COM.

B.2.3 Contracted

Contracted nghĩa là các hành vi của Web Service bao gồm cách thức kết nối, ràng buộc, các tham số đầu vào đầu ra đều được cung cấp đến người sử dụng Web Service đó.

B.2.4 Giao thức chuẩn

Web Service là một tập các chuẩn mở và phi thương mại. Web Service còn dựa trên nền tảng XML để định dạng dữ liệu và HTTP làm giao thức truyền dữ liệu, ngoài ra còn có SOAP, WSDL và UDDI là các chuẩn nền tảng của Web Service cũng dựa trên XML. Các chuẩn này có thể là đã cũ nhưng nếu không dựa trên đó, Web Service sẽ được triển khai như những hệ thống đóng, độc quyền và đặc trưng cho mỗi nhà cung

cấp trao đổi với nhau trên một nền tảng cố định. Tính mở của đặc tả Web Service cho phép nhiều nhà cung cấp và tổ chức trình thể hiện những cách nhìn đa dạng góp phần phát triển các công nghệ Web Service..

B.2.5 Tự định nghĩa

Web Service tự định nghĩa chính nó. Tính tự định nghĩa (self-defining) là một đặc tính của XML được tận dụng trong những công nghệ nền tảng của Web Service như SOAP và WSDL.

B.2.6 Tìm kiếm và triệu gọi động

Bằng cách tận dụng công nghệ UDDI , một Web Service consumer có thể định vị và triệu gọi một Web Service trong khi chạy mà không cần biết trước cụ thể Web Service nào cài đặt, cung cấp chức năng mình mong muốn.

B.3 Lợi ích của Web services

Lợi ích của Web Service có thể tóm gọn trong một đoạn như sau : “Web Service cung cấp một cơ chế đơn giản để kết nối những ứng dụng lại với nhau bất kể công nghệ hoặc thiết bị chúng đang sử dụng hoặc vị trí. Web Service dựa trên các chuẩn giao thức được hỗ trợ rộng rãi trên internet nhằm giảm chi phí liên lạc. Cách tiếp cận có tính loose coupling hỗ trợ đa kết nối và chia sẻ thông tin giữa các dịch vụ, mà các được đó tự định nghĩa và có thể được tìm thấy một cách tự động” . Ta sẽ lần lượt tìm hiểu chi tiết lợi ích thương mại và lợi ích kỹ thuật của Web Service.

- **Web Service có thể được truy cập ở mọi nơi, mọi lúc, hầu như trên bất kì công nghệ và thiết bị nào hỗ trợ Web Service.**

Lợi ích thương mại– Tăng hiệu suất quy trình nghiệp vụ	Lợi ích kỹ thuật – Giảm chi phí
Tăng hiệu suất của quy trình nghiệp vụ bằng cách giảm chi phí và thời gian kết nối các ứng dụng với nhau.	Giảm chi phí kết nối
Tăng khả năng truy xuất dữ liệu theo gian thực, kết nối từ xa đến các nguồn dữ liệu	Giảm mức độ phức tạp của quá trình tích hợp
Hỗ trợ thương mại thời gian thực	Độc lập nền tảng và độc lập công nghệ
Hỗ trợ khác hàng, đối tác và nhân viên	

➤ **Dựa trên các chuẩn giao thức được hỗ trợ rộng rãi trên internet**

Lợi ích thương mại và kỹ thuật – Giảm chi phí và lựa chọn
<p>Lợi ích từ các « chuẩn mở » Không phụ thuộc vào một công nghệ độc quyền nào Nhiều nhà cung cấp khác nhau Giảm chi phí công nghệ</p>

➤ **Loosely coupled**

Lợi ích thương mại – Tăng cường quan hệ	Lợi ích kỹ thuật – Giảm chi phí bảo trì
<p>Tạo thuận tiện cho việc thêm vào hay thay đổi đối tác. Thay đổi về mặt hệ thống không hưởng đến nghiệp vụ</p>	<p>Giảm chi phí bảo trì Giảm ảnh hưởng của thay đổi Sử dụng lại các thành phần có sẵn</p>

➤ **Hỗ trợ đa kết nối**

Lợi ích thương mại	Lợi ích kỹ thuật – Tiết kiệm chi phí
	<p>Giảm số lượng phần mềm, công cụ, kỹ năng cần thiết trên nhiều lĩnh vực khác nhau. Tiếp cận vững chắc cho mọi bối cảnh bài toán Một kiến trúc chung cho mọi bối cảnh bài toán (ví dụ như bảo mật)</p>

➤ **Tự mô tả**

Lợi ích thương mại – Tiếp cận thị trường	Lợi ích kỹ thuật – Rút ngắn chu kỳ phát triển
<p>Tăng thời gian tiếp cận thị trường bởi vì các kết nối đến đối tác và khách hàng bây giờ trở nên nhanh hơn và tự động. Tạo điều kiện thuận tiện cho đối tác hợp tác làm ăn.</p>	<p>Giảm công sức phát triển vì dịch vụ được tự động hoá. Giảm ảnh hưởng của những thay đổi, tự động phản ứng với những thay đổi Dịch vụ có thể được triệu gọi tự động mà không cần đến sự can thiệp của nhân viên phát triển phần mềm</p>

➤ **Tự động tìm kiếm**

Lợi ích thương mại	Lợi ích kỹ thuật – Rút ngắn chu kỳ phát triển
<p>Tạo điều kiện thuận tiện khách hàng tìm đến dịch vụ của ta. Dễ dàng tìm kiếm các đối tác mới và dịch vụ của họ.</p>	<p>Giảm hoặc không cần loại bỏ công sức phát triển hỗ trợ các kết nối mới.</p>

B.4 SOAP

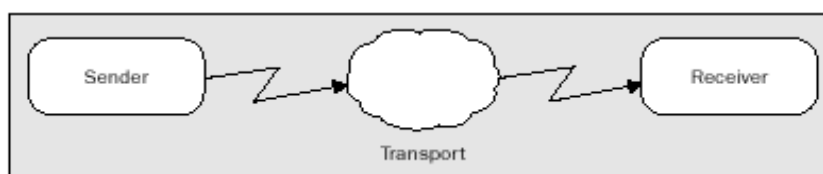
SOAP là một đặc tả việc sử dụng các tài liệu XML theo dạng các thông điệp. Bản thân SOAP không định ra các ngữ nghĩa ứng dụng hoặc cách cài đặt chi tiết. SOAP cung cấp một cơ chế đơn giản và gọn nhẹ cho việc trao đổi thông tin có cấu trúc và định dạng giữa các thành phần trong một môi trường phân tán sử dụng XML. SOAP được thiết kế dựa trên những chuẩn nhằm giảm chi phí tích hợp các hệ thống phân tán xây dựng trên nhiều nền tảng khác nhau ở mức càng thấp càng tốt. Đặc tả về SOAP định nghĩa một mô hình trao đổi dữ liệu dựa trên 3 khái niệm cơ bản: các thông điệp là các tài liệu XML, chúng được truyền đi từ bên gửi đến bên nhận, bên nhận có thể chuyển tiếp dữ liệu đến nơi khác.

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. Sau đây là một ví dụ về một thông điệp SOAP :

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  soap:encodingStyle="http://schemas.xmlsoap.org/soap
    /encoding/">

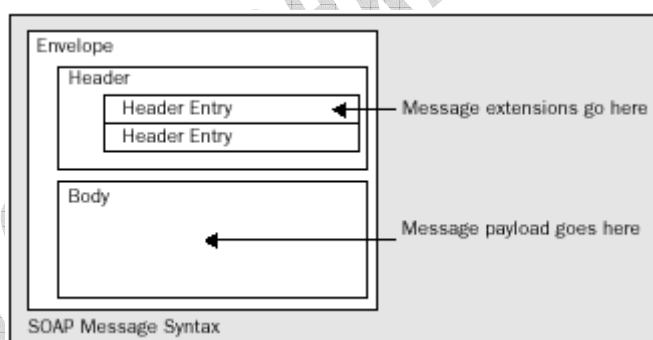
  <soap:Body>
    <w:Greeting xmlns:w="http://www.wrox.com/helloworld/">
      <w:message>Hello world!</w:message>
    </w:Greeting>
  </soap:Body>
</soap:Envelope>
```

Khi trao đổi các thông điệp SOAP, có hai thành phần liên quan: bên gửi và bên nhận. Thông điệp sẽ được chuyển từ bên gửi sang bên nhận. Đây là ý niệm đơn giản nhất trong trao đổi thông điệp SOAP. Trong nhiều trường hợp, kiểu trao đổi này không cung cấp đủ chức năng. Nhưng đây là mô hình cơ bản, dựa trên đó sẽ phát triển các mô hình trao đổi phức tạp hơn



Hình B-1 – Một SOAP Operation đơn giản

Một cấu trúc SOAP được định nghĩa gồm các phần: <Envelope> , <Header> và <Body>.



Hình B-2 – Cấu trúc thông điệp SOAP

Envelop là thành phần gốc của một thông điệp SOAP, nó chứa các thành phần Header và Body. Thành phần Header là một cơ chế mở cho phép thêm các tính năng vào bên trong một thông điệp SOAP. Mỗi thành phần con của Header gọi là một Header Entry. Các Header Entry dùng để diễn giải , quy định một số ngữ nghĩa của thông điệp SOAP. Các ứng dụng có thể xử lý và định tuyến các thông điệp dựa trên thông tin header và thông tin bên trong thông điệp đó. Đây là ưu điểm mà các mô hình kiến trúc như DCOM, CORBA và RMI không có được, vì các protocol header của chúng phải được chỉ định chi tiết cho mỗi ứng dụng.

B.5 WSDL

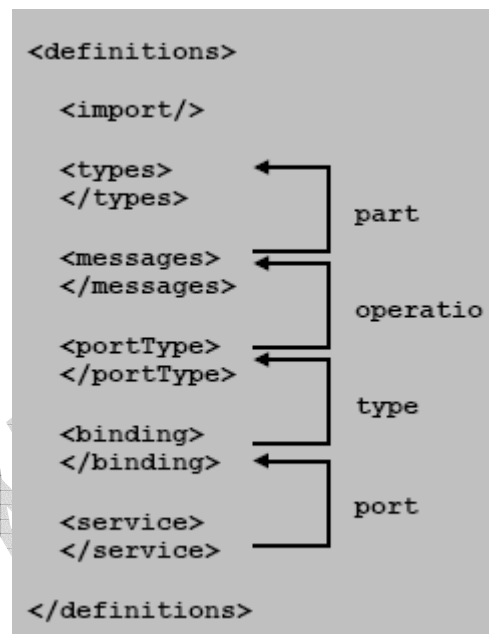
Web Service Description Language (WSDL) định nghĩa một lược đồ XML dùng để mô tả cấu trúc các dịch vụ theo dạng XML. Nó cung cấp thông tin cho các hệ thống phân tán và cho phép các ứng dụng trao đổi với nhau một cách tự động. Trong khi SOAP tập trung vào việc trao đổi thông tin giữa bên yêu cầu và bên cung cấp thì WSDL mô tả dịch vụ được cung cấp bởi bên cung cấp và được xem như một công thức để phát sinh các thông điệp SOAP đến các dịch vụ.

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 những hành vi. Hành vi tương ứng với “operation” và dữ liệu tương ứng với “message”. Một tập các hành vi 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. Kể 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 và thông tin cụ thể.

- Thông tin trừu tượng
 - a. types
 - b. messages
 - c. portType
- Thông tin cụ thể
 - d. bindings
 - e. services

Các thành phần chứa những tham chiếu lẫn nhau như trong hình.



Sau đây là một ví dụ mẫu về một đặc tả WSDL.

```
<?xml version="1.0" encoding="utf-8"?>
<definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:s0="http://tempuri.org/services/loanassessor"
targetNamespace="http://tempuri.org/services/loanassessor">
  <message name="checkSoapIn">
    <part name="name" type="s:string" />
  </message>
  <message name="checkSoapOut">
    <part name="result" type="s:boolean" />
  </message>
  <portType name="LoanAssessorSoap">
    <operation name="check">
      <input message="s0:checkSoapIn" />
      <output message="s0:checkSoapOut" />
    </operation>
  </portType>
  <binding name="LoanAssessorSoap" type="s0:LoanAssessorSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document" />
    <operation name="check">
      <soap:operation
soapAction="http://tempuri.org/services/loanassessor/check"
style="document" />
      <input>
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal" />
      </output>
    </operation>
  </binding>
  <service name="LoanAssessor">
    <port name="LoanAssessorSoap" binding="s0:LoanAssessorSoap">
      <soap:address
location="http://localhost/LoanAssessor/LoanAssessor.asmx" />
    </port>
  </service>
</definitions>
```

B.6 UDDI

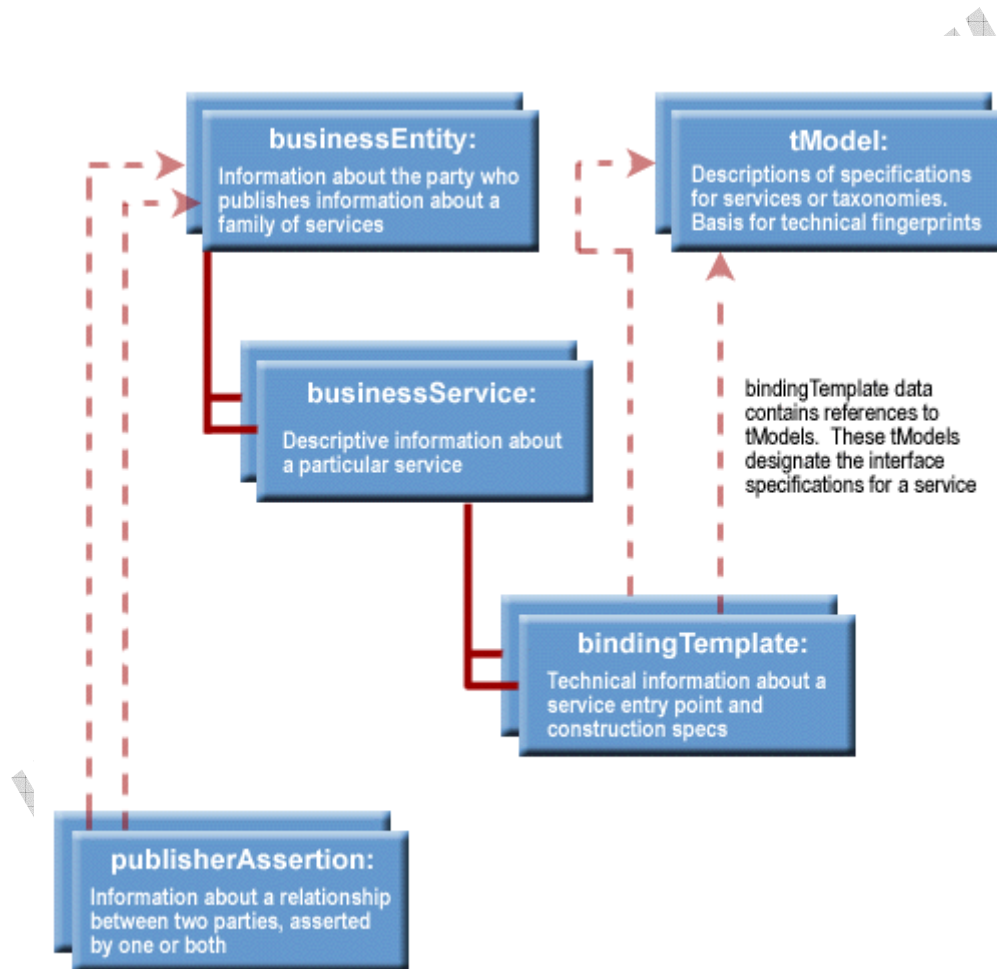
Về cơ bản Universal Description, Discovery, and Intergration (UDDI) là một nơi mà các tổ chức đăng ký và tìm kiếm các Web Service. Nó đóng vai trò như service broker cho phép người sử dụng dịch vụ tìm đúng nhà cung cấp dịch vụ cần tìm. UDDI hỗ trợ chức năng:

- Thực hiện tìm kiếm, định vị những doanh nghiệp cung cấp dịch vụ hay sản phẩm theo phân 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 Web site hoặc sử dụng các công cụ lập trình sử dụng các dịch vụ theo đúng đặc tả UDDI programmer's API.

Lược đồ XML UDDI định nghĩa bốn loại thông tin cơ bản để kết nối đến Web service.



Như hình trên, cấu trúc thông tin bao gồm :

- **Business entity**

Một business entity chứa thông tin về công bao gồm tên công ty, một đoạn mô tả ngắn gọn và một vài thông tin liên lạc cơ bản (địa chỉ, số điện thoại, email, v.v....). Mỗi doanh nghiệp được cấp một định danh duy nhất, ví dụ như số D-U-N-S.

- **Business service**

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.

- **Binding template**

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.

- **tModels**

Mục đích của tModels là dùng để liên kết đến metadata bên ngoài UDDI. Thành phần quan trọng nhất của tModels là một URL trỏ đến một tài liệu mô tả thông tin metadata. Tài liệu này có thể là tài liệu bất kỳ 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.

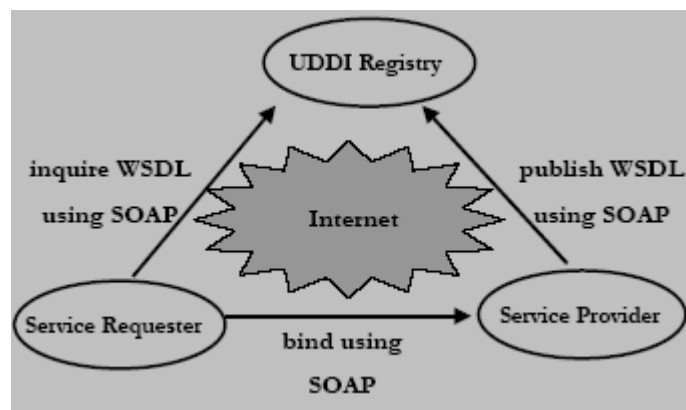
```
<?xml version="1.0" encoding="utf-8" ?>
<businessEntity xmlns="urn:uddi-org:api"
  businessKey="BBBBBBBBB-BBBB-BBBB-BBBB-BBBBBBBBBBBB">
  <name>Sample Services</name>
  <description xml:lang="en"> a description </description>
  <businessServices>
    <businessService businessKey="..." serviceKey="...">
      <name> Symbol service </name>
      <description> a description </description>
      <bindingTemplates>
        <bindingTemplate>
          .....
          <accessPoint urlType="http">
            http://anyURL/symbol
          </accessPoint>
          <tModelInstanceDetails>
            <tModelInstanceInfo
              tModelKey="123123">
            </tModelInstanceInfo>
          </tModelInstanceDetails>
          <tModel authorizedName="..."
            operator="..." tModelKey="123123">
            <name> StockQuote Service</name>
            <description xml:lan="en">
              WSDL description of a Symbol
              lookup service
            </description>
```

```

<overviewDoc>
  <description xml:lang="en">
    WSDL source document
  </description>
  <overviewURL>
    http://anyURL/symbol.wsdl
  </overviewURL>
</overviewDoc>
<categoryBag>
  <keyedReference tModelKey="UUDI:12313" keyName="uudi-
org:types" keyValue="wsdlSpec" />
</categoryBag>
</tModel>
</bindingTemplate>
</bindingTemplates>
</businessService>
</businessServices>
</businessEntity>

```

Tóm lại SOAP, WSDL và UDDI có thể kết hợp với nhau theo sơ đồ sử dụng sau :



1. Nhà cung cấp Web Service mô tả Web Service trong một tài liệu WSDL và đăng ký nó lên một UDDI bằng cách sử dụng Publisher's API.
2. Một người sử dụng dịch vụ UDDI Inquiry API để tìm thông tin về nhà cung cấp dịch vụ thích hợp. Nếu có, nó sẽ tìm kiếm tiếp tModel rồi từ đó lấy ra tài liệu mô tả WSDL.
3. Một yêu cầu dạng SOAP được tạo ra dựa trên tài liệu mô tả WSDL.
4. Yêu cầu SOAP trên sẽ được gửi đến nhà cung cấp dịch vụ và được xử lý.

B.7 Một số chuẩn Web services mới

- **WS-Security** : Mô tả cách thêm vào các header signature và mã hoá vào bên trong thông điệp dạng SOAP, cách thêm vào các token bảo mật như X.509, chứng thực và Keberos.
- **WS-Policy** : Mô tả khả năng và những ràng buộc về bảo mật và chính sách nghiệp vụ giữa thành phần trung gian và địa chỉ endpoint.
- **WS-Trust** : Một framework cho các mô hình trust, cho phép các Web Service cộng tác một cách an toàn và bảo mật.
- ...

B.8 SOA và Web Service

SOA là một kiến trúc phần mềm, bao gồm một tập các nguyên tắc thiết kế độc lập với kỹ thuật và công nghệ nhằm liên kết các dịch vụ. SOA bắt đầu với việc định nghĩa thành phần giao tiếp (interface) và sau đó, xây dựng kiến trúc hệ thống như là sự liên kết của các thành phần interface, phần thực thi của các interface và cách thức tương tác giữa các interface. Trong khi đó, Web services lại là một tập hợp các kỹ thuật và công nghệ (SOAP, WSDL, UDDI, HTTP...) được dùng để hiện thực hóa các nguyên tắc thiết kế của SOA.

Trong thực tế, khái niệm SOA không phải là một khái niệm hoàn toàn mới, mà đã xuất hiện cách đây khá lâu. Và trong quá khứ đã có rất nhiều kỹ thuật khác được dùng để triển khai một hệ thống SOA như :

- Distributed object : như CORBA, J2EE, COM/DCOM
- Message-oriented middleware (MOM) : WebSphere MQ, Tibco, Rendezvous
- TP monitor: CICS, IMS, Encinia, Tuxedo.
- B2B platform: ebXML, RosettaNet

Trong phần này, sẽ giải thích rõ một hệ thống SOA dựa trên Web services khác ra sao với một hệ thống SOA dựa trên các kỹ thuật cũ trước đây.

- Cách tiếp cận dựa trên API và dựa trên nghiệp vụ
 - ▶ Sự khác nhau cơ bản giữa việc triển khai một hệ thống SOA dựa trên các kỹ thuật cũ với khi dùng web service đó là ở cách tiếp cận. Các kỹ thuật trước tiếp cận theo cách sử dụng các hàm API trong khi kỹ thuật web services lại sử dụng cơ chế thông điệp. Lợi ích của cách tiếp cận mới đó là tạo ra được nhiều mô hình tương tác đa dạng, linh hoạt hơn như có thể xây dựng nhiều trạm điều khiển trung gian, xử lý vấn đề định tuyến, orchestration và tương quan, ngoài ra còn hỗ trợ độc lập với môi trường truyền dẫn (transport independence).
 - ▶ Tương tác giữa nhà cung cấp dịch vụ và đối tượng sử dụng dịch vụ không còn đơn giản là những kết nối điểm-điểm nữa.
 - ▶ Hiệu suất hoạt động của hệ thống được tốt hơn bởi sự hỗ trợ của cơ chế định tuyến giúp định tuyến cho các luồng trao đổi dịch vụ một cách hiệu quả.
 - ▶ Với những hỗ trợ trong việc sắp xếp, kết hợp, ... và quản lý các dịch vụ sẽ giúp xây dựng toàn bộ các qui trình nghiệp vụ một cách nhanh chóng, dễ dàng và hiệu quả hơn là thiết kế từng thành phần.
 - ▶ Việc xác định một dịch vụ sẽ hiệu quả hơn thông qua việc sử dụng các thông tin tương quan (correlation information)
- Sử dụng các dữ liệu mô tả (metadata) để tăng tính linh hoạt
 - ▶ Một điểm yếu nữa của các tiếp cận dựa trên API khi triển khai một hệ thống SOA đó là sự hạn chế (thậm chí là hoàn toàn không có) trong thông tin mô tả dịch vụ.
 - ▶ Web services dựa trên lược đồ XML để hỗ trợ dữ liệu tự mô tả. Ngoài ra Web services hỗ trợ nhiều chuẩn như WSDL để mô tả thông tin web service, UDDI để tạo nền tảng cho việc quản lý các web service cũng như

các thông tin mô tả liên quan của web service. Từ đây sẽ hỗ trợ tốt hơn cho quá trình sử dụng và quản lý các tài nguyên mới lẫn cũ, thông qua:

- Tích hợp ngữ nghĩa: các nhà quản lý có thể chỉ định thông tin nào cần trao đổi và cách thức trao đổi sẽ như thế nào.
- Chuyển đổi thông tin: hỗ trợ việc chuyển đổi các định dạng dữ liệu trong quá trình trao đổi giữa các hệ thống khác nhau.

- Tăng khả năng tương thích và tái sử dụng:

- ▶ Các hệ thống SOA không phải là các hệ thống kín (monolithic) và cũng không hẳn là phải được xây dựng mới hoàn toàn. Điều này có nghĩa là hệ thống phải có khả năng tương thích cao để có thể tích hợp với các hệ thống khác bên ngoài cũng như là các ứng dụng, tài nguyên sẵn có nhằm thực hiện tái sử dụng một cách có hiệu quả.
- ▶ Các kỹ thuật trước gặp rất nhiều khó khăn trong việc giải quyết vấn đề này, thậm chí là không thể, bởi bản thân các kỹ thuật đó không đưa ra được một chuẩn chung để các hệ thống khác nhau (về công nghệ, kỹ thuật, định dạng dữ liệu) có thể tương tác được với nhau.
- ▶ Trong khi đó Web service có thể giải quyết các yêu cầu trên một cách dễ dàng, hiệu quả, tiết kiệm được thời gian lẫn chi phí đó là thực hiện bao bọc những chức năng, ứng dụng, dữ liệu, ... thành các services có thể thực hiện quá trình tương tác với các hệ thống mới thông qua các chuẩn của Web service (SOAP, WSDL...)

- Vấn đề sử dụng chuẩn

- ▶ Các kỹ thuật trước cũng đưa ra một định dạng thông điệp dùng trong quá trình giao tiếp của các dịch vụ. Nhưng thông điệp chung này được xây dựng dựa trên những chuẩn riêng, ràng buộc chặt chẽ với ngôn ngữ và môi trường của hệ thống. Điều này dẫn đến sự rối rắm, trùng lặp và khó tương thích giữa các chuẩn của các hệ thống.

- ▶ Trong khi đó, web service được xây dựng dựa trên các chuẩn đặt được sự nhất trí của cộng đồng phát triển (định dạng thông điệp dựa trên các chuẩn của W3C như là XML, và môi trường truyền thông điệp dựa trên SOAP).
- ▶ Ngoài ra các chuẩn này có thể tương thích tốt với nhau (ví dụ như WS-Security và WS-Policy). Hơn nữa, các chuẩn của Web service đều được thiết kế để có khả năng mở rộng để có thể phối hợp hoặc tích hợp với các chuẩn mới trong tương lai. Điều này có nghĩa rằng, sẽ không có chuyện lệ thuộc vào một nhà phát triển và khả năng tùy biến sẽ được hỗ trợ tối đa. Tinh thần này phù hợp với nguyên tắc thiết kế của SOA : “Chỉ cần đầu tư một lần vào xây dựng cơ sở hạ tầng, rồi sau này mọi chuyện đều trở nên dễ dàng.”
- ▶ Như vậy, tính tương thích và khả năng tích hợp cao đã được thiết kế và xây dựng trong kiến trúc Web service.

Ta sẽ thấy rõ hơn những lợi ích của việc triển khai một hệ thống SOA dựa trên Web service qua việc so sánh với hai kỹ thuật khác là CORBA và WebSphere MQ

Cơ sở so sánh	WebSphere MQ	CORBA	XML Web services
Chuẩn mở	Không hỗ trợ	Có hỗ trợ (Chuẩn của tổ chức OMG xây dựng)	Có hỗ trợ (do tổ chức W3C, OASIS đưa ra)
Hỗ trợ nhiều ngôn ngữ lập trình	Có	Có	Có
loosely-coupled	Có thể	Có thể	Có
Hợp đồng dịch vụ			
Định nghĩa hợp đồng độc lập với kỹ thuật?	Word, Excel, Access	CORBA IDL	WSDL, Xml Schema, WS-Policy
Định nghĩa thông điệp và các tham số	COBOL	CORBA IDL	WSDL, XmlSchema
Kiểm tra định dạng thông điệp và tham số	Phải tự xây dựng	CORBA IDL compiler	XmlSchema và Xml parsers.
Phân tích thông điệp và tham số	Phải tự xây dựng	Xử lý tự động dựa theo các ràng buộc về ngôn ngữ	Tự động dựa vào các mô hình DOM, SAX, JAAS, JAX-RPC...
Quản lý dữ liệu			
Kiểu dữ liệu	Không có	CORBA IDL	XmlSchema và WSDL
Kiểm tra dữ liệu	Không có	Có hỗ trợ kiểm tra tham số của các phương thức	XmlSchema và Xml parser
Truy vấn dữ liệu	Không có	Không có	XPath và XQuery

Quản lý dịch vụ			
Lưu trữ interface	Không có	CORBA interface repository	UDDI
Định danh dịch vụ	Không có	CORBA naming service	UDDI
Cung cấp dịch vụ ra bên ngoài	Không	CORBA trading service	UDDI
Vấn đề bảo mật			
Chứng thực	Userid/password	IIOP/TLS	HTTPs, WS-Security
Phân quyền	Không có	IIOP/TLS	SAML, XACML, WS-Security
Bảo vệ dữ liệu	Không có	IIOP/TLS	HTTPs, WS-Security
Tích hợp dữ liệu	Không có	IIOP/TLS	HTTPs, WS-Security
Các kiểu tương tác			
Một chiều, đồng bộ	Có	Có	Có
Request/Response	Có, nhưng không được hỗ trợ mạnh, quản lý thông điệp ở mức ứng dụng	Có	Có
Một chiều, bất đồng bộ	Có, và hỗ trợ mạnh	Có, nhưng hỗ trợ yếu	WS_Reliability, WS-ReliableMessaging
Publish/subscribe	Có, nhưng yếu	Có	WS-Eventing, WS-Notification
Giao tiếp ở mức dịch vụ			
Chuyển đổi định dạng dữ liệu giữa các nền tảng và ngôn ngữ lập trình	Mức độ đơn giản: ASCII \leftrightarrow EDC/DIC	Có	Các thông điệp dạng text nên không cần chuyển đổi
Chất lượng của dịch vụ			
Quản lý phiên làm việc	Có	Có	WS-SecureConversation, WS-Context
Cân bằng tải	Có	Có	Không giải quyết bởi chuẩn, việc hỗ trợ tùy thuộc vào các nhà cung cấp sản phẩm
Đảm bảo dữ liệu truyền	Có	Có	WS-Reliability, WS-ReliableMessaging
Quản lý giao tác	Có	Có	WS-AT, BA, C và WS-CAF.
Quản lý dịch vụ	Không	Không	Web services distributed management (WSDM) và WS-Management.

Phụ lục C

ĐỊNH DẠNG CÁC FILE TRONG PROJECT BPEL DESIGNER

File Process Deployment Descriptor (.pdd)

Với mỗi tiến trình BPEL cần triển khai ta cần một mô tả triển khai tiến trình (.pdd). File XML này mô tả thông tin cho BPEL engine biết về tiến trình BPEL cần triển khai. Mỗi tiến trình (mỗi file .bpel) có file .pdd riêng của nó. Thành phần <process> sẽ chứa toàn bộ các partner link và tham chiếu WSDL. Lược đồ XML cho file .pdd có thể tìm thấy ở <http://schemas.active-endpoints.com/pdd/2004/09/pdd.xsd>.

```
<?xml version="1.0" encoding="utf-8" ?>
<process name="qname" location="relativeDeploymentLocation">
  <partnerLinks>
    <partnerLink name="ncname">
      <partnerRole endpointReference="static|dynamic|invoker|principal">
        [... endpoint reference....]?
      </partnerRole>?
      <myRole service="name" allowedRoles="namelist"?
        binding="MSG|RPC"/>?
    </partnerLink>+
  </partnerLinks>
  <wsdlReferences>
    <wsdl namespace="uri" location="uri"/>+
  </wsdlReferences>?
</process>
```

relativeDeploymentLocation là đường dẫn tương đối đến file BPEL. Thông thường, giá trị này chỉ là tên file BPEL có trong cùng thư mục với file .pdd.

Bởi vì *qname* là một tên định danh cho nên ta phải sử dụng thuộc tính *xmlns* để xác định the namespace. Ví dụ

```
<process name="bpelns:loanApprover"
  location="approver.bpel"
  xmlns:bpelns="http://acme.com/loanprocessing">
  <!-- ... -->
</process>
```


The partner link mô tả các role được sử dụng trong mỗi partner của tiến trình.

Partner Role	Endpoint reference
static	Được mô tả trong bản mô tả cài đặt (BPEL4WS)
dynamic	Được định nghĩa trong tiến trình (BPEL4WS)
invoker	Lấy từ thông tin SOAP header của partner (WS-Addressing)
principal	Dò từ file Partner Definition (.pdef) dựa trên xác nhận chứng thực

Ví dụ ứng dụng BPEL xử lý cho vay thì thông tin về partner link như sau:

```
<partnerLinks xmlns:lns="http://loans.org/wsd/loan-approval">
  <!-- Partner Link for inbound request from customer -->
  <partnerLink name="customer" type="lns:loanApprovalLinkType">
    <myRole service="ApproveLoan" allowedRoles="" binding="RPC" />
  </partnerLink>
  <!-- Partner Link for outbound request to approver -->
  <partnerLink name="approver" type="lns:loanApprovalLinkType">
    <partnerRole endpointReference="static">
      <wsa:EndpointReference
        xmlns:approver="http://tempuri.org/services/loanapprover">
        <wsa:Address>approver:anyURI</wsa:Address>
        <wsa:ServiceName
          portName="SOAPPort">approver:LoanApprover</wsa:ServiceName>
        </wsa:EndpointReference>
      </partnerRole>
    </partnerLink>
    <!-- Partner Link for outbound request to assessor -->
    <partnerLink name="assessor" type="lns:loanAssessorLinkType">
      <partnerRole endpointReference="static">
        <wsa:EndpointReference
          xmlns:assessor="http://tempuri.org/services/loanassessor">
          <wsa:Address>assessor:anyURI</wsa:Address>
          <wsa:ServiceName
            portName="SOAPPort">assessor:LoanAssessor</wsa:ServiceName>
          </wsa:EndpointReference>
        </partnerRole>
      </partnerLink>
    </partnerLinks>
```

Thành phần <wsdlReferences> liệt kê mọi file WSDL tham chiếu bởi tiến trình BPEL. Nó được dùng bởi BPEL engine để tạo ra các đại diện WSDL trong bộ nhớ. Sau đây là ví dụ <wsdlReferences> cho tiến trình xác nhận cho vay:

```
<wsdlReferences>
  <wsdl namespace="http://tempuri.org/services/loandefinitions"
    location="wsdl/loandefinitions.wsdl"/>
  <wsdl namespace="http://loans.org/wsd/loan-approval"
    location="wsdl/loanapproval.wsdl"/>
  <wsdl namespace="http://tempuri.org/services/loanapprover"
    location="wsdl/loanapprover.wsdl"/>
  <wsdl namespace="http://tempuri.org/services/loanassessor"
    location="wsdl/loanassessor.wsdl"/>
</wsdlReferences>
```

Trong mỗi thành phần <wsdl>, thuộc tính namespace không cần sử dụng tới. Thuộc tính location có thể là tên một file hay một địa chỉ URL bất kì.

File wsdlCatalog.xml

File wsdlCatalog.xml được chứa trong thư mục META-INF.

```
<wsdlCatalog>
  <wsdlEntry location="string"
    classpath="slash/separated/classpath/filename.wsdl" />*
</wsdlCatalog>
```

Thuộc tính location chỉ đến một file WSDL theo một trong hai cách. Giá trị của nó có thể là:

- Thuộc tính location của một thành phần <wsdl> trong phần wsdlReferences của một file .pdd
- Thuộc tính location của một element <import> bên trong một file WSDL

Khi nạp một file WSDL lúc triển khai, engine xử lý đọc tham chiếu WSDL từ file .pdd và sử dụng thuộc tính location của thành phần <wsdl> như là khoá đến WSDL catalog. Nếu WSDL catalog chứa địa chỉ tương ứng thì engine sẽ nạp file WSDL từ đường dẫn tương ứng. Nếu không có, engine sẽ tự hiểu đây là địa chỉ tuyệt đối URL và sẽ nạp file WSDL từ địa chỉ này. Sau đây là một ví dụ về file wsdlCatalog.xml.

```
<wsdlCatalog>
  <wsdlEntry location="wsdl/loanapproval.wsdl"
    classpath="wsdl/loanapproval.wsdl" />
  <wsdlEntry location="wsdl/loanapprover.wsdl"
    classpath="wsdl/loanapprover.wsdl" />
  <wsdlEntry location="wsdl/loanassessor.wsdl"
    classpath="wsdl/loanassessor.wsdl" />
  <wsdlEntry location="wsdl/loandefinitions.wsdl"
    classpath="wsdl/loandefinitions.wsdl" />
</wsdlCatalog>
```

File Partner Definition (.pdef)

```
<pre><partnerDefinition principal="name">
  <partnerLinkType name="qname">
    <role name="ncname" authtype="basic" username="xx" password="yy">
      ... endpoint reference....
    </role>*
  </partnerLinkType>*
</partnerDefinition></pre>
```

Partner link mô tả mối quan hệ giữa các. File định nghĩa partner không nhất thiết phải cần có trong mọi tiến trình BPEL processes, chỉ những tiến trình có endpoint reference dạng principal mới sử dụng chúng. Mỗi khi có yêu cầu chứng thực thì file này được sử dụng để cung cấp thông tin chứng thực.