UNIVERSIDADE FEDERAL DO CEARÁ
Centro de Tecnologia
Departamento de Engenharia de Teleinformática
Programa de Pós-Graduação em Engenharia de Teleinformática - PPGETI

Barramento de Serviços Federados para integração federativa de sistemas distribuídos

Josênio Candido Camelo

Fortaleza – CE,
Janeiro/2008

Josênio Candido Camelo

Barramento de Serviços Federados para integração federativa de sistemas distribuídos

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Teleinformática da Universidade Federal do Ceará (PPGETI/UFC), como requisito parcial para a obtenção do título de Mestre em Engenharia de Teleinformática

Orientador:
José Neuman de Sousa
Co-orientador:
Danielo Gonçalves Gomes

Fortaleza – CE, 2008

Resumo

Esta dissertação apresenta uma proposta de middleware de comunicação baseado em Enterprise Service Bus (ESB) para sistemas federados, isto é, formados por sistemas de diferentes organizações. Este trabalho não aborda o problema clássico de sistemas federados, cujo enfoque principal é autenticação e a segurança, mas sim uma necessidade crescente de intercomunicação de serviços heterogêneos. O middleware proposto, chamado de Federated Service Bus (FSB), faz uso de ESBs internos para permitir o isolamento, aplicação de políticas e roteamento de cada domínio que compõe o sistema federado, visando separar interesses e evitar conflitos. Nossa proposta é modelada por redes de Petri coloridas, o que lhe atribui confiabilidade de simulação e de validação com base em um modelo formal matemático. Assim, ganhos significativos foram obtidos na implementação com o uso de web services e BPEL (Business Process Execution Language). A modelagem com redes de Petri coloridas não só validou o fluxo, como o documentou em detalhes e possibilitou a diminuição no número de erros. Por fim, enquadramos o FSB em arquiteturas consolidadas com SOA (Service Oriented Achitecture), EDA (Event-Driven Architecture) e NGOSS (Next Generation Operation System and Software).

Palavras-chave: barramento federados, sistemas distribuídos, integração, modelagem

Abstract

This work presents the Federated Service Bus (FSB), a communication middleware based on Enterprise Service Bus (ESB) for federated systems. We do not address the classic problem of federated systems, focused mainly on authentication and security, but a growing need for heterogeneous service intercommunication. The proposed middleware makes use of internal ESBs to allow the isolation, application of policies and routing of each domain that comprises the federal system, seeking separate interests and avoid conflicts. Our proposal is modeled using coloured Petri nets, which gives it reliability of simulation and validation based on a formal mathematical model. Thus, significant gains were achieved in the implementation with the use of web services and BPEL (Business Process Execution Language). The modeling with coloured Petri nets not only validated the flow as allowed a error reduction. Finally, the FSB is embedded with SOA (Service Oriented Achitecture), EDA (Event-Driven Architecture) and NGOSS (Next Generation Operation System and Software).

Keywords: Distributed systems, Federated Service Bus, modeling, integration

Índice

Lista de figuras

Lista de tabelas

Lista de acrônimos

AS - *Application Servers*

BPEL - *Business Process Execution Language*

BPM - *Business Process Management*

CPN - redes de Petri coloridas (Coloured Petri Network)

EAI - *Enterprise Aplication Integration*

EAI - *Extraction Tranformation and Load*

EDA - *Event-Driven Architecture*

EDA - *Event-Driven Architecture*

ESB - *Enterprise Service Bus*

FSB - *Federated Service Bus*

J2CA - *J2EE Connector Architecture*

JBI - *Java Business Integration*

JCP - *Java Community Process*

JMS - *Java Message Service*

JSR - *Java Specification Request*

MOM - *Message Oriented Midleware*

NGOSS - *Next Generation Operation System and Software*

NGOSS – *Next Generation Operation System and Software*

NGOSS TNA – NGOSS *Technology Neutral Architecture*

NMR - *Normalized Message Router*

ORB - *Object Request Broker*

OSS - *Operational Support Systems*

P2P – Peer-to-Peer

PN - redes de Petri (Petri Network)

QoS - *Quality of Service*

RM-ODP - *Reference Model for Open Distributed Processing*

RPC - *Remote Procedure Call*

SID - *Shared Information Model and Data model*

SNMP - *Simple Management Protocol*

SOA - *Service Oriented Achitecture*

SOA - *Service Oriented Achitecture*

SOAP - *Simple Object Access Protocol*

TI - Tecnologia da Informação

TMF - *TeleManagement Forum*

TPM – *Transaction Processing Monitor*

TR – Transição

TSP - *Telecom Service Provider*

UDDI - *Universal Discovery and Description Integration*

UML - *Unified Modeling Language*

WS – *Web Service*

WSDL - *Web Service Description Language*

XML - e*Xtensible Markup Language*

1 Introdução

1.1 O Problema

Durante a década de 80, o crescimento exponencial do poder de processamento das máquinas resultou na proliferação de servidores departamentais autônomos em processamento de dados. No final dos anos 90, foram criados os sistemas distribuídos de "n" camadas. Até então, os sistemas eram isolados, implicando na necessidade do uso de ferramentas de integração de aplicações corporativas (*Enterprise Aplication Integration* – EAI) e de extração, transformação e carga (*Extraction Tranformation and Load* – ETL) para se conseguir uma visão consolidada das informações empresariais [Nadhan 2004]. Nesse contexto, percebe-se a utilização cada vez mais freqüente de sistemas autônomos. Atualmente, a arquitetura orientada a serviços (*Service Oriented Achitecture* – SOA) está mudando o modelo de negócio em várias empresas [SUN 2008], permitindo integrar virtualmente todos os recursos tecnológicos da corporação para o intercâmbio de informações e a delegação de responsabilidades em ambientes corporativos e federativos.

As ferramentas tradicionais de integração de aplicações e *midleware* vêm se tornando insuficientes devido às novas abordagens de gestão de TI (Tecnologia da Informação) e entrega de serviços, como SOA, gerenciamento de processos e negócio (*Business Process Management* - BPM) e *web services*. Há cerca de três anos, *brokers* baseados em padrões proprietários, replicações de dados e funções de negócio tornaram-se ineficientes em relação às demandas das áreas de negócio devido à necessidade de

maior velocidade e eficácia a pressões competitivas, inovadoras e regulatórias [Chapel 2006].

Dentre as propostas de midleware com suporte a federação, os principais pontos atacados são os requisitos tradicionais de federação, como autenticação e segurança, não havendo ênfase na resolução conjunta de outros problemas de integração de sistemas como o mapeamento de entidades. Neste cenário, várias propostas de resolução de problemas de integração e federação foram proposta, mas ainda possuem falhas em atender a determinados contextos.

1.2 Objetivos

Este trabalho apresenta uma proposta de *middleware* de comunicação baseado em *Enterprise Service Bus* (ESB) para sistemas federados, isto é, aqueles formados por sistemas de diferentes organizações. Entretanto não é nosso foco abordar o problema clássico de sistemas federados, onde é abordado principalmente autenticação e segurança, mas sim uma necessidade crescente de intercomunicação de serviços heterogêneos. O *middleware* aqui proposto, chamado de *Federated Service Bus* (FSB), faz uso de ESBs internos para permitir o isolamento, aplicação de políticas e roteamento de cada domínio que compõe o sistema federado, de forma a separar interesses e evitar conflitos.

Nossa proposta é modelada por redes de Petri coloridas, o que lhe atribui confiabilidade de simulação e de validação com base em um modelo formal matemático. Assim, ganhos significativos são obtidos na implementação com o uso de *web services* e BPEL (*Business Process Execution Language*). A modelagem com redes

de Petri coloridas não só valida o fluxo, como o documenta em detalhes e possibilita a diminuição no número de erros. Por fim, enquadramos o FSB em arquiteturas consolidadas como SOA (*Service Oriented Achitecture*), EDA *(Event-Driven Architecture)* e NGOSS (*Next Generation Operation System and Software).*

1.3 Trabalhos relacionados

Visto que nossa proposta utiliza amplamente os princípios do ESB, este é o trabalho relacionado de base. No mercado temos a especificação *Java Business Integration* (JBI), a partir da JSR-208 [JCP 2008], que é uma implementação de ESB orientada a serviços, e a usamos como base de nossa implementação. Esta especificação vem tendo uma grande atenção tanto de marketing, quanto dos servidores de aplicação Java que tem cada vez mais suporte a essa especificação.

O artigo "Autenticação e Autorização em Arquiteturas Orientadas a Serviço através de Identidades Federadas" [Camargo 2007] complementa nosso trabalho no tocante ao problema clássico de federações. Este trabalho define um modelo de autenticação e autorização em sistemas orientados a serviços, em ambientes de larga escala, de forma transparente e interoperável.

De fato nossa principal contribuição é a de constituir de um *workflow* compatível com o ESB de forma a tornar possível usar um ESB em ambientes federados. Analisando nossa proposta sob esse ângulo, podemos citar o trabalho "Infra-Estrutura com Segurança de Funcionamento para Cooperação de Serviços Web" [Alchieri 2007] o qual foi tratado o problema de federação em *workflows*. Este artigo

define uma infra-estrutura para coordenação de serviços web segura e confiável, isto é, tolerante a falhas e intrusões.

1.4 Estrutura do texto

Esta dissertação está dividida em cinco capítulos, sendo que o primeiro é a Introdução, o segundo capítulo trata dos fundamentos conceituais utilizados ao longo da dissertação, o terceiro capítulo apresenta a proposta, a qual é modelada e simulada no quarto capítulo. Finalmente, no quinto capítulo, são apresentadas as conclusões finais e perspectivas de trabalhos futuros.

De forma mais detalhada, a revisão bibliográfica do segundo capítulo inicia na seção 2.1, onde é definido o conceito de federação. No item 2.2 são apresentadas as tecnologias de integração que usaremos para classificar o FSB. A seção 2.3 faz uma revisão das arquiteturas que atenderemos com a nossa proposta. O item 2.4 explica o ESB que é a base do FSB. Na seção 0 são apresentadas as redes de Petri, uma modelagem matemática usada para simular o fluxo do FSB.

Nossa proposta, o FSB, é inicialmente explicada no capítulo 3 onde mostramos como realizar a composição de ESBs para criar o FSB. Neste capítulo também vemos as principais funcionalidades do FSB, assim como uma explicação do seu *workflow* básico modelado em UML.

O capítulo 4 retoma a explicação do fluxo do FSB em dois estudos de caso modelados e simulados com o uso de redes de Petri e BPEL. A validação de nossa

proposta também ocorre neste capítulo a partir das simulações e inferência de comportamento.

Finalmente no capitulo 5 apresentamos as conclusões finais onde são mostradas as principais contribuições e classificando o FSB segundo critérios definidos no segundo capítulo. Também são comentadas as limitações da proposta para que sejam listados alguns trabalhos futuros.

2 Conceitos Básicos

Este capítulo apresenta os principais conceitos usados na construção da proposta. Iniciamos na seção 2.1 apresentando o conceito de Federação ao abordar tanto o problema clássico, como o de autenticação federada, como outros que tratamos em nossa proposta, como o mapeamento de entidades.

As tecnologias de integração são abordadas na seção 2.2, mostrando um pouco dos modelos existentes e tipos de midlewares de integração para que possamos posteriormente enquadrar nossa proposta nas categorias descritas. O FSB também segue algumas arquiteturas de sistemas distribuídos explicadas na seção 2.3. Como base de nossa proposta, estudaremos o Enterprise Service Bus (ESB) na seção 2.4

2.1 Federação

Federação é uma solução para realizar o *login* único (*single sign-on*) que também traz a delegação de responsabilidades honrada pelas relações de confiança entre as partes federadas [Chong 2004]. Federações são estabelecidas quando sistemas distintos, comumente de empresas diferentes, necessitam cooperar para realizar uma tarefa. A grande maioria dos trabalhos encontrados na literatura trata somente aspectos de segurança e autenticação de federações, alguns dos quais são vistos neste capítulo. Entretanto, nossa proposta segue outra direção ainda pouco explorada, abordando funcionalidades tais como a correlação de entidades e mapeamento de atributos.

A Federação é uma coleção de domínios (vide Figura 2.1), cada um regido pelas políticas de seu próprio domínio, e.g. políticas de segurança e administração, entre outras. Essas políticas se tornam importantes em uma federação pela necessidade de negociação de diferenças através de domínios no processo de estabelecimento de uma federação. A negociação se torna importante entre objetos da federação no momento em que os objetos entram em uma federação.

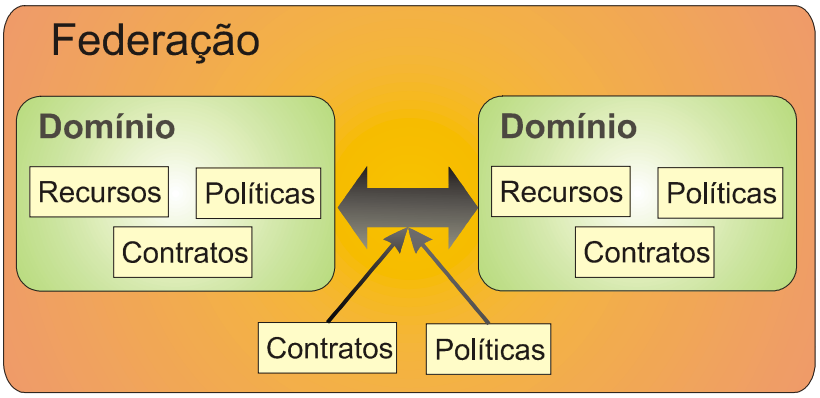


Figura 2.1: Federação de sistemas

Cada domínio possui seus próprios contratos, que são interfaces de serviços com informações de pré-condições e pós-condições de execução. O compartilhamento de recursos que um domínio oferece aos outros domínios também é importante. A habilidade de entrar ou sair de uma federação, assim como mudar a natureza do compartilhamento, faz parte da especificação de autonomia para cada domínio, associado a um contrato específico do domínio com a federação. Esses comportamentos e ações são adquiridos em termos de comunidade (a federação) e associados a um

ambiente de contratos, um para cada domínio. A própria federação é definida por contratos e está submissa pelas políticas da federação. [Putman 2001].

Normalmente, a federação de sistemas pode ser feita de duas formas: a partir dos serviços dos sistemas ou a partir dos dados [Ashford 2004]. A federação de serviços indica que os protocolos de comunicação entre os serviços sejam similares. A federação de dados fornece mecanismos pelos quais os dados subjacentes (ambos os sistema e usuário) podem ser compartilhados. O NGOSS reconhece ambas as formas de maneira que a federação seja transparentemente.

2.1.1 Funcionalidades da Federação

O estudo de federação atualmente está centrado principalmente no gerenciamento de identidades e aspectos de segurança, embora não seja o foco deste trabalho. O *Liberty Aliance Project* [Liberty 2006], por exemplo, é apoiado por grandes empresas, como IBM e Sun, com o objetivo de construir uma especificação aberta para federação de identidades e *web services* baseados em identidades. Contudo, existem outras funcionalidades importantes [Chong 2004], embora não obrigatórias, na integração federativa de sistemas, as quais são descritas a seguir.

Na funcionalidade conhecida como ***sign-on*** **federado**, a responsabilidade da autenticação é delegada a alguém confiável. Para isso, deve haver uma infra-estrutura de autenticação que compreendam as relações de confiança e inter-operem através de protocolos padrões como, por exemplo, o protocolo Kerberos [Camargo 2007].

Outra funcionalidade federativa é a **correlação de identidade**, a qual representa uma identificação pessoal de uma entidade, sistêmica ou humana, que em um contexto é

identificada através de um atributo único chamado identificador. Em um "ecossistema computacional", podem existir inúmeros contextos com a identificação de uma mesma entidade existente em mais de um contexto, mesmo possuindo identificadores diferentes. A correlação de identidades tem uma grande contribuição ao recuperar credenciais de uma entidade na interação entre contexto.

. O problema da **correlação de entidade** é semelhante ao da correlação de identidade. Em um sistema de *tarifação*, uma fatura deve ser reconhecida como a mesma nos contextos de faturamento e cobrança, embora que em cada um desses contextos a fatura possa ter atributos específicos.

O **mapeamento de atributos** é outro problema, pois estes são persistidos em várias bases podem ter a mesma semântica, mas possuir representações sintáticas iguais ou diferentes. Um exemplo simples é quando campos diferentes possuem a mesma informação como nome que pode ser chamado de `nome` ou `nomeUsuario`. Um caso mais complexo é a idade que em alguns sistemas podemos ter um campo idade e em outro somente a data de nascimento. Neste caso, só é possível a transformação do atributo em um sentido.

A proteção de entidades e atributos específicos de contexto é outro ponto importante. Cada contexto possui no seu domínio um conjunto de entidades e atributos específicos que podem necessitar de algum grau de proteção. Um exemplo é a entidade "cartão de crédito" que possui como atributos o nome do portador, número, bandeira e dígito de segurança. Essas informações só devem ser acessadas por contextos específicos como no cadastro dessa informação e pagamento.

Gerenciamento de acesso é o processo de controlar e conceder o acesso as solicitações dos recursos. A conclusão desse processo se dá por meio das ações de autenticação, autorização e auditoria. A autenticação ocorre quando a identidade é comprovada. A autorização é o processo de verificar as permissões para permitir ou não a execução de ações. A auditoria é o processo de contabilização do registro de eventos de segurança ocorridos.

Propagação de identidade ocorre durante a troca de mensagens entre serviços, uma identidade deverá ser propagada junto com o contexto, de forma que seja possível verificar suas credenciais. A identidade passada pode representar um usuário, um pseudônimo ou uma entidade sistêmica.

Um requisito comum em federação é a **auditoria**. Neste processo, eventos poderão ser auditados a partir de seus *logs* de forma eficiente, disponível, precisa e não repudiada. Em auditorias multi-site, deverá ser possível a identificação das identidades originais envolvidas com o evento, respeitando as políticas de privacidade.

Em sistemas onde o anonimato do usuário é um requisito, o **suporte a serviços de pseudônimos** pode ser uma alternativa de extrema relevância. Utilizando esse tipo de serviço, ao realizar a troca de mensagens, a identidade passada entre os domínios é um pseudônimo, isto é, não é passada a verdadeira identidade.

Embora o uso de pseudônimos garanta o sigilo da identidade, este também oferece mecanismos para correlação de identidade. Na prática, isso significa que embora um serviço não saiba quem realmente está usando o serviço, dependendo das políticas

aplicadas, é capaz de identificar que um mesmo usuário usou determinados pseudônimos.

A auditoria também é possível de ser feita sem perdas de informação. Pode ser feito um rastreamento de todas as operações feitas por um ator mesmo entre domínios diferentes através da correlação entre a identidade original e seus pseudônimos.

Na subseção 2, serão abordadas características e principais categorias das tecnologias de integração, requisitos para o entendimento da nossa proposta.

2.2 Tecnologias de Integração

As tecnologias de integração têm como base mecanismos conhecidos como *middleware*, os quais possibilitam a comunicação entre entidades (bancos de dados ou aplicações) [Linthicum 2000]. Basicamente, a função do *midleware* é fazer a supressão de diferenças entre as partes comunicantes de forma a facilitar ou viabilizar a troca de informações.

As diferentes características das tecnologias de integração possuem vantagens e desvantagens que devem ser levadas em consideração para a escolha mais adequada em cada cenário. Essas características recaem em dois modelos básicos: o lógico e o físico [Linthicum 2000].

2.2.1Modelo lógico

O modelo lógico define características de configuração, que pode ser um-para-um (*one-to-one*) ou muitos-para-muitos (*many-to-many*), e funcionamento, que pode ser síncrono e assíncrono.

A configuração um-para-um ou ponto-a-ponto (P2P) possui um canal único e direto onde duas aplicações são isoladamente integradas. Essa configuração não possui uma camada própria para abrigar a lógica, pois está limitada a integração de duas aplicações. Já a configuração de muitos-para-muitos é capaz de oferecer integração entre várias aplicações.

No funcionamento assíncrono, o processamento das aplicações integradas ao *midleware* ocorre sem bloqueio, ou seja, as requisições de uma aplicação são atendidas sem que a aplicação requisitante seja bloqueada. Este último continua seu fluxo mesmo sem a resposta da requisição enviada. No funcionamento síncrono, existe a dependência da resposta para que o processo continue normalmente, havendo um tempo limite para a execução de uma requisição.

2.2.2Modelo físico

O modelo físico trata as diversas formas com que o modelo de mensagens pode ocorrer:

- Com ou sem conexão: para a troca de mensagens, pode ser necessário ou não o estabelecimento de uma "sessão de comunicação", desta forma as aplicações estariam ligadas de forma conectada ou desconectadas;

- Comunicação direta: a mensagem é passada sincronamente e diretamente entre o *middleware* e as aplicações envolvidas;
- Comunicação intermediada por fila de mensagens: mensagens são depositadas e requisitadas pelas aplicações em um gerenciador de filas de forma assíncrona e independente;
- Comunicação por publicação e assinatura: a comunicação entre as aplicações é intermediada indiretamente, de forma que os participantes só precisam se conhecer o *middleware*. As aplicações interessadas em receber mensagens se inscrevem ou assinam uma lista de entrega. Desta forma aplicações que têm informações para publicar podem somente disponibilizar suas mensagens no *middleware* que este se encarregará da entrega;
- Comunicação por requisição e resposta: normalmente utiliza o modelo síncrono, mas também é possível usar a forma assíncrona;
- Comunicação negociada: o midleware controla o estado das regras de negócio de forma a negociar transações dependentes de estado com as aplicações.

2.2.3Tipos de *middleware*

As tecnologias de middleware orientadas a dados utilizam primordialmente uma única interface abstrata que permite acesso a diversos tipo de banco de dados [Juric et al 2001]. Esse middleware permite a uma aplicação acessar diversos tipos de banco de dados ao mesmo tempo com uma única interface, que torna fácil a integração dos dados [Linthicum 2000].

Middlewares orientados a mensagens (*Message Oriented Midleware* – MOM) é uma tecnologia apropriada para comunicação orientada a eventos. Oferece API´s padrões para seu acesso que tornam transparentes a diferenças entre as aplicações, plataformas, e protocolos envolvidos [Linthicum 2000]. Baseia-se na recepção, armazenamento e envio assíncrono de mensagens que suportam falhas. Os MOMs não são uma solução completa de integração, sendo usados para transportar mensagens para outros intermediários de mensagens que realizam funções complementares, como ocorre no *Enterprise Service Bus* (vide seção 2.4).

As chamadas a procedimentos remotos (*Remote Procedure Call* – RPC) são outro tipo de *middleware*, de funcionamento síncrono do tipo cliente/servidor, iniciado pelo cliente, o qual realiza uma chamada ou requisição (*request*) permanecendo bloqueado aguardando uma resposta (*response* ou *reply*) do servidor. Este mecanismo apresenta limitações de *performance* tanto do cliente, que fica bloqueado aguardando a resposta, quanto da rede, cujo volume de comunicação realizado pode afetar o desempenho [Linthicum 2000]. Também há problemas caso ocorra a interrupção do processamento por qualquer tipo de falha.

2.2.4Objetos Distribuídos

Os objetos distribuídos (*Object Request Brokers* – ORBs) são uma tecnologia que gerencia e suporta a comunicação entre componentes e objetos distribuídos [Juric et al 2001], sendo normalmente realizada de forma síncrona. Os ORBs funcionam como uma camada que escondem a complexidade de plataformas e ambientes distribuídos de forma a se ter a percepção de estar trabalhando com componentes locais.

2.2.5Monitores de Processamento de Transações

Os monitores de processamento de transações (*Transaction Processing Monitor* – TPM) são baseados no conceito de transações. Eles fazem o gerenciamento de *performance* e controle de segurança, sendo tolerantes a falhas. Este *middleware* faz o intermédio entre aplicações e banco de dados, podendo inclusive diminuir o custo com licenças necessárias para acesso ao banco. Fazendo parte de uma tecnologia mais antiga que precedeu aos servidores de aplicação, os TPMs laçam mão de s RPCs para realizar a comunicação entre as aplicações [Juric et al 2001]. Normalmente não suportam processamento orientado a eventos [Linthicum 2000]. De forma combinada, os TPM podem se utilizar de outros tipos de *middleware* além dos RPCs, como, por exemplo, os MOMs.

2.2.6Servidores de Aplicação

Os servidores de aplicação (*Application Servers* – AS) completam o conjunto de *middlewares*, formando um ambiente de integração com controle de distribuição de carga do processamento e utilização de recursos, alem de ser gerenciável do ponto de vista de segurança [Juric et al 2001]. Os ASs já foram criados pensando em transações baseadas na Internet. São capazes de abrigar, alem das aplicações, vários tipos de *middleware* [Linthicum 2000], possibilitando assim o suporte a vários padrões de integração, como SOAP, WSDL (*Web Service Description Language)* e J2CA (*J2EE Connector Architecture*) , e várias funcionalidades como *workflow* e BPM (*Business Process Management* – gerenciamento de processos de negócio).

A seguir, na seção 1.3, são apresentadas três arquiteturas de sistemas distribuídos que serviram como base na elaboração do FSB proposto.

2.3 Arquiteturas de Sistemas Distribuídos

2.3.1 SOA

SOA (*Service Oriented Architecture* – Arquitetura Orientada a Serviços) é uma arquitetura para sistemas distribuídos baseada em interações de serviços. SOA é para várias empresas uma mudança em seu modelo de negócio. Não é fácil conseguir uma definição específica, pois existem diferentes interpretações e técnicas envolvendo SOA [Sun 2008]. Entender SOA é entender serviço. Primeiramente, são apresentados os *web services* e a seguir os Serviços Autônomos.

2.3.1.1 *Web Services*(WS)

O uso de *web services* está fortemente associado às ferramentas do tipo SOA. Os requisitos de serviços em SOA são atendidos pelos *web services*, pois eles são autodescritos, autocontidos, modulares, publicáveis, localizáveis, dinâmicos, orquestráveis e independente de linguagem [Endrei et al 2004].

Web services vêm se destacando nos últimos tempos na interoperabilidade de sistemas heterogêneos. Segundo a W3C, "*Web Services* é uma aplicação de Software, identificada por uma URL, cujas interfaces e ligações são capazes de serem definidas, descritas e descobertas como artefatos XML. *Web services* suportam iteração direta com outros agentes de software usando troca de mensagens baseadas em XML via protocolos baseados na Internet." De maneira simplificada, podemos dizer que w*eb services* é a composição de XML com um protocolo de transporte (como HTTP) [Endrei et al 2004].

Web services também podem ser chamados de "*XML Web Services*" pois é uma tecnologia baseada no XML. A palavra "Web" tem um sentido mais comercial e não se restringe a *World Wide Web*, estendendo seu significado as várias tecnologias de transmissão de dados usadas tanto em Intranets como na Internet com HTTP, HTTPS, FTP e SMTP.

Basicamente, existem três atores em uma arquitetura de WS: o *provedor de serviços* ou simplesmente *servidor*; o *cliente*, o qual utiliza o(s) serviço(s) disponibilizados pelo *servidor* e o *registro de serviços*, que contém informações de vários serviços e *servidores*.

2.3.1.2 Operações

Existem pelo menos seis operações possíveis para que o serviço seja usado por completo. A Figura 2.2 representa a seqüência de operações entre os componentes.

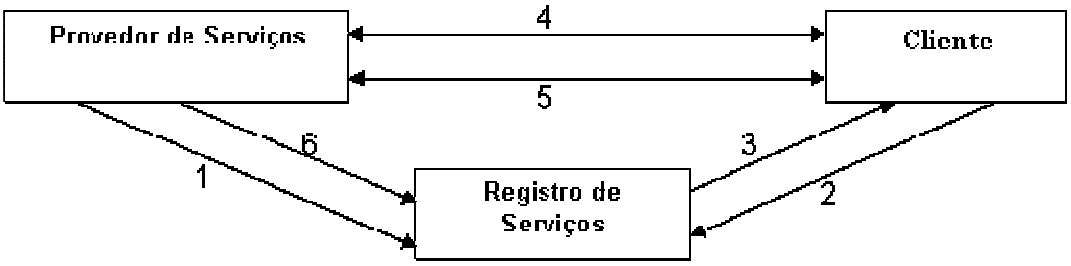


Figura 2.2: Operações via *Web Services* [Endrei et al 2004]

1. Publicar: Depois do serviço estar disponível, este é publicado no *Registro de Serviços* afim de estar disponível para pesquisa.

2. Procurar: Quando o *Cliente* precisa de um serviço, é enviado uma mensagem com os parâmetros do serviço que se deseja encontrar.

3. Achar: Uma vez recebido os parâmetros de pesquisa, é feito uma busca e o resultado é retornado ao *Cliente*.
4. Mapear Interface: Antes de usar o serviço é necessário que o *cliente* conheça as interfaces dos serviços que vai usar a fim de utiliza-los.
5. Usar: Neste momento é que o serviço é realmente usado.
6. Retirar: É quando o *Provedor de Serviço* decide retirar o registro de seu(s) serviço(s) do *Registro de Serviços*.

Estes seis passos garantem dinamismo, podendo-se gerar um cliente capaz de se comunicar com praticamente qualquer serviço sem configuração prévia.

2.3.1.3 Especificações

Não existe uma especificação única para os WSs, sendo estes compostos por uma coleção de especificações. A seguir, apresentamos algumas especificações de relevância para este trabalho.

2.3.1.4 SOAP

SOAP (***Simple Object Access Protocol***), cuja base é o XML, é o protocolo de comunicação mais utilizado em *web services* [Chappell 2002]. Originalmente, SOAP era somente um protocolo de mensagens. Desta forma um documento XML que seguisse o padrão SOAP era uma mensagem SOAP. Isso não era suficiente, pois se precisava mapear objetos e fazer RPC. Daí surgiram duas novas especificações opcionais para o protocolo: *Encode Rules* e *RPC support* [Chappell 2002]. Estas especificações opcionais descrevem como fazer o mapeamento dos objetos além de

padronizar a forma de fazê-lo RPC via SOAP. Com isso, as chamadas RPC via SOAP se tornam muito mais flexíveis. As especificações podem ser encontradas no site da W3C [W3C 2008].

Após isso, foi criada ainda uma terceira especificação. Esta, também opcional, define como colocar uma mensagem SOAP dentro de uma mensagem HTTP. Como a maioria dos serviços implementados na nossa proposta são executados sobre HTTP, colocamos todos os exemplos desta sessão seguindo esta especificação e destacando a mensagem SOAP propriamente dita. Através do uso do protocolo HTTP, podemos tirar proveito de tecnologias já projetadas para ele como *LoadBalance* e HTTPS. Na verdade, SOAP pode rodar em cima de vários protocolos, como por exemplo o SMTP (*Simple Mail Transfer Protocol)* ou sobre a pilha TCP/IP.

A Tabela 2.1 possui o resumo das principais Especificações SOAP em Inglês e uma descrição.

Tabela 2.1: Especificações SOAP [Chappell 2002]

Especificação	Descrição

Message envelope	Descreve como deve ser a mensagem.
Encode Rules	Define as regras de codificação para a serialização de tipos definido na aplicação.
RPC support	Define a construção para suportar a interação RPC entre o transmissor e o receptor da mensagem. Também são suportadas mensagens assíncronas.
HTTP binding	Especifica como realiza a ligação entre a mensagem SOAP ao protocolo HTTP de forma a se ter um pacote só.

A estrutura básica de uma mensagem SOAP, isto é, o *SOAP Envelop* é descrito na Figura 2.3.

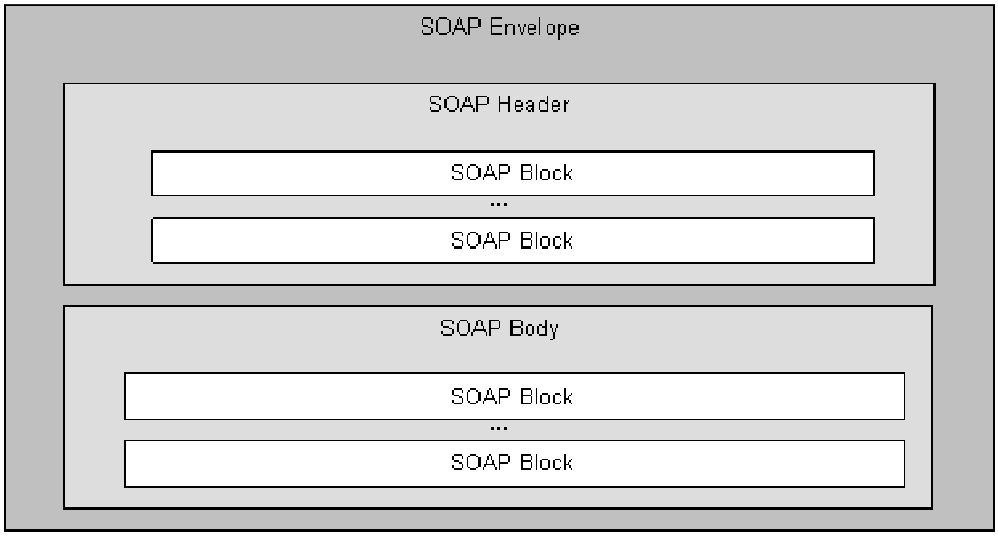


Figura 2.3: SOAP Envelop [Chappell 2002]

2.3.1.5 WSDL

A sigla WSDL representa *Web Service Description Language* que poderia ser traduzida como linguagem de descrição de *web services* [Gabrick 2002]. Como o WSDL só faz uma descrição em um arquivo externo, é possível implementar um serviço sem utilizá-

la. Um documento WSDL contém informações relativas à localização, operações disponíveis e assinaturas dos serviços.

A riqueza de detalhes é tanta que existem ferramentas que através do WSDL são capazes de gerar código de cliente e servidor (*stubs* e *skeletons*) em tempo de compilação ou execução [Apache 2008].

Existem cinco elementos principais em um documento WSDL. Estes elementos estão enumerados e brevemente descritos na Tabela 2.2.

Tabela 2.2: Descrição dos elementos do WSDL [Chappell 2002]

Elemento WSDL	Descrição
Type	Permite a inclusão de novos tipos (XML Schema) para posterior referência em outros elementos WSDL.
message	Define o conteúdo de uma mensagem SOAP nomeando cada parte da mensagem. Em uma chamada RPC, é necessário definir duas mensagens: uma para requisição, onde cada parte representa um parâmetro, e outra para a resposta onde definimos o retorno.
portType	Ele representa uma interface abstrata. Os filhos deste são elementos do tipo *operation* e representam um método abstrato.
Biding	Contém detalhes de como uma interface abstrata (*portType*) é mapeada para um mecanismo de comunicação como SOAP sobre HTTP.
Service	É uma coleção de elementos *port*. Cada elemento *port* descreve uma instância de um WS, incluindo onde ele está localizado (*URL*), qual interface ele implementa (*portType*) e como interagir com ele (*biding*).

O modelo de requisição e resposta é flexível, podendo ser classificado em quatro tipos:

- *Request-Response* – o cliente envia uma requisição ao servidor e espera uma resposta;
- *One-Way* – o cliente envia uma mensagem ao servidor sem uma resposta;

- *Notification* – O servidor envia uma mensagem ao cliente;
- *Solicit-Response* - O servidor envia uma mensagem ao cliente e aguarda um retorno do cliente.

Existem muitas formas de se construir um WS e, para cada uma delas, a linguagem WSDL deve conseguir descrevê-la.

2.3.1.6 UDDI

Os protocolos anteriores se referem a transporte e descrição de serviços, que em alguns casos é mais do que suficiente para se implementar um sistema, mas para sistemas nos quais se busca independência de localização, por exemplo, é necessário um serviço de localização. O UDDI (*Universal Discovery and Description Integration*) é um serviço bem completo de localização de serviços [uddi.org].

O UDDI nasceu de uma iniciativa entre Ariba, IBM e Microsoft que durou meses. Oficialmente, nasceu em 6 de Setembro de 2000. Em Maio de 2001, Microsoft e a IBM lançaram, o primeiro site operador de UDDI, o *UDDI Registry*. Em junho de 2001, foi anunciada a versão 2.0 de UDDI. O apoio, que era inicialmente de três empresas, cresceu e já chega a mais de 310 empresas no atual projeto.

Conceitualmente, as informações armazenadas em um registro UDDI estão divididas em três categorias [Chappell 2002]:

- Páginas Brancas: contêm informações gerais de uma empresa específica como nome e descrição de um negócio, informações de contato, telefone, fax. Também podem incluir identificadores de negócios (*business identifiers*), no formato de

classificações *Dun & Bradstreet's D-U-N-SÒ* (*Data Universal Numbering System*), representados por números de nove dígitos atribuídos a negócios. Na versão 2.0, o UDDI oferece suporte para identificadores específicos de indústrias, tal como o sistema do *Standard Industrial Classification* (SIC), o qual atribui um identificador único a uma determinada indústria (e.g.7371 representa serviços de programação de computadores);

- Páginas Amarelas: contêm informações de dados de classificação geral independente de empresa ou serviço oferecido. Por exemplo, esses dados podem incluir a indústria, o produto, ou códigos geográficos baseados sobre padrões de taxonomias;
- Páginas Verdes: guardam as informações mais técnicas de como invocar os serviços oferecidos. Normalmente o descritor do serviço se encontra externo e existe somente um ponteiro. Embora alguns pensem que os serviços apontados sejam somente WS, é possível localizar outros tipos de serviços como CORBA, Java RMI, EJB, ou mesmo uma página *web* ou e-mail.

A diversidade de informações disponibilizadas pelo UDDI facilita a localização de um serviço. Em muitos casos não é necessário algo tão sofisticado. Talvez nestes casos, um simples repositório compartilhado de WSDL seja suficiente. Mas se o intuito é lançar um serviço genérico e publicá-lo para a Internet, o UDDI é uma opção. Também existem outros protocolos com a mesma finalidade como é o caso do ebXML.

2.3.1.7 WS-Coordination e WS-Transaction

Quando um grupo de WS interage coletivamente para executar uma unidade de uma tarefa de negócio, é necessário que esses serviços compartilhem um contexto comum. O contexto define tanto a existência da execução da atividade, como estabelece um nível de controle sobre como a tarefa está sendo executada e pode ser processada.

Devido a uma especificação de acoplamento fraco, manter o contexto de uma transação em WS não é algo tão trivial quanto em outras tecnologias. Outras tecnologias usam uma conexão de rede, por exemplo, para manter um contexto de transação. Caso a conexão seja desfeita, a transação é abortada. Com WS não existe garantia de que será utilizada uma única conexão necessitando de especificações mais completas para manter persistente o contexto de uma atividade. Por essa razão, *WS-Coordination* e *WS-Transaction* foram criados de forma a prover um framework de gerenciamento de contexto [Erl 2004].

A especificação do *WS-Coordination* provê mecanismos padrões para que serviços possam distribuir e registrar as definições de contexto que representam as atividades fazendo uso de informações adicionadas ao cabeçalho das mensagens SOAP. O modelo de serviço de coordenação consiste de serviços individuais que provêem operações dedicadas para as funções de criação do contexto, registro para o contexto de coordenação e escolha do protocolo.

Esses serviços são chamados coletivamente de serviços de coordenação ou simplesmente *coordenador*. Outras especificações de WS podem implementar este *framework* provendo uma definição de tipo de coordenação. A especificação de *WS-*

Transaction define dois tipos de coordenação: *WS-AtomicTransacion e WS-BusinessActivity.*

WS-AtomicTransacion foi criada para transações semelhantes a dos bancos de dados, com as mesmas características básicas: atomicidade, consistência, isolação e durabilidade. É recomendado o uso do *WS-AtomicTransacion* somente em transações curtas, pois durante seu fluxo são feitos bloqueios de recursos e mantidos até o término da transação. Em caso de falha, os valores bloqueados retornam ao seu estado inicial e são liberados da mesma forma que um comando `rollback` em um gerenciador de banco de dados.

WS-BusinessActivity foi desenvolvida para dar suporte a transações longas, que podem até durar dias. Não se utiliza bloqueios pois poderia tornar o recurso indisponível por muito tempo. Em caso de falhas, são realizadas ações compensatórias. A *WS-BusinessActivity* provê dois tipos de protocolos de coordenação, o *BusinessAgreement* e *BusinessAgreementWithComplete*, permitindo aos serviços determinarem se serão notificados ou não do término da transação.

2.3.1.8 WS-BPEL

Em sistemas um pouco mais complexos, uma operação lógica pode envolver mais de um serviço. Pensando em definir o fluxo de invocações de serviços para um determinado fim, foram e estão sendo criados novos protocolos.

A IBM criou inicialmente o protocolo WSFL (*Web Service Flow Language*) em maio de 2001. A Microsoft também desenvolveu sua solução, a XLang. Ao invés da Microsoft e a IBM disputarem o mercado com seus protocolos, elas se uniram e da

união das idéias do WSFL e XLang, nasceu o BPEL*4*WS (*Business Process Execution Language for Web Services*) também conhecido como WS-BPEL [IBM 2008]. O mercado ainda não teve tempo de decidir qual padrão adotar.

WS-BPEL está intimamente relacionada com outras especificações de WS, por exemplo, a interface de serviço suportada pelo WS-BPEL é o WSDL. Da mesma forma, o suporte a transações é feito através das definições de *WS-Transaction* e *WS-Coodination*.

É possível tratar exceções indo para um fluxo alternativo ou através do suporte a ações compensatórias. Para aplicações B2B, ele também dá suporte a Acordos. Acordos representam à capacidade de estabelecer contratos entre dois ou mais participantes para realizar funções especificas.

Embora suporte modelagem baseada em colaboração, ele não suporta o uso direto de UML. A falta do suporte direto ao UML não é tão grave devido ao fato de que se pode estender o modelo.

Uma vantagem para a comunidade Java é que a IBM disponibilizou o BPEL*4*J (*Business Processes in Web Services for Java*) gratuitamente em seu site [IBM 2008]. O BPEL*4*J é composto por uma plataforma que executa processos definidos em BPEL4WS, uma ferramenta para validação de documentos BPEL4WS, e um *plug-in* do o *Eclipse* (ambiente de desenvolvimento Java da IBM) para alterar integrado com a ferramenta de validação de documentos BPEL4WS.

Atualmente, o BPEL faz parte de JBI (*Java Business Integration*), definido pela JSR-208 [JCP 2008]. Esta especificação já foi incorporada por alguns servidores de

aplicação como o Apache Geronimo [Geronimo 2008] e o GlassFish [GlassFish] da Sun.

2.3.1.9 Serviços autônomos

Serviços autônomos [Dahan 2006] podem ser vistos como serviços com total autonomia para executar suas funções, isto é, não dependem de nenhum outro serviço. Essa abordagem, como no EDA, é composta por serviços que possuem todas as informações necessárias para sua execução, canal de comunicação com capacidade de transmissão em broadcast, faz uso de mensagens assíncronas, entretanto também faz uso de mensagens síncronas, que não é utilizado no EDA.

O funcionamento dos serviços autônomos se dá de duas formas: chamadas ao serviço e atualização de informações do serviço. As chamadas ao serviço podem ser feitas tanto de forma síncrona (chamadas RPC) como assíncrona. Como o serviço é autônomo, este não precisa realizar chamada a nenhum outro serviço, pois já tem um repositório com todas as informações que o interessam para processar a mensagem recebida.

As chamadas de atualização de informações do serviço são realizadas através de mensagens assíncronas tendo como objetivo atualizar as informações do repositório do serviço que foram alteradas ou geradas por outros serviços. Com esse mecanismo, as chamadas ao serviço podem ocorrer sem a necessidade de consultar informações em outros serviços, mas adiciona a responsabilidade de atualizar o repositório de outros serviços com suas próprias informações.

2.3.2 Arquitetura NGOSS TNA

O NGOSS *(Next Generation Operation System and Software*) se concentra em definir uma arquitetura que automatiza o processo de negócio dos OSS (*Operational Support Systems* – Sistemas de Suporte Operacional). OSSs são usados pelas provedoras de serviços de telecomunicações (*Telecom Service Provider* - TSP) para gerenciar seus negócios e suas redes. As TSP são normalmente empresas de grande porte, com vários departamentos.

Dentre as especificações do NGOSS, destacamos o NGOSS TNA (*Technology Neutral Architecture*), no qual é definida uma arquitetura neutra a tecnologia. A partir dessa arquitetura neutra, é possível especificar inúmeras arquiteturas específicas a uma tecnologia de forma a se obter todos os benefícios do NGOSS TNA. Em nossa proposta, esta arquitetura é uma das bases arquiteturais e portanto estaremos atendendo seus pontos no FSB.

O NGOSS TNA, assim como toda a especificação NGOSS, se mostra consistente com os princípios do RM-ODP (*Reference Model for Open Distributed Processing*) [TMF 2006]. As próximas sessões apresentaram os principais elementos arquiteturais do NGOSS [Strassner 2004], sendo divididas em quatro categorias [TMF 2008].

Os tópicos da arquitetura neutra a tecnologia do NGOSS são apresentados na Figura 2.4 e detalhados no Apêndice 1.

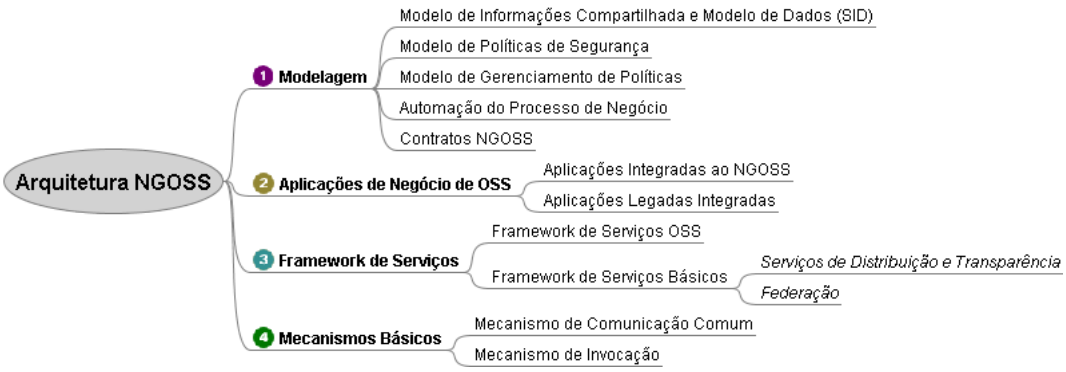


Figura 2.4: Visão geral do NGOSS TNA

2.3.3 Arquitetura Dirigida a Eventos

A inspiração de EDA (*Event-Driven Architecture* – Arquitetura Dirigida a Eventos) é que no mundo real estamos sempre cercados de eventos que nos atingem e provocam alguma reação. Essa reação pode inclusive provocar um novo evento. Por exemplo, quando dirigimos, a falta de gasolina gera um evento que faz acender a luz da reserva de combustível. Recebemos esse evento disparando uma nova ação que é a de ir a um posto de gasolina abastecer o carro a partir de novas ações e eventos.

EDA [Hohpe 2006], assim como SOA, faz uso de serviços como forma de desenvolvimento, tendo os *Web Services* como uma referência. SOA faz uso tanto de RPC e mensagens assíncronas, entretanto EDA se utiliza somente de mensagens assíncronas para propagação de eventos. Com essa abordagem, serviços ficam aguardando eventos que o interessem para processá-lo e, se for o caso, gerar novos eventos. Os eventos são propagados em um canal de broadcast de forma que um evento possa disparar mais de um serviço ao mesmo tempo.

O uso de EDA traz como vantagem um grande paralelismo ao mesmo tempo em que rompe com o paradigma de RPC. O foco agora não é a pilha de chamadas a métodos (*Call Stack*) e sim a interação entre componentes. Também há uma mudança de responsabilidade, pois, no RPC, o processo antecessor deve invocar o próximo processo. No EDA, o processo antecessor somente envia um evento e os próximos processos é que devem ter a responsabilidade de capturá-los e processá-los. Tanto EDA como RPC possui suas vantagens e desvantagens, mas deixaremos para outro trabalho um detalhamento maior.

2.4 Barramento de Serviços Corporativos

Nesta subseção estudaremos o ESB que é usado como base para a criação de nossa proposta. Por definição, o ESB é um barramento para mensagens de serviços usados para interconectar diversas aplicações [Chappel 2004]. Além da função de barramento, o ESB tem a capacidade de resolver alguns problemas de integração. Pode-se afirmar que o ESB provê SOA dirigida a eventos de forma altamente distribuída e servirá como base de nossa proposta, o FSB.

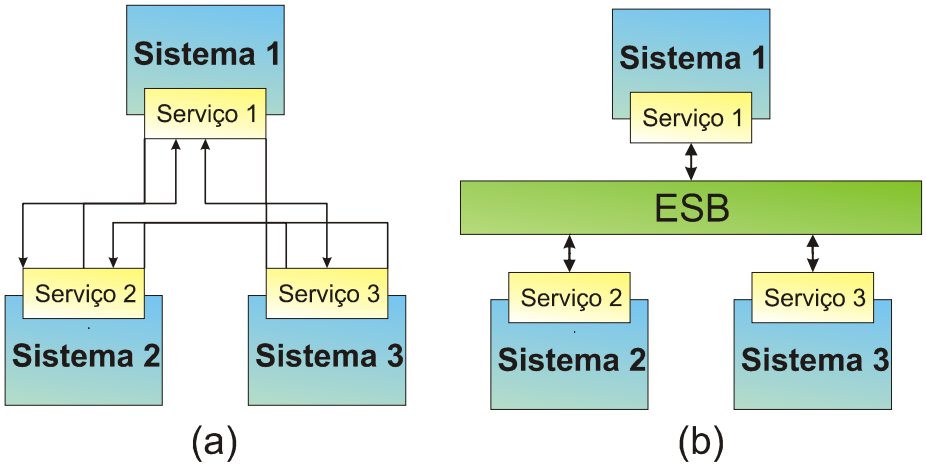


Figura 2.5: Integração sem ESB (a) e com ESB (b) [Chappel 2004]

De forma geral, sem o ESB, cada sistema teria canais de comunicação distintos entre si, como mostra a Figura 2.5(a), podendo se tornar mais complexo se cada um destes usar tecnologias diferentes como *Web Service* e Corba. Se pensarmos que cada sistema se relaciona com todos os demais por um "canal de integração", teríamos $\frac{n!}{(n-2)!2}$ canais de integração, tal que n é o número de sistemas. Adicionando o ESB, como ilustrado na Figura 2.5(b), o barramento provê a integração entre as aplicações, oferecendo o devido suporte à comunicação. Dessa forma, ESB define uma plataforma padronizada de integração, que combina serviços web, roteador de mensagens, interface com fontes de dados, fluxo de processos e acesso a funções de sistemas legados.

ESB foi projetado para usar o melhor de tecnologias anteriores. Pode-se ver na Figura 2.6 uma comparação entre algumas abordagens de integração. Pode-se ainda verificar que o uso de ESB abstrai a lógica de integração da aplicação, ao mesmo tempo em que pode atuar de forma distribuída [Chappel 2004].

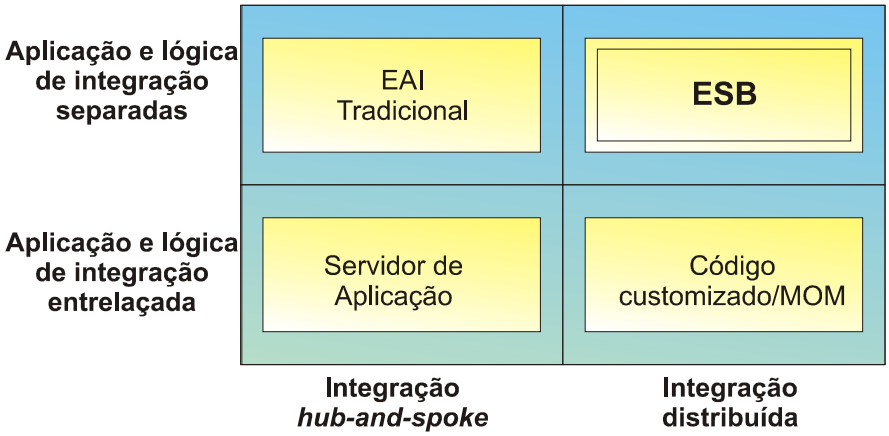


Figura 2.6: Abordagens para Integração [Chappel 2004]

É possível encontrar especificações e implementações para ESB. No caso de especificações, destacamos o *Java Business Integration* (JBI), ou Integração de Negócios Java, definido pela JSR-208 [JCP 2008]. Esta, embora esteja baseada em mensagens, permite que a aplicação use a estratégia de integração que lhe for mais conveniente, como registro em arquivos, uso de banco de dados ou de *web services*. A integração é realizada por componentes que se utilizam dessas estratégias transformando as informações em mensagens roteáveis que seguem um formato padronizado.

É possível encontrar divergências nas funcionalidades que devem ser oferecidas pelo ESB [Richards 2006], mas entre esse conjunto de operações podemos destacar: publicação, comunicação confiável, transformações e roteamento. A publicação refere-se ao mecanismo de disponibilização, atualização e remoção de serviços. A comunicação confiável garante que as mensagens chegarão ao seu destino, mesmo que os mecanismos de transmissão não sejam confiáveis. As transformações são necessárias quando as comunicações entre dois serviços exigem mensagens de formatos diferentes. O roteamento garante que, mesmo que existam várias rotas, as mensagens serão

entregues pelas rotas corretas e ao destino correto. Como existem diferenças entre outras funcionalidades que o ESB disponibiliza, considera-se aqui as disponíveis pela JBI, ilustradas pela Figura 2.7.

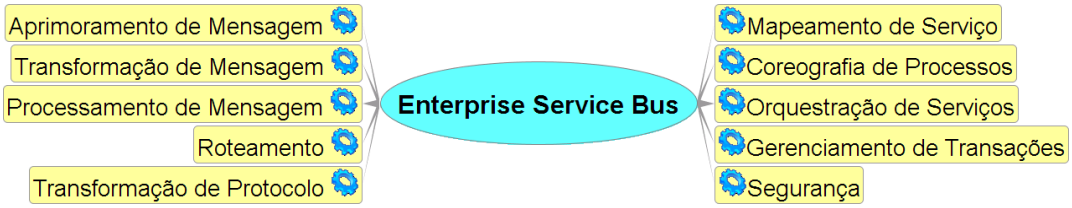


Figura 2.7: Principais competências do ESB

Embora as funcionalidades sejam inúmeras e abrangentes, especificações como JBI tornam possível criar novos componentes tão funcionais quanto os existentes. Isso é possível porque, internamente, todos os componentes estão conectados e se comunicam através de um roteador de mensagem denominado roteador de mensagem normalizada (*Normalized Message Router* – NMR). Esses componentes adicionais podem ser usados para possibilitar a comunicação do ESB com sistemas que utilizam protocolos específicos, como Corba ou SNMP, assim como para oferecer um processador de mensagens através de scripts como Groovy [ServiceMix 2006].

Embora o ESB ofereça uma solução de integração completa para ambientes corporativos, possui sérias limitações se aplicado em federações [Richards 2006]. Essas limitações serão explicadas no capítulo 3, assim como a nossa proposta que visa criar um ambiente com todos os benefícios do ESB.

3 Proposta: Barramento de Serviços Federado (FSB)

Neste capítulo, veremos as limitações do ESB em ambientes federativos assim como nossa solução para a criação do FSB. Algumas características de nossa proposta também serão vistas neste capítulo. Por fim apresentaremos uma modelagem do fluxo do FSB em UML de forma genérica para que no capítulo 4 seja possível um melhor entendimento e validação de nossa proposta a partir dos estudos de casos modelados com redes de Petri e BPEL.

3.1 O Barramento de Serviços Federado (FSB)

ESB foi projetado para ser um barramento corporativo e não é facilmente aplicável em federações, principalmente em relação à autonomia de cada sistema. No exemplo da Figura 3.1(a), existem três sistemas e um barramento. Se considerarmos que todos os sistemas estão na mesma corporação, o barramento e toda a política de integração destes sistemas serão de responsabilidade da corporação. Já em ambientes federados, pode não ser possível delegar toda a responsabilidade de tais políticas a um único domínio. Dividir a administração de um único barramento também não é uma solução aceitável, pois cada domínio possui suas próprias políticas. Assim, um domínio não pode interferir nas políticas de outro.

Manter os benefícios do ESB com a autonomia necessária na federação é o desafio da abordagem que denominamos FSB (*Federated Service Bus*). Esta proposta utiliza um ESB para cada domínio que, unidos, compõem um FSB, conforme ilustrado na Figura 3.1(b). Opcionalmente, poderão existir serviços específicos da federação para ajudar a dar suporte às funcionalidades mais específicas, como serviço de pseudônimos e transações federadas, não cobertas neste trabalho.

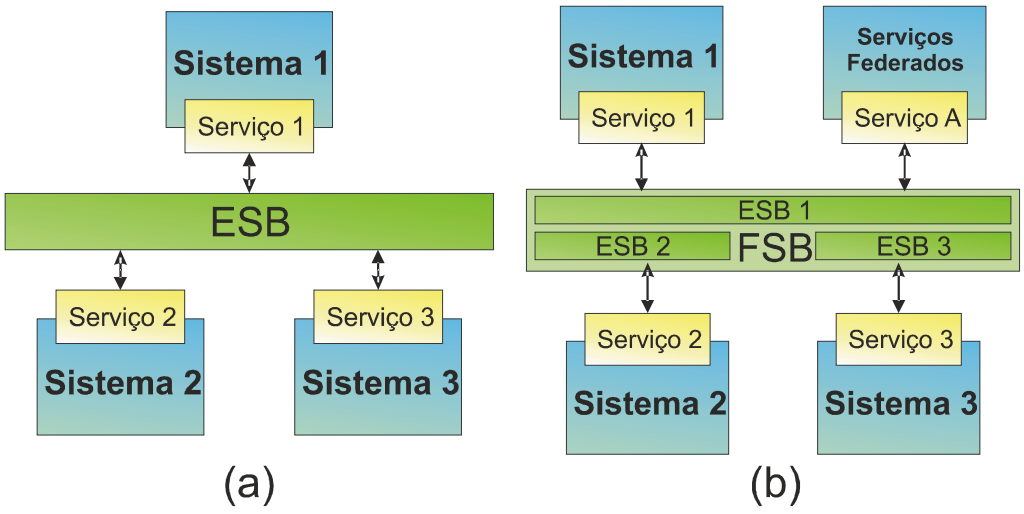


Figura 3.1: Integração com ESB (a) e FSB (b)

3.2 Funcionalidades básicas do FSB

O FSB permite que tanto sistemas que usam arquitetura orientada a serviços como outras arquiteturas possam participar de uma federação baseada na arquitetura orientada a serviços e eventos, ao assumir para si as responsabilidades necessárias para que estes sistemas necessitem do mínimo de alterações ou até mesmo nenhuma. Dentre as funcionalidades do FSB, podemos destacar três como principais: recepção e

encaminhamento de mensagens dos sistemas, aplicação de políticas, transformação de mensagens e roteamento.

A recepção e encaminhamento de mensagens dos sistemas é o ato de receber e enviar mensagens dos sistemas. Esta é uma característica herdada do ESB onde, como vimos na seção 2.4, pode-se receber ou enviar mensagens de várias fontes diferentes como *web services*, banco de dados, arquivo texto, JMS, e-mail, entre outras. Com isso, a integração do FSB com os sistemas pode usar a tecnologia mais compatível para o sistema. Esta facilidade permite que a comunicação do FSB se dê da forma mais fácil e transparente e permite que mesmo sistemas monolíticos possam enviar e receber as mensagens do FSB.

Aplicação de políticas traz o benefício de mover políticas específicas da federação para o FSB, o que torna transparente para as aplicações a execução de tais políticas. Dentre algumas políticas específicas, podemos citar o Log de mensagens. Outro exemplo seria ignorar mensagens repetidas para evitar possíveis custos, ou adiar o envio de mensagens em horários inapropriados para sua recepção, como em horários de backup do servidor.

Na transformação de mensagens, esta deve passar de um formato específico para outro comum à federação. Na arquitetura NGOSS, esse formato comum é definido no SID, sendo tão importante que possui documentação específica e abrangente. Devido à autonomia dos sistemas da federação, cada um pode ter representações diferentes para uma mesma entidade, como os alarmes de rede ou boletos que veremos nos estudos de caso. É necessária a transformação da mensagem para que sejam resolvidos alguns

problemas da federação vistos na seção 2.1, como mapeamento de atributos e correlação de identidade e entidade.

O roteamento é outro ponto forte do FSB, servindo como uma infra-estrutura para uma arquitetura orientada a eventos. O sistema que cria o evento, representado normalmente por uma mensagem, não necessita de nenhum conhecimento sobre o receptor da mensagem. O FSB tem a responsabilidade de identificar um evento e encaminhar para os receptores corretos, pois um evento pode interessar a mais de um sistema. Algumas políticas também podem ser aplicadas no momento do roteamento, como adicionar ou remover informações a mensagem, pois cada sistema pode ter necessidades ou privilégios diferenciados sobre as informações. Da mesma forma que um transmissor pode enviar uma mensagem para mais de um receptor, o contrário também é verdadeiro, podendo mais de um transmissor enviar mensagens para o mesmo receptor. Todas essas situações são abordadas no estudo de caso.

3.3 Fluxo básico do FSB

Internamente, o FSB é composto por ESBs que têm um fluxo básico de processamento da mensagem representado na Figura 3.2: Fluxo básico do FSB. Este fluxo limita o fluxo interno dos ESBs que compõem o FSB. Inicialmente neste fluxo, o emissor encaminha a mensagem a ser entregue para o FSB sendo representado pelo ESB 1, que é o ESB de envio. Por se tratar de um ESB, a recepção da mensagem, como visto na seção 2.4, pode ser feita a partir de um web service, banco de dados, fila JMS, corba, arquivo texto, entre outros. O ESB 1 possui duas etapas bem definidas: transformação e roteamento.

A etapa de transformação ocorre em dois passos, o enriquecimento e a transformação propriamente dita. O enriquecimento é o ato de adicionar informações a mensagem que serão necessárias para a transformação da mensagem. Uma vez que o enriquecimento realiza somente operações de consulta, podemos realizar em paralelo a aplicação de alguma política para reduzir o tempo em relação a um processamento seqüencial. As políticas aplicadas neste momento podem usar serviços específicos da federação, como uma autenticação federada. Um resultado excepcional da política poderia interromper o envio gerará um erro que seria retornado ao sistema. Caso contrário, a mensagem poderia ser transformada para seu formato federado utilizando as informações do resultado da política e do enriquecimento.

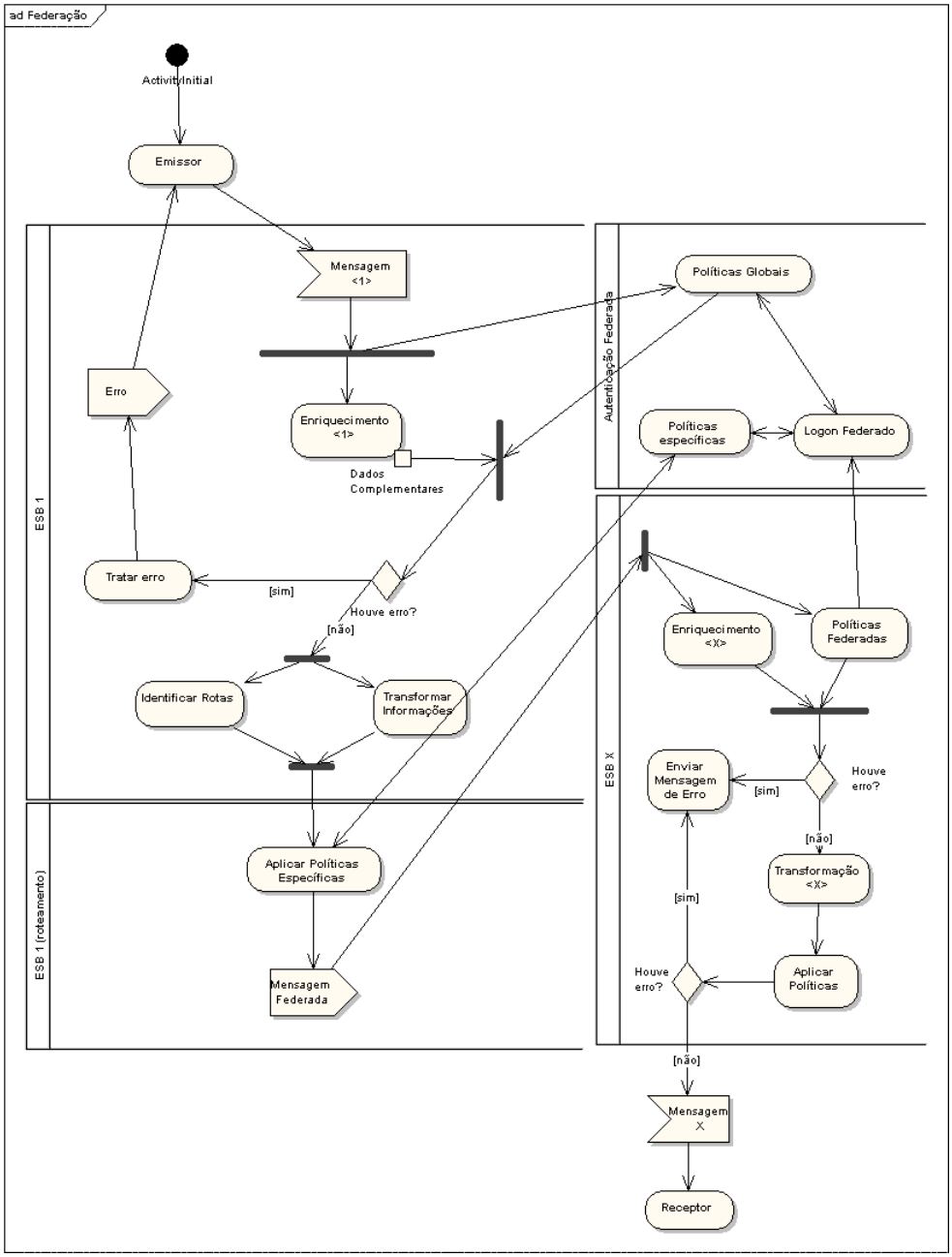


Figura 3.2: Fluxo básico do FSB

O roteamento já recebe a mensagem no formato federado e passa inicialmente pela aplicação de políticas de roteamento cuja principal função é identificar as rotas.

Nesta etapa podem ser criadas mensagens diferentes para cada rota, visto as necessidades e privilégios diferenciados sobre as informações de cada sistema receptor. Por fim, a mensagem é enviada para o ESB X, que representa o ESB do sistema receptor. A denominação X foi dada visto que pode haver mais de um ESB que recebe a mensagem.

Ao ESB X receber a mensagem, é de responsabilidade deste aplicar as políticas federadas cabíveis e transformar a mensagem para o formato específico do sistema receptor, desta forma o sistema destino pode processar a mensagem sem a necessidade de conhecer nenhum aspecto da federação. Para que isso ocorra, a mensagem que chega no ESB X segue inicialmente dois fluxos: o enriquecimento e a aplicação de políticas. O enriquecimento é a primeira fase da transformação da mensagem e, como no ESB 1, tem como objetivo coletar informações necessárias para transformar a mensagem do formato de origem, que é o formato federado, para o formato destino, no caso , formato do sistema. Podemos destacar que o ESB de recepção sempre recebe mensagens no formato global, o que facilita a lógica de transformação da mensagem. Caso o formato da mensagem fosse o mesmo do sistema de origem, a cada novo sistema, seria necessário desenvolver uma rotina específica a cada sistema para a transformação da mensagem aumentando bastante a complexidade da recepção.

A aplicação de políticas federadas pode envolver alguns aspectos de segurança como a autenticação da mensagem, mapeamento de identidade, log de mensagem para posteriormente ser usada em auditorias, entre outros.

Uma vez que à mensagem enriquecida e passada pelas políticas específicas, a mensagem é transformada para o formato específico do sistema e passa por uma última aplicação de políticas específicas do sistema receptor. Essas políticas estão mais ligadas a regras entre o sistema receptor e a federação, por exemplo, se o sistema receptor cobra uma tarifa por número de mensagens recebidas, é possível incluir um contador de mensagens neste ponto com esta finalidade.

Por fim, a mensagem é encaminhada ao destino apropriado do sistema receptor. Por se tratar de um ESB, assim como na transmissão, este destino, pode ser variado como um banco de dados, web service, fila JMS, corba, entre outros. Essa flexibilidade permite que sistemas que não foram desenvolvidos para trabalhar em uma federação possam participar de uma federação com uma arquitetura orientada a serviços e eventos com muito pouca ou nenhuma alteração.

3.4 Conclusão

Vimos nesse capítulo uma explicação sobre algumas funcionalidades do FSB, assim como o seu fluxo básico e genérico modelado em UML. No próximo capítulo, relativo à modelagem e simulações para validação da proposta, serão descritas duas provas de conceito modeladas e simuladas com redes de Petri coloridas, de forma que uma delas também foi implementada como *web services* e BPEL. Alguns pontos da modelagem e *workflow* explicados neste capítulo serão retomados e melhor explicados junto à modelagem.

4 Modelagem e Simulações

A fim de comprovar a eficácia do fluxo descrito no capítulo anterior, desenvolvemos dois estudos de caso e seus respectivos cenários fictícios: sistema de *tarifação* e sistema concentrador de alarmes. Internamente, essas mensagens seguem um fluxo que pode ser diferente de acordo com o seu conteúdo, o que facilita uma modelagem e validação baseada em redes de Petri coloridas (CPNs), sendo estas usadas para modelar e simular o fluxo interno dos estudos de caso com o auxílio da ferramenta CPN Tools [CPN 2006]. Cada um dos cenários apresentados demonstra algumas funcionalidades do FSB, visto que criar um único cenário que contém todas as funcionalidades seria muito complexo e confuso.

4.1 Estudos de Caso #1: Sistema de tarifação

Uma empresa de telecomunicações conta com o suporte de três diferentes sistemas para receber pelos serviços prestados. O primeiro sistema é o de faturamento, onde são gerados os boletos de pagamento e enviados aos clientes. O segundo sistema é o de arrecadação, responsável pelo recolhimento do dinheiro pago pelos boletos. Por fim, um sistema de cobrança é responsável por cobrar as dívidas dos clientes inadimplentes.

O fluxo de pagamento de um boleto no sistema de arrecadação deve notificar o pagamento ao sistema de cobrança, para que sejam removidas eventuais ações contra o cliente, e, em seguida, o sistema de faturamento, para impedir a impressão da segunda via de uma fatura já paga. Para a situação acima descrita, pode-se pensar em um ESB

como uma solução viável. Entretanto, supondo que a empresa de telecomunicações terceirize os setores de cada sistema para uma empresa diferente, é provável que ela não queira arcar com a responsabilidade de manter o ESB e, neste caso, não se configura um cenário corporativo, mas sim, um cenário federado, onde o uso do FSB é mais apropriado.

4.1.1 Modelagem e simulação baseadas em CPNs

A modelagem inicia na rede *Principal*, onde algumas transições representam as sub-redes que formam a hierarquia mostrada na Figura 4.1. Observa-se que a primeira sub-rede, *FSB*, detalha o FSB na rede principal. Seguindo, as sub-redes *ESB_Arr*, *ESB_Cob* e *ESB_Fat* representam, respectivamente, o detalhamento dos ESBs dos sistemas de Arrecadação, Cobrança e Faturamento. Algumas dessas redes serão exibidas e detalhadas ainda nesta seção.

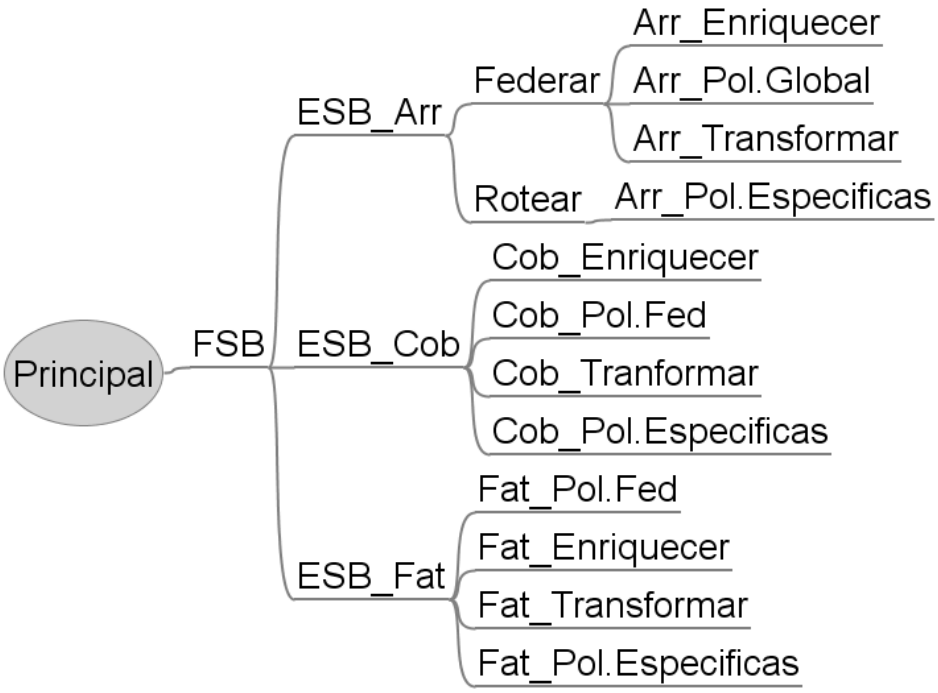


Figura 4.1. Hierarquia da rede de Petri do sistema de tarifação

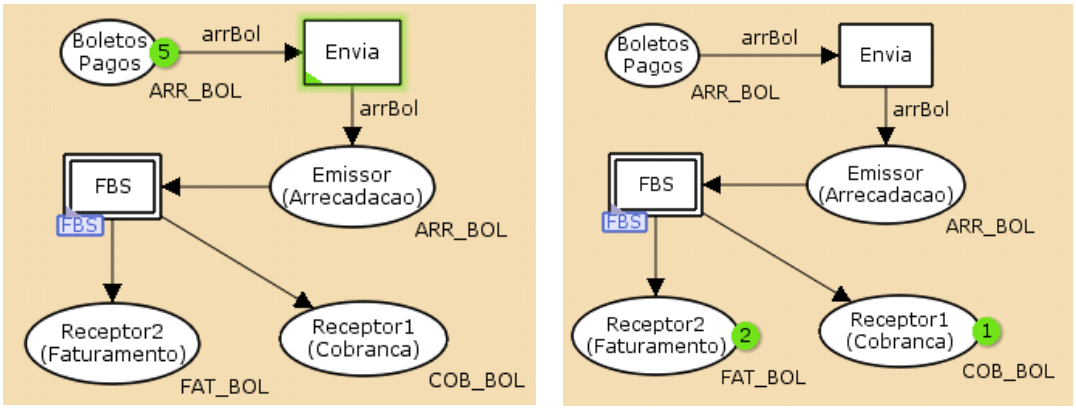
Na rede apresentada na Figura 4.2(a), os boletos pagos são enviados para o sistema de Arrecadação que os repassam para o FSB. Esse, por sua vez, tem a responsabilidade de processar a mensagem e transmiti-la para os sistemas de Cobrança e Faturamento. Pode-se supor que cada sistema possui sua própria representação de boleto, sendo, neste caso, tarefa do FSB realizar a conversão de tipos. A simulação aqui apresentada conta com cinco fichas de boleto, representadas na Tabela 4.1. Esses boletos são compostos pelo seguinte conjunto de informações: *id*, *cli*, *st*, *v*, *dvc*, *dpg* e *fpg*, que representam, respectivamente, o identificador do boleto, identificador do cliente, estado do boleto, valor do boleto, data do vencimento, data do pagamento e forma do pagamento. Esse tipo de boleto é específico do sistema de Arrecadação.

Existem outras três representações de boletos, uma para o sistema de Cobrança, outra para o sistema de Faturamento e outra representação genérica, essencial para padronizar a comunicação entre os três ESBs.

Tabela 4.1. Marcação inicial dos boletos

```
1`{id=0, cli=1822, st="gerada", v=834,  dvc="5708", dpg="5189", fpg="1"}++
1`{id=1, cli=4910, st="emNeg",  v=1429, dvc="2651", dpg="",     fpg="2"}++
1`{id=2, cli=4910, st="emNeg",  v=9587, dvc="4055", dpg="",     fpg="4"}++
1`{id=3, cli=4910, st="gerada", v=4107, dvc="8103", dpg="7779", fpg="0"}++
1`{id=4, cli=1822, st="emNeg",  v=7515, dvc="486",  dpg="",     fpg="1"}
```

Executando a rede da Figura 4.2(a) até seu estado final, representado na Figura 4.2(b), verifica-se a existência de apenas duas fichas no sistema de Faturamento e uma no sistema de Cobrança. Isso se deve à lógica de integração do FSB, onde os boletos com estado “emNeg” (em negociação) não podem ser pagos. Dessa forma, três dos cinco boletos são impedidos de serem enviados. O sistema de Faturamento recebe então os dois boletos restantes, mas o sistema de Cobrança recebe somente um, pois o outro boleto não é de um cliente inadimplente e, portanto, não é de interesse do sistema. Podemos verificar a partir da simulação que o FSB tem a capacidade de fazer o roteamento seletivo baseado no conteúdo das mensagens e no conjunto de políticas.

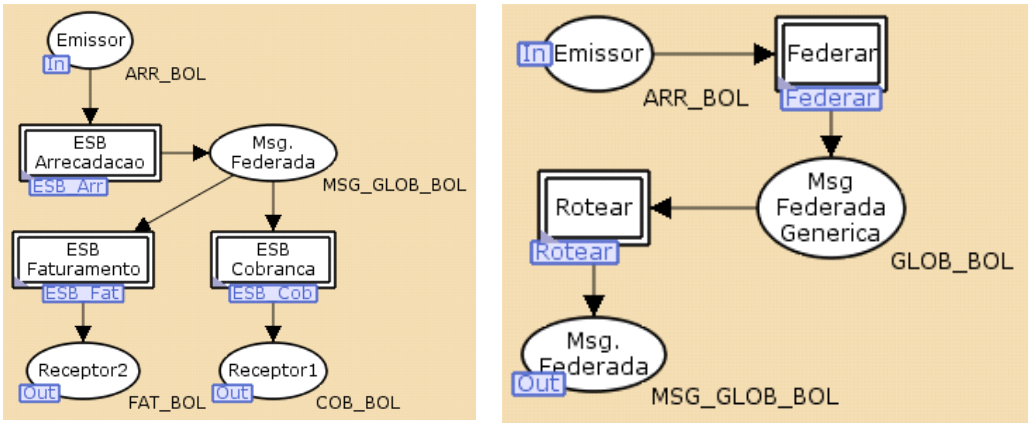


(a) (b)

Figura 4.2: Visão macro do FSB antes (a) e depois (b) da simulação.

A sub-rede da transição de substituição FSB da Figura 4.2(a) é apresentada na Figura 4.3(a). Pode-se verificar que o FSB é composto por três ESBs, cada um representa um sistema. Essa composição de ESBs determina a responsabilidade de cada sistema dentro do *middleware* federativo. Um tipo particular de ficha para boleto é definido para a mensagem federada, de forma que cada ESB somente necessita compreender esse tipo de mensagem, por ser comum à federação.

Explodindo-se a "TR (transição) ESB Arrecadação" (Figura 4.3(b)), vê-se que este é composto por duas transições: uma correspondente à etapa de conversão do boleto do formato interno para o formato global (TR Federar) e outra onde é feito o roteamento (TR Rotear), gerando mensagens no formato apropriado.



(a) (b)

Figura 4.3: (a) FSB e (b) ESB Arrecadação

O modelo referente à transição Federar é apresentado na Figura 4.4. Inicialmente, a mensagem segue dois fluxos paralelos. O primeiro fluxo realiza a aplicação de políticas globais que, no cenário aqui configurado, considera um erro o pagamento de boletos que estejam em negociação. O segundo fluxo faz o enriquecimento da mensagem que resgatará informações necessárias para transformar a mensagem específica do ESB Arrecadação em uma mensagem global. Os dois fluxos são sincronizados e, caso o resultado da política seja falso, é gerado um erro do boleto. Caso contrário, a mensagem é transformada em um boleto no formato global (TR Transformar).

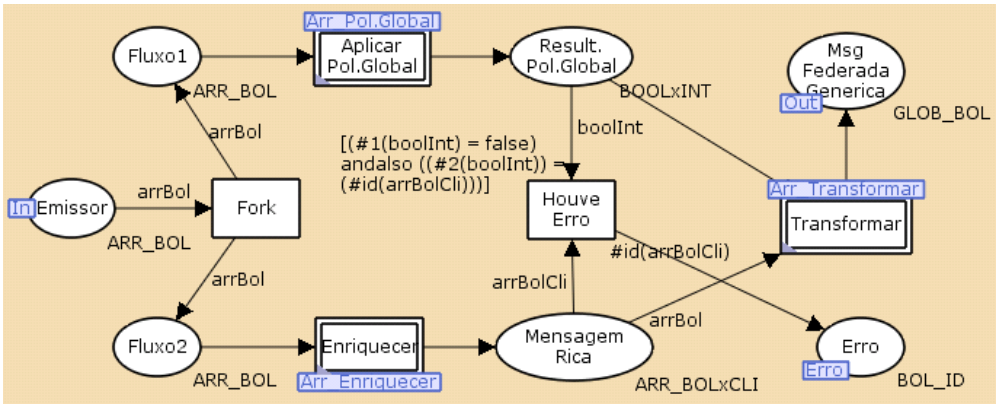


Figura 4.4: Federar mensagem

Observando-se a Figura 4.3(b), vê-se que, após a conversão para o formato correto (TR Federar), a mensagem pode ser enviada (TR Rotear). A Figura 4.5 representa a aplicação das políticas de roteamento (TR Aplicar Políticas de Roteamento) para definir seus destinatários e, finalmente, enviá-la (TR Envia).

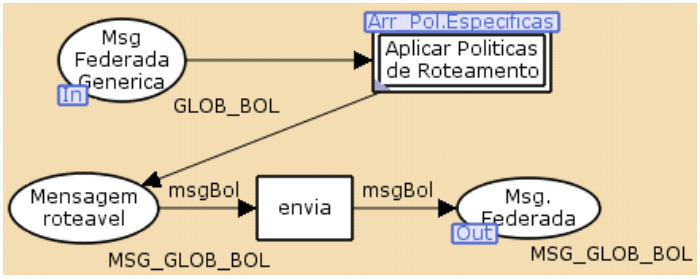


Figura 4.5: Rotear

A Figura 4.6 representa a TR Aplicar Políticas de Roteamento, onde se pode verificar que é adicionado um usuário à mensagem. Neste ponto são geradas mensagens roteáveis para os ESBs de Cobrança e de Faturamento, entretanto o conteúdo da mensagem não é o mesmo. No caso do Faturamento, a data de pagamento não é relevante, assim essa informação não é enviada.

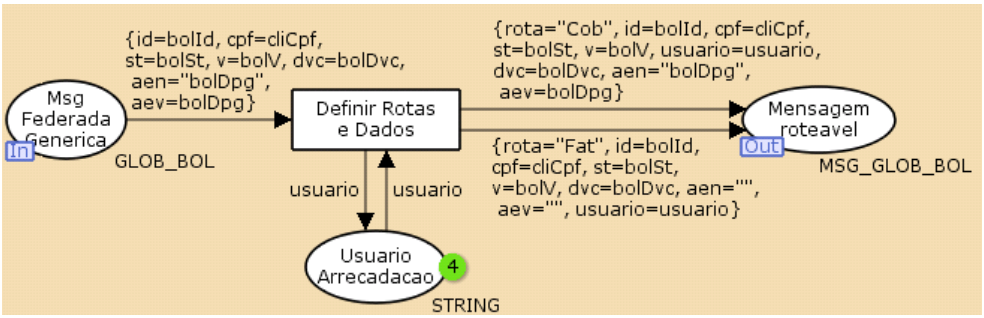


Figura 4.6: Política de roteamento

Uma vez enviada, tanto o ESB de Cobrança quanto de Faturamento recebem a mensagem, como pode ser visto na Figura 4.3(a). A Figura 4.7 representa o ESB de Cobrança que é semelhante ao ESB de Faturamento. Inicialmente, são criados dois fluxos: o primeiro aplica as políticas federativas e faz o registro de todas as mensagens. O segundo fluxo faz o enriquecimento da mensagem ao adicionar as informações necessárias para transformar o boleto de tipo genérico em um boleto reconhecido pelo sistema de Cobrança.

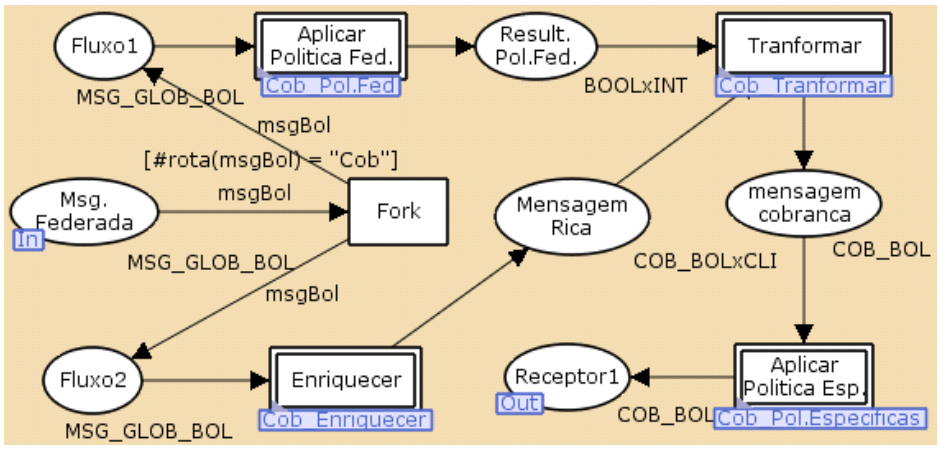


Figura 4.7: ESB Cobrança

A diferença entre os fluxos de Cobrança e de Faturamento está nas políticas específicas. No caso de Cobrança é aplicado um filtro que apenas deixa passar boletos de clientes inadimplentes, como mostra a Figura 4.8(a). O ESB do Faturamento possui um contador de boletos pagos, que é incrementado a cada boleto recebido pela rede representada na Figura 4.8(b). A aplicação dessas políticas faz com que, nesta simulação, o sistema de Cobrança tenha recebido somente uma mensagem e o Faturamento duas, embora ambos ESBs tenham recebido duas mensagens. Esse tipo de lógica não poderia ficar no ESB de Arrecadação, pois saber quais clientes estão inadimplentes foge ao seu escopo.

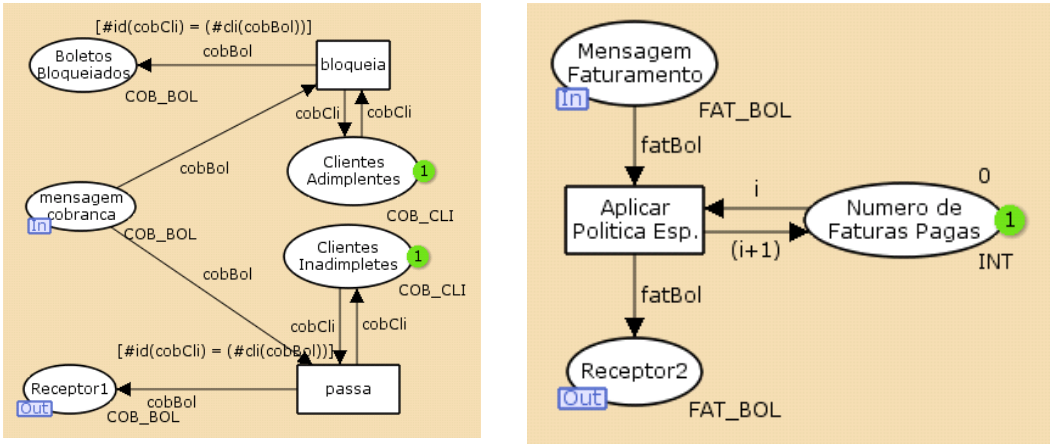


Figura 4.8: Aplicar políticas específicas do Cobrança (a) e do Faturamento (b)

4.1.2Implementação com *Web Service* e BPEL

A modelagem com redes de Petri colorida validou o fluxo e permitiu simulações e análises matemáticas que garantem a vivacidade esperada e ausência de *deadlocks* no fluxo. A compatibilidade do FSB com especificações atuais de SOA é demonstrada no

desenvolvimento de fluxos BPEL e *Web Services* baseados na modelagem de redes de Petri.

A modelagem gráfica do BPEL descrita nesta seção cria um *workflow* para cada FSB. Tem-se uma classificação clara de dois tipos de FSB, o de transmissão e recebimento, de forma que a representação gráfica do BPEL fica quase idêntica tanto nos estudos de caso apresentados nesta dissertação, como em outras modelagens feitas durante o estudo realizado. A diferença gráfica de fluxos do mesmo tipo está no conjunto de *Web Services* que o BPEL usará.

O primeiro BPEL, apresentado na Figura 4.9, é o de transmissão, ou seja, o ESB do sistema de arrecadação. Assim como na rede de Petri, o fluxo inicia-se com uma mensagem de pagamento de boleto. Em seguida, para melhorar o desempenho, dois fluxos são executados ao mesmo tempo, isto é, em paralelo: a aplicação de políticas e o enriquecimento de informações.

Neste momento já existem duas diferenças sensíveis em relação à rede de Petri. A primeira é que no BPEL há a presença de *Web Services* que executam atividades específicas que tornam o BPEL quase o mesmo para ESBs de transmissão. A segunda diferença é que a etapa de enriquecimento é facilitada pelo fato que o BPEL tem a capacidade de guardar as informações de retorno em variáveis internas, sem necessidade de criar novos tipos que contenham todas as informações, como no caso da rede de Petri.

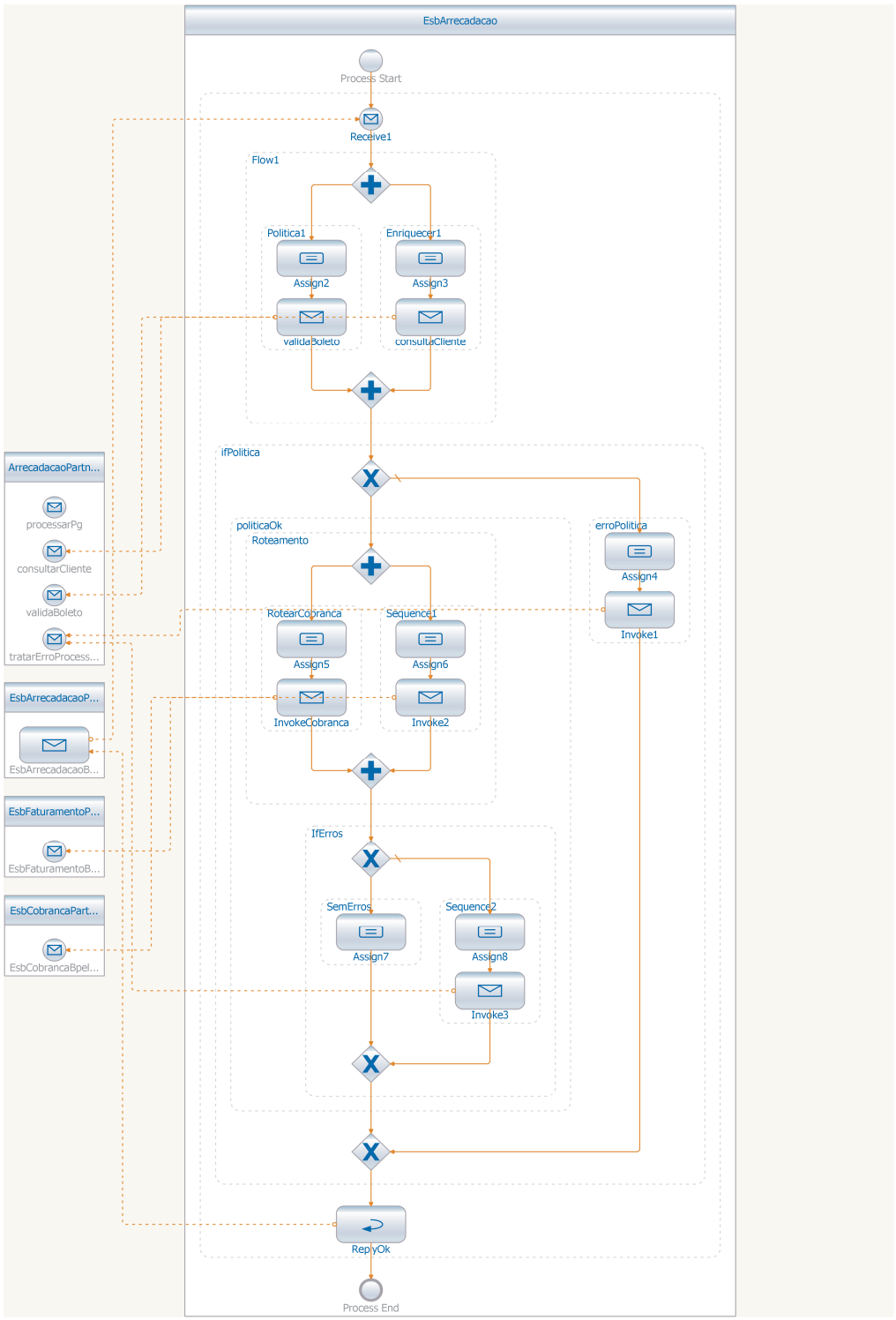


Figura 4.9: BPEL de Transmissão

Depois de aplicada a política, o resultado dessa etapa é verificado. Caso tenha falhado, entra-se em um fluxo de tratamento de erro que executa um *Web Service* e traz como resultado do BPEL um erro. Caso seja bem sucedido, o roteamento é realizado. Até este ponto, todas as políticas eram realizadas com *Web Services*, mas para o roteamento, a princípio só é necessário uma atividade de "*Assign*" que permite manipular o conteúdo que cada mensagem conterá de acordo com o destinatário. Em um fluxo mais completo poderíamos ter a presença de um condicional para que, em determinadas circunstâncias a mensagem nem seja enviada.

Um aspecto importante para observar é o fato de que os pontos de retorno desse fluxo são as últimas atividades executadas, com isso se tem um comportamento síncrono do fluxo. Caso se queira um comportamento assíncrono, pode-se elaborar um BPEL que não tenha ponto de retorno, ou outro em que o ponto de retorno seja a primeira atividade, apenas sinalizará que o *workflow* foi iniciado.

O segundo BPEL, apresentado na Figura 4.10, é o do sistema de cobrança, ou seja, é do tipo ESB de Recepção. Assim como o BPEL de transmissão, esse se inicia ao executar dois fluxos paralelos, uma para aplicação de políticas da federação e outro para enriquecimento. O fluxo segue transformando a mensagem do formato global para o específico e, em seguida, aplicará políticas específicas. Uma vez que todas as políticas sejam validadas, o serviço alvo é executado.

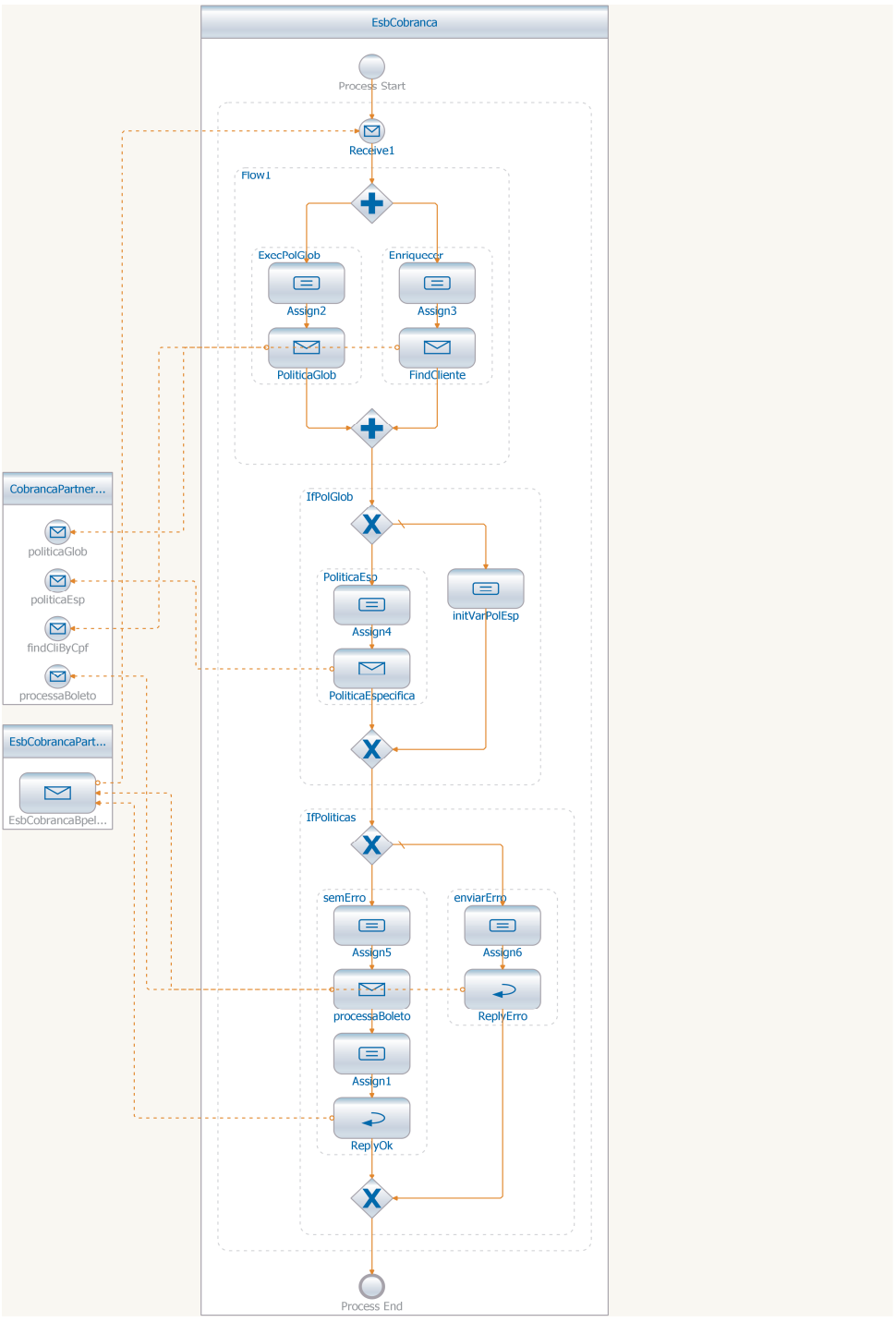


Figura 4.10: BPEL de Recepção

4.2 Estudo de Caso 2: Sistema concentrador de alarmes

Uma empresa prestadora de serviços em telecomunicações possui um sistema capaz de processar alarme da rede de seus clientes. Esta empresa tem basicamente dois serviços. O primeiro serviço é o encaminhar um técnico para resolver alarmes de erro. Como segundo serviço, o sistema é capaz de processar os alarmes enviados, que podem ser de erro ou aviso e predizer possíveis erros que possam ocorrer. Ambos os serviços fazem parte do sistema concentrador de alarmes.

4.2.1 Modelagem e Simulação baseadas em redes de Petri coloridas

Em nossa modelagem, esta prestadora de serviço possui dois clientes, entretanto um destes clientes opera com rede fixa e o outro com rede móvel, como vemos na Figura 4.11. O cliente de rede fixa contratou os dois serviços, enquanto o da rede móvel contratou somente o serviço de encaminhamento de técnico para resolver alarmes de erro.

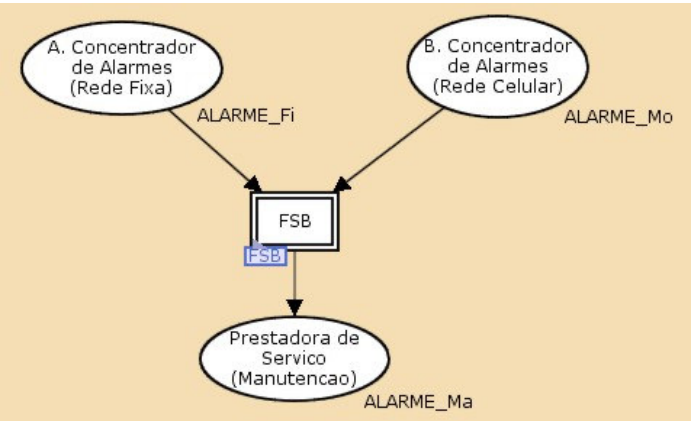


Figura 4.11: Visão geral da PN do sistema concentrador de alarmes

Ao examinar a Figura 4.12, pode-se perceber que a maior diferença entre este estudo de caso e o anterior é que enquanto neste estudo de caso, o sistema receptor recebe mensagens de mais de um sistema transmissor, enquanto que no sistema de cobrança ocorria o contrário.

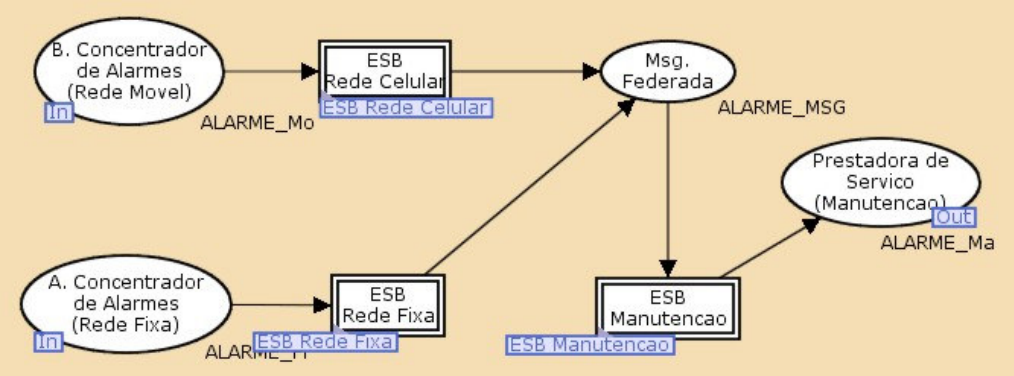
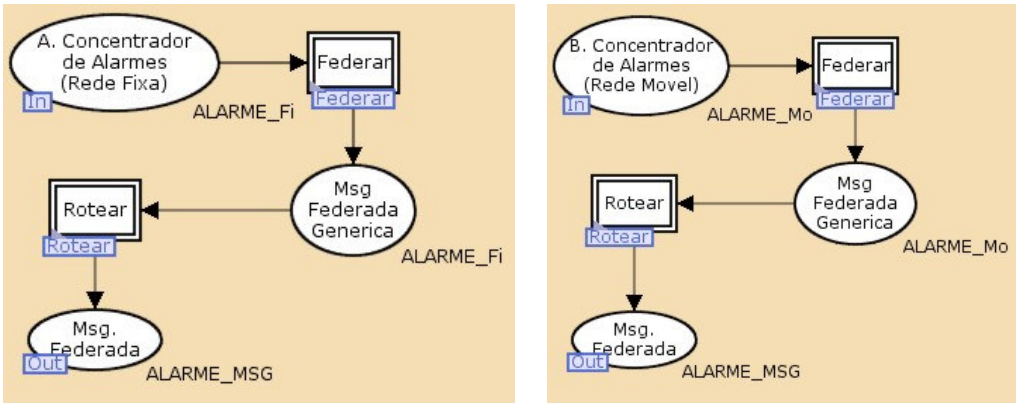


Figura 4.12: FSB do sistema concentrador de alarmes

Podemos verificar que o funcionamento do FSB continua o mesmo, de forma que os ESBs dos sistemas transmissores (ESB Rede Celular e ESB Rede Fixa) geram uma mensagem federada que será consumida pelo ESB do sistema receptor (ESB Manutenção).



(a) (b)

Figura 4.13: FSB da Rede Fixa (a) e Rede Móvel(b)

Os ESBs dos sistemas transmissores (ESB Rede Celular e ESB Rede Fixa) são modelados de forma bem semelhante, como podemos ver na Figura 4.13. A diferença entre ambos está nas sub-redes que implementam as políticas específicas, principalmente no roteamento, que na rede fixa envia todas as mensagens e a rede móvel envia somente as de erros. Essa diferença ocorre pela diferença de contrato entre as duas e a empresa prestadora do serviço.

Se compararmos a modelagem da rede do primeiro e segundo estudo de caso até este ponto, verificamos que são quase iguais, entretanto deste ponto em diante, seriam idênticas, exceto para as políticas específicas. Já podemos verificar esta repetição se compararmos a Figura 4.13 e a Figura 4.3(b). Esta repetição, aliado ao fato que ambas as redes executam com sucesso e chegam ao estado final esperado, comprovam que o fluxo básico do FSB, descrito na sessão 3.3, é aplicável e replicável em situações distintas.

4.3 Conclusão

Vimos dois estudos de caso que cobrem situações onde existe mais de um receptor para um único emissor (sistema de tarifação) e o contrário, isto é, quando existe mais de um emissor para o mesmo receptor (sistema concentrador de alarmes). Ambos os casos foram modelados com redes de Petri onde conseguimos a partir, de análises feitas pelo CPN Tools [CPN Tools 2008], validar propriedades da rede como dead locks, lugares ou transições mortos.

Em ambos os estudos de caso, foi possível perceber que o fluxo modelado com redes de Petri foi quase idêntico, havendo alterações somente referentes aos tipos de

dados e implementação de políticas. De fato o fluxo entre emissores ou entre receptores é tão semelhante que se mostrássemos o diagrama BPEL, estes só mudariam o nome do serviço.

5 Conclusões Finais

5.1 Principais Contribuições

Nesse trabalho, propomos um *middleware* de comunicação baseado em ESB para sistemas federados, denominado barramento de serviços federados ou FSB. O FSB mantém todas as características positivas do ESB e supera sua limitação em ambientes federativos. O FSB mostrou-se alinhado com SOA e técnicas de *workflow*, permitindo realizar modelagem e simulação com redes de Petri coloridas e BPEL.

A modelagem por redes de Petri coloridas permitiu a simulação e a validação da proposta. Dessa forma, ganhos significativos foram obtidos na implementação com o uso de BPEL. De fato, modelagem com redes de Petri coloridas não só validou o fluxo, como o documentou em detalhes e possibilitou a diminuição no número de erros.

Por usar como base o ESB, especificações baseadas nesse conceito como a JBI podem ser usadas para realização do FSB. Realizou-se a implementação em BPEL e a implantação em servidores com suporte a JBI. No caso especial dessa especificação, como está alinhada ao Java EE, também agregam-se funcionalidades do tipo "clusterização", monitoração e *cache* distribuído, entre outras.

Podemos classificar o FSB no modelo lógico como muito para muitos, pois podem existir múltiplos emissores e receptores. Ainda no modelo lógico também classificamos como assíncrono, pois após a mensagem ser transmitida não existe

motivos para o sincronismo do retorno uma vez que em caso de erro uma função callback será chamada. Um resumo pode ser visto na Tabela 7.1.

Tabela 5.1: Modelo lógico do FSB

Classificação	*Conformidade/Observações*
muitos-para-muitos (*many-to-many*)	Atende. Há suporte para múltiplos emissores e receptores. Uma única mensagem pode ser enviada para vários receptores.
um-para-um (*one-to-one*)	Suportado, embora não haja grandes vantagens em adotar essa abordagem
Assíncrono	Atende. Após a mensagem ser transmitida não existe motivos para o sincronismo do retorno uma vez que em caso de erro uma função callback será chamada evitando a necessidade de bloqueio.
Síncrono	Suportado.

Em nossos estudos de casos, podemos classificar fisicamente como sem conexão, uma vez que o evento é realizado em uma única mensagem e sem necessidade de estabelecer uma seção de comunicação, e de comunicação direta. Essa foi uma característica da forma como implementamos o FSB com *Web Services* e BPEL, mas seria perfeitamente possível implementar com seção, sendo todas as mensagens de um transmissor enviadas na mesma sessão, e de comunicação intermediada por filas. Uma classificação mais ampla pode ser visto na Tabela 7.2.

Tabela 5.2: Modelo físico do FSB

Classificação	*Conformidade/Observações*
Sem conexão	Atende. Em nosso estudo de caso foi adotado este modelo.
Com conexão	Suportado, é possível utilizar uma mesma sessão do BPEL, por exemplo, para receber todas as mensagens de um emissor.
Comunicação direta	Atende. Um fluxo BPEL pode ser tanto exposto de forma síncrona como assíncrona, sem comprometer seu fluxo interno.

Comunicação intermediada por fila de mensagens	Suportado. É possível substituir as chamadas remotas por mensagens com tal intermediação. É uma pratica usada em fluxos BPEL
Comunicação por publicação e assinatura	Suportado. Ambientes JBI, como o que usamos na implementação, permite que a entrada de mensagens seja feita a partir de filas JMS que tem suporte a publicação e assinatura.
Comunicação por requisição e resposta	Atende. Em nosso exemplo o BPEL foi exposto desta forma.
Comunicação negociada	Atende. A aplicação de políticas permite controlar o estado das regras de negócio de forma a negociar transações dependentes de estado com as aplicações.

Em relação ao tipo, como usamos o ESB, classificado como um MOM, como base do FSB, nossa proposta também herda essa categorização sendo "Orientado a Mensagens", entretanto podemos classificá-lo de outras formas. No fluxo do FSB são feitas várias chamadas remotas, podendo ser classificado também como "Chamada a Procedimentos Remotos". Também podemos perceber na implementação em BPEL e redes de Petri colorida para que o fluxo ocorra, existe a necessidade de transações, assumindo também a classificação de "Monitor de Processamento de Transações". Uma classificação mais ampla pode ser visto na Tabela 7.3.

Tabela 5.3: Tipo de tecnologias de integração do FSB

Tipo	*Conformidade/Observações*
Orientados a Dados	Suportado. O FSB pode usar como infraestrutura o JBI que tem suporte a esse tipo de integração.
Orientados a Mensagens	Suportado. O FSB pode usar como infraestrutura o JBI que tem suporte a esse tipo de integração.
Chamada a Procedimentos Remotos	Suportado. O FSB pode usar como infraestrutura o JBI que tem suporte a esse tipo de integração.
Objetos Distribuídos	Suportado parcialmente. O FSB é baseado em SOA que só garante o comportamento correto de serviços, que pode ser considerado um subconjunto de Objetos Distribuídos

Monitores de Processamento de Transações	Atende. O BPEL tem essa funcionalidade residente.
Servidores de Aplicação	Não atende. O FSB pode rodar em um servidor de aplicação com suporte a JBI, mas ele em si não é um servidor de aplicação.

Em nossa simulação, verificamos que não existe dependência direta entre os serviços, satisfazendo os requisitos de SOA com serviços autônomos, além de também estar de acordo com arquitetura orientada a eventos. Considerando que a entidade sistema como a composição do próprio sistema em si, contendo lógica de negócio, mais o ESB do mesmo, responsável pela lógica de integração, também podemos concluir que o FSB é uma integração compatível com o NGOSS. Essa conclusão é tomada, pois todas as condições são atendidas como mostrado na Tabela 5.4, tomando como base a rede de Petri modelada e um desenvolvimento baseado em JBI.

Tabela 5.4: Conformidade FSB/NGOSS

Requisito NGOSS	*Conformidade/Observações*
Modelo de Informações Compartilhada e Modelo de Dados	Atende. Utilizamos uma representação global das entidades boleto e cliente.
Modelo de Políticas de Segurança	Atende. Embora não esteja presente em nossa simulação, especificações com JBI já trazem mecanismos para essa funcionalidade.
Modelo de Gerenciamento de Políticas	Atende. Nossa simulação traz algumas políticas.
Automação do processo de negócio	Atende. O FSB se utiliza de *workflows* que é uma técnica de Automação do processo de negócio. Em ambiente real, a implementação do *workflow* utilizaria BPEL.
Contratos NGOSS	Atende parcialmente. A descrição em ambiente real com o JBI seria feito por WSDL que não possui padronização para representar pré e pós-condições.
Aplicações de Negócio de OSS	Atende. Embora a aplicação não precisa ser classificada como NGOSS, o FSB a torna NGOSS.

Framework de Serviços	Atende. A especificação JBI possui todos os serviços necessários
Mecanismos Básicos	Atende. A comunicação entre os ESB que compõem o FSB se dá de uma forma comum.

5.2 Limitações da Proposta

Em nosso trabalho, não entramos no mérito do problema tradicional de Federações que são principalmente a autenticação e permissões e segurança. Existem especificações de *web services* que poderiam ser usadas nas implementações, em especial a WS-Trust e WS-Federation [Camargo 2007] [Alchieri 2007]. Como nossa proposta utiliza uma arquitetura orientada a serviços, é possível adicionar de forma transparente e interoperável a autenticação, autorização e segurança [Camargo 2007]

Outra limitação que existe é a falta de comparação de desempenho do FSB para outros modelos e tecnologias. Utilizamos na nossa implementação um modelo baseado em *web services* e BPEL que possui um processamento extra com manipulação de documentos XML, se compararmos com representação binária das informações.

5.3 Trabalhos futuros

Como trabalhos futuros, vários estudos podem ser realizados para o refinamento desse trabalho. A autenticação das mensagens em ambientes federados, por exemplo, é de extrema importância, podendo envolver o mapeamento de identidades e o uso de serviço de pseudônimo [Chong 2004]. Outros pontos de relevância investigativa são o roteamento, o qual já possui algum tratamento no JBI, mas que pode ser aprimorado, e

aspectos de Qualidade de Serviços (*Quality of Service* – QoS) também podem ser explorados. Além destes, um outro possível estudo em perspectiva para este trabalho são as questões relacionadas à manipulação de mensagens federadas transacionais.

Outro trabalho futuro é a extração de métricas do tempo de execução do FSB, para que seja possível expandir as redes de Petri coloridas com o conceito de redes de Petri temporizada. Com este tipo de rede, é possível verificar o tempo de execução dos fluxos nas simulações facilitando assim a sua otimização. Por exemplo, pode-se adicionar ou remover fluxos paralelos, permitindo melhoria contínua no fluxo do FSB.

Referências bibliográficas

Alchieri, E. A. P., Bessani, A. N., e Fraga, J. S. (2007) "Infra-Estrutura com Segurança de Funcionamento para Cooperação de Serviços Web". In *Anais do 25o. Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)*. Belém, PA, maio

Apache (2008), *Web Service* Project, http://ws.apache.org, janeiro

Ashford C. (2004), "OSS through Java as an Implementation of NGOSS", White Paper, http://www.ossj.org/learning/docs/wp_technologycomparison1.0.pdf, abril

Camargo, E. T., Fraga, J. S., Wangham, M. S., Mello, E. R. (2007) "Autenticação e Autorização em Arquiteturas Orientadas a Serviço através de Identidades Federadas". In: In *Anais do 25o. Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)*, Belém, PA, maio

Chappel, D. (2004), "Enterprise Service Bus", O'Reilly, 1st edition.O'Reilly

Chappell, David A. et al.(2002) Java *web services*. 1. ed. [S.I.], O'Reilly

Chong, F. (2004), "Identity and access management", Microsoft architects journal, p. 20-31, julho.

CPN Tools (2008), http://wiki.daimi.au.dk/cpntools/cpntools.wiki, janeiro.

Dahan, Udi. (2006), "Autonomou Service and Enterprise Entity Agregation", Microsoft architects journal, p. 10-15

Endrei M., et al. (2004), Patterns: Service-oriented Architecture and *Web Services*, IBM Redbook

Erl, T. (2004) Service-Oriented Architecture: A Field Guide to Integrating XML and *Web Services*. Prentice Hall PTR.

Gabrick, Kurt A. et al. (2002), J2EE and XML development**,**. 1. ed. [S.I.], O'Reilly

Geronimo (2008), "Apache Geronimo", http://geronimo.apache.org/, janeiro

GlassFish (2008), "GlassFish – Open Source Application Server", https://glassfish.dev.java.net/, janeiro.

Heuser C. (1990), "Modelagem Conceitual de Sistemas: Redes de Petri", V EBAI

Hohpe, Grego (2006). "Programmieren ohne Stack: Ereignis-getriebene Architekturen", March/April 2006, ObjektSpektrum, disponível em http://www.sigs-datacom.de/sd/publications/pub_article_show.htm?&AID=1791&TABLE=sd_article

IBM (2008), alphaWorks BPWS4J: Overview, http://www.alphaworks.ibm.com/tech/bpws4j, Janeiro

ISO (2008), The ISO 17799 Directory, http://www.iso-17799.com, janeiro

JCP (2008), JSR-208, http://jcp.org/en/jsr/detail?id=208, janeiro.

Jensen K. (1997), "Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use". Volume 2, Analysis Methods – Monographs in Theoretical Computer Science, Springer-Verlag, 2nd corrected printing, ISBN: 3-540-58276-2.

Jensen, K. (1996), “Colored Petri Nets, Basic Concepts, Analysis Methods and Practical Use: Volume 1”, Springer, 2nd edition, 1996.

Juric, Matjaz B., et allal (2001). Professional J2EE EAI. Wrox Press Ltd., UK, EAI Principles, Part I

Liberty (2008), http://www.projectliberty.org/index.php/liberty/about, janeiro.

Linthicum, David S. (2000). Enterprise Application Integration. Information technology series. Addison Wesley.

Murata, T. (1989). Petri nets: Properties, analysis and applications, IEEE Proceedings, Vol. 4, pp. 541–580.

Nadhan, E. G. and Weldson, J. L. (2004), “A strategic approach to data transfer methods”, Microsoft architects journal, p. 44-54, julho.

Putman, J. R. (2001), “Architecting with RM-ODP”, Prentice Hall, USA, 2001.

Richards, M. (2006), “The Role of the Enterprise Service Bus”, http://www.infoq.com/presentations/Enterprise-Service-Bus, outubro.

ServiceMix (2008), “ServiceMix”, http://servicemix.apache.org/documentation.html, janeiro.

Strassner ,John et al. (2004), “TMF White Paper on NGOSS and MDA”, TeleManagement Forum / Object Management Group, abril

Strassner, J. (2003), Policy Based Network Management – Solutions for the Next Generation”, Elsevier

Sun Microsystems (2008), “Service Oriented Architecture”, http://www.sun.com/products/soa/, janeiro.

TMF (2003), “Shared Information/Data (SID) Model – Addendum 1-POL, Common Business Entities Definitions – Policy”, v1.0, Julho 2003

TMF (2006), NGOSS Architecture Overview, http://www.tmforum.org/browse.asp?catID=2009, dezembro

TMF (2008), TM Forum, http://www.tmforum.org/, janeiro

W3C (2008), Word Wide Web Consortium, http://www.w3.org/, janeiro

Apêndice 1 : NGOSS TNA

A1.1 Modelagem

Existem cinco elementos de modelagem definidos na arquitetura NGOSS TNA. Estes elementos modelam todas as informações e dados de um sistema NGOSS, sejam de negócio ou de controle.

A1.1.1 Modelo de Informações Compartilhada e Modelo de Dados

A fim de possibilitar a comunicação, integração e interoperabilidade é caracterizado um modelo comum de informações chamado de modelo de informação compartilhada (*Shared Information Model and Data model* – SID). O SID é mais do que simplesmente uma padronização dos dados do sistema, também definindo semântica e comportamentos das entidades gerenciáveis, além de interação entre as mesmas. O conjunto das informações do SID é utilizado para descrever as informações de domínio em um sistema NGOSS como clientes, ordem de serviço, serviços de rede e definições de configuração. A representação do SID é feita em UML.

A1.1.2 Modelo de Políticas de Segurança

Um sistema NGOSS deve ser projetado seguindo um modelo de segurança requerendo que as operações possam ser configuradas de forma a utilizar um ou mais mecanismos de segurança e políticas de forma que se possa operar um sistema NGOSS de forma segura. Esses mecanismos de segurança são objetos de estudo da ISO 17799 [ISSO 2007] que apresenta as melhores práticas em segurança da informação e apresentam um

framework para gerenciar e operar um sistema NGOSS para unir os a segurança com as operações do sistema.

A1.1.3 Modelo de Gerenciamento de Políticas

A arquitetura de um sistema NGOSS é definida como uma arquitetura de gerenciamento habilitado à política. De forma geral, políticas provêem regras que regem os comportamentos de um sistema.

O NGOSS utiliza o modelo de política DEN-ng que está documentado em [TMF 2003] e [Strassner 2003], sendo parte de um modelo maior também chamado DEN-ng. Todos os aspectos de negócio do DEN-ng foram submetidos ao TMF e modelados junto ao SID [TMF 2003].

Podemos ver na , no modelo DEN-ng, a regra de uma política está associada a um evento. Os possíveis eventos estão descritos no modelo de eventos. Após o disparo do evento, é verificado se a condição de execução da política é satisfeita e, caso seja, uma ação associada à regra é executada.

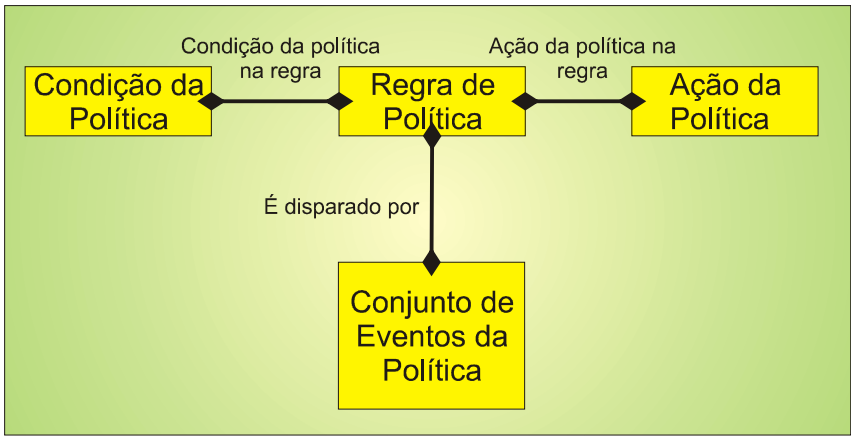


Figura A.1: Modelo de políticas

As regras de políticas são definidas para localizar objetos de negócio, sistemas, entre outros objetivos, dessa forma, políticas podem fazer a ligação entre objetos de negócio, sistemas e implementações de apresentação de sistemas NGOSS uns com os outros.

A1.1.3.1 Automação do processo de negócio

Um sistema NGOSS é composto a partir de um conjunto de serviços que realizam operações mais simples sendo orquestrado utilizando scripts ou alguma tecnologia de gerenciamento de processos. Para tanto, deve haver a separação da implementação do comportamento dos componentes e do software que automatiza o processo de negócio a partir dos componentes. O modelo de processo pode ser especificado usando os procedimentos do processo de negócio.

Gerenciamento de processos é a aplicação de técnicas modernas de gerenciamento de negócio para processos de negócio. Isso inclui técnicas para definir, mensurar, analisar, testar e disponibilizar o processo de negócio assim como o executar. Todas essas atividades fazem parte do gerenciamento de negócio e podem contribuir para a meta de melhorar os resultados de negócio dos OSS.

Automação do processo de negócio também é chamada de Modelo de Processo de Negócio, e suas técnicas e atividades são partes de um estudo maior chamado de Gerenciamento do Processo de Negócio (*Business Process Management* – BPM).

A1.1.4 Contratos NGOSS

O contrato em NGOSS é a unidade fundamental de interoperabilidade. Pode-se dizer que, no mínimo, um contrato é a especificação da interface para um serviço. Contudo,

para que um contrato possa ser a unidade fundamental de interoperabilidade é necessário especificar mais que simplesmente uma interface, deve também definir semântica de interação.

Basicamente, um contrato especifica três pontos:

- Uma representação neutra a tecnologia de uma interface;
- A interação de entidades participantes no contrato;
- A definição do comportamento das entidades participantes na interação específica.

Deste modo, a funcionalidade de um serviço é se torna disponível através de uma interface com o contrato específico e um ou mais serviços tomam parte em uma interação regida pelas restrições comportamentais especificadas no contrato da interação.

A1.2 Aplicações de Negócio de OSS

As Aplicações de Negócio de OSS provê serviços que realizam uma funcionalidade específica relacionada ao negócio usando NGOSS como cobrança, gerenciamento de dados da rede, entre outros. Essas aplicações podem ser classificadas de duas formas:

A1.2.1 Aplicações integradas ao NGOSS

São componentes de software que foram concebidos para serem usados em um ambiente NGOSS. Suas funcionalidades são acessíveis através das interfaces conceituais do NGOSS. Os dados são visíveis externamente conforme mapeado no SID. As funcionalidades do processo de negócio e o gerenciamento de políticas são separados da

implementação da aplicação e especificados usando um conjunto de procedimentos do processo de negócio. Dessa forma, o processo de negócios e as técnicas de gerenciamento de política são aplicados separando o processo de negócios das definições de políticas que por sua vez orquestram o fluxo entre componentes e aplicativos.

A1.2.2 Aplicações legadas integradas

São componentes de software que foram desenvolvidas fora do escopo da arquitetura NGOSS e que tem que se tornar disponível como componente NGOSS. Para tornar possível a utilização desses sistemas em um ambiente NGOSS é necessário o desenvolvimento de componentes NGOSS que encapsulem as funcionalidades desejadas da aplicação legada de forma a satisfazer as exigências da arquitetura NGOSS.

A1.3 Framework de Serviços

Os *frameworks* de serviços provêem funcionalidades da computação distribuída para o correto funcionamento da implementação arquitetural. Interfaces para esses serviços são disponibilizadas através da especificação de contratos apropriados. Podem ser divididas em duas subclasses:

A1.3.1 Framework de serviços OSS

São serviços disponibilizados por OSS padrões e traz funcionalidades comuns aos OSS de forma a poderem ser utilizadas em uma orquestração de serviço. Os *frameworks* de serviço OSS não são obrigatórios, embora sejam incluídos de alguma forma no desenvolvimento de uma solução OSS. Exemplos desses serviços são Auditoria e *Loggin*.

A1.3.2 Framework de serviços básicos

Oferecem as funcionalidades de distribuição básicas necessárias para suportar os modelos de interação entre componentes implementados nos serviços de negócios, podendo também ser usados por outros serviços. São obrigatórios em um sistema NGOSS. Cada sistema NGOSS deve incluir pelo menos uma instância de cada serviço de Distribuição e Transparência que incluem os Serviços de Framework Básico (como *naming*, *registration*, ou *service location*). Por exemplo, a definição do processo de cobrança poderia ser instanciada fazendo uma busca no repositório por uma instância ativa do serviço.

A1.3.2.1 Serviços de distribuição e transparência

São definidos como uma NGOSS TNG (*Technology Neutral Architecture*) a nível de serviços neutros. Estes serviços são fundamental para a construção, desenvolvimento e uso de soluções em um ambiente NGOSS. Uma instância de cada Serviço deve ser incluída em cada desenvolvimento NGOSS. Os serviços de *framework* de transparência de distribuição incluem:

- *Naming*: Para proteger usuários das complexidades futuras (e inerente incompreensibilidade) de tipos diferentes de informações de gerenciamento, são definidos *namespaces*.
- *Repositories* (ou *registry services*): Armazena e oferece informações sobre os serviços distribuídos disponíveis.

- *Registration Services*: Fornece serviços incluindo a adição, modificação e remoção de serviços do Repositório, e a habilidade de paginar serviços previamente adicionados ao Repositório.

- *Service Location Services*: Facilita a localização entre clientes e servidores potenciais. Estes Serviços aceitam requisições por um determinado serviço de clientes e retornam os registros dos provedores para aquele serviço armazenado no Repositório.

A1.3.2.2Federação

O NGOSS prevê uso em ambiente federativo conforme está descrito no capítulo 2.

A1.4 Mecanismos Básicos

Os mecanismos básicos provêem as funcionalidades básicas que tem que ser implementadas em qualquer desenvolvimento NGOSS para prover o *Inter-Working* entre componentes distribuídos independentemente e para suportar o paradigma de *plug-and-work* no nível mais fundamental.

A1.4.1 Mecanismo de comunicação comum

Sistemas NGOSS podem ser caracterizados pela existência de um mecanismo de comunicação (como um barramento de mensagem). Todas as entidades de Software se utilizam de um ou mais mecanismos de comunicação para se comunicar uns com os outros. Cada mecanismo de comunicação oferece um ou mais mecanismos de transporte. Os mecanismos de comunicação podem representar mapeamentos de uma tecnologia específica e deve prover mecanismos de transportes consistentes com estilos de interação básicas definidas na arquitetura e qualquer política de segurança que um

serviço tenha definido para sua rede. Mais importante, o uso de um mecanismo de comunicação possibilita a padronização das operações do sistema pela rede, mensagens, ou eventos que podem ser distribuídos para componentes de interesse. Esse é um ponto importante pois o transporte somente carrega as informações e por se próprio não prove a interoperabilidade.

A1.4.2 Mecanismos de Invocação

Esses mecanismos possibilitam uma forma comum para executar os passos associados com a chamada de uma operação em uma instância de um serviço. Existem vários passos associados com a invocação de uma operação em uma instância de serviço: localização da instância provedora do serviço, uso correto do mecanismo de comunicação, tratamento dos resultados e erros, entre outros. Políticas e segurança têm que ser aplicadas em cada passo do processo. Por essa razão, a chamada de mecanismos tem que ser definida de forma conveniente para percorrer todos esses passos e garantir a integridade das operações no contexto do sistema como todo e do usuário do serviço chamado.

Apêndice 2: Redes de Petri

Nossa proposta basicamente define um *workflow* padrão de ESBs para criar o FSB e, neste sentido, a técnica de redes de Petri foi escolhida como um formalismo matemático capaz de criar *workflows* tanto os simulando como extraindo algumas informações relevantes, tais como detecção de condições de impasse (*deadlocks*).

A2.1 Breve histórico

Carl Adam Petri apresentou em sua tese de doutorado "Kommunication mit Automation" a teoria inicial de redes de Petri, em 1962, na faculdade de Matemática e Física da Universidade de Darmstadt, na Alemanha. Essa tese formulou a base para a teoria da comunicação entre componentes assíncronos de um sistema de computação.

Na década de 1970, as redes de Petri (Petri Nets, PN) começaram a ser aplicadas na modelagem de linguagens de programação, de componentes de hardware e controle de processos, de sistemas distribuídos e protocolos de comunicação. Ainda nesta década, surgiram três tipos de PN com capacidade de modelar diferentes características temporais: as PN Temporizadas de *Ramchandani*, as de *Merlin* e as de *Sifak*. [Jensen 1997]

As redes de alto nível (e.g. coloridas e numéricas) surgiram na década de 1980 possíbilitando que as PN alcançassem outras áreas como automação de escritórios, bancos de dados, inteligência artificial e sistemas de informação. Na década de 1990 houve a consolidação das redes de petri com o surgimento da segunda versão das redes

coloridas, desenvolvidas por Kurt Jensen, da Århus Universitet (Dinamarca), que uniam as características das redes coloridas e temporizadas [Jensen 1992] .

5.3.1 Representação gráfica das redes de Petri

As redes de Petri [Murata 1989] são um formalismo matemático e gráfico que permitem a especificação formal de sistemas e protocolos. As PNs são representadas por dois nós distintos: lugares transições, interligados por arcos, como podemos ver na .

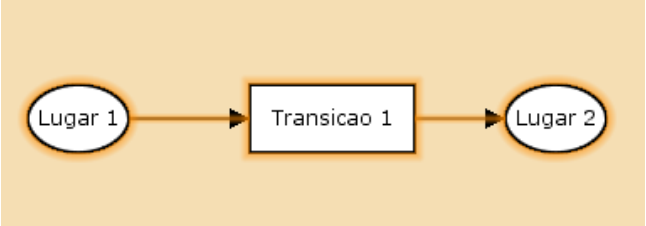


Figura A.2: Rede de Petri simples sem marcação

Os lugares armazenam fichas (inteiros positivos) que podem representar informações, determinando pré-condições para o disparo das transições a qual são ligados pelos arcos de entrada, como é mostrado na .

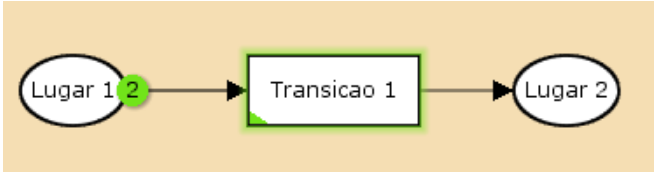


Figura A.3: Rede de Petri simples com marcação

Lugares também recebem fichas através de arcos de saída das transições, geradas com o disparo de transições. Através do disparo de transições, pode-se simular o comportamento dos modelos desenvolvidos.

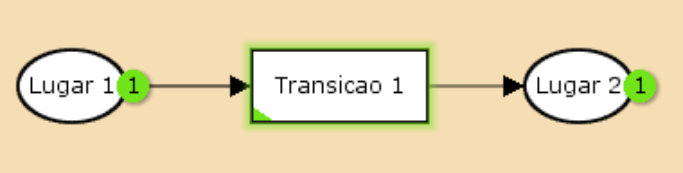


Figura A.4: Rede de Petri simples com transição disparada

As redes de Petri coloridas (Colored Petri Network – CPN) [Jensen 1992] são uma extensão das PNs, em que tipos de dados são associados aos lugares, o que possibilita às fichas possuírem diferentes valores. Nesta dissertação usamos variações das redes de Petri coloridas suportada pelo CPN Tools [CPN Tools, 2008]. Os arcos possuem uma ou mais variáveis do mesmo tipo que o lugar associado. As transições, representadas graficamente por retângulos, podem possuir guardas associadas que condicionam a sua habilitação além das pré-condições estabelecidas pelos arcos de entrada.

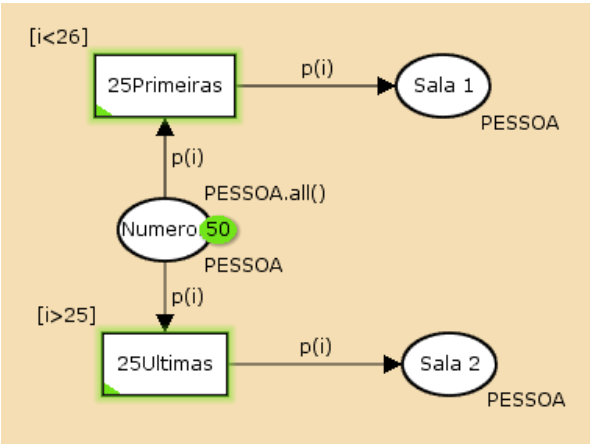
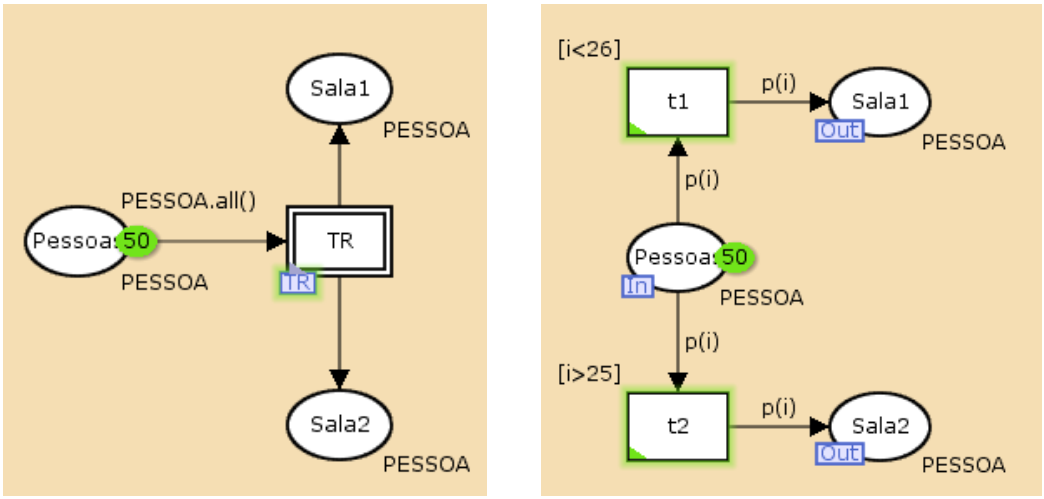


Figura A.5: Exemplo de rede de Petri colorida

A mostra um exemplo de rede de Petri colorida. Nesta rede os lugares são do tipo “PESSOA”, que é uma enumeração de “p” que varia de p(1) a p(50), neste caso temos cinqüenta cores distintas de pessoas. Os arcos possuem a variável “i” que serve como índice para “p”.

Nas CPNs, as transições que possuem um retângulo interno indicam que há uma sub-rede representada pela transição, definindo uma hierarquia. O exemplo da expresso com subrede pode ser modelado conforme a . Este exemplo mostra na (a) uma transição TR com um retângulo interno que sinaliza a existência neste ponto de uma subrede. A subrede mostrada na (b) possui um lugar de entrada (In) e dois de saída (Out), pois a rede principal possui um arco de entrada e dois de saída.



(a) (b)

Figura A.6: Exemplo de rede principal (a) e subrede (b)

A2.2 Representação formal das redes de Petri

As redes de Petri também podem ser representadas seguindo um formalismo matemático. Para nossa proposta, utilizamos somente a representação gráfica, que

internamente é transformada em vetores e matrizes, portanto não nos estenderemos neste vasto assunto. Nesta dissertação apresentaremos somente a representação formal.

As redes de Petri coloridas são representadas pela tupla uma tupla CPN = **(S, L, T, A, N, C, G, E, I)** [Jensen 1997], onde:

- **S** é um conjunto finito não-nulo de **tipos,** também chamados conjuntos de cores;
- **L** é um conjunto finito de **lugares;**
- **T** é um conjunto finito de **transições;**
- **A** é um conjunto finito de **arcos,** de forma que:
    - **L x T = L x A = T x A = 0**
- **N** é uma função **nó,** definida de **A** em **LxT u TxL;**
- **C** é uma função **cor,** definida de **L** em **S;**
- **G** é uma função **guarda,** definida de **T** em expressões, de forma que:
    - **V teT: [Tipo(G(t)) = B a Tipo(Var(G(t))) í S];**
- **E** é uma função de **expressão de arco,** definida de **A** em expressões, de forma que:
    - **V aeA: [Tipo(E(a)) = C(1)mc a Tipo(Var(E(a))) í S]**
        - onde **l** é o lugar de **N(a);**
- **I** é uma função de **inicialização,** definida de **L** em expressões fechadas, de forma que:

- **V leL: [Tipo(I(l)) = C(1)mc].**

A partir da representação formal é possível matematicamente tanto realizar simulações a partir de disparos de transições, como verificar vários outros aspectos da rede de Petri como vivacidade, transições mortas, lugares inalcançáveis, entre outros [Jensen 1997].