UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Onix
Framework para Integração de Sistemas com Enterprise Service Bus

Gabriel Nunes Menezes
Pedro Castello Branco Galoppini

Trabalho de conclusão de curso submetido à Universidade Federal de Santa Catarina como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação.

Florianópolis - SC
2009/1
Gabriel Nunes Menezes
Pedro Castello Branco Galoppini

Onix
Framework para Integração de Sistemas com Enterprise Service Bus

Trabalho de conclusão de curso submetido à Universidade Federal de Santa Catarina
como
parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Frank Augusto Siqueira, Dr.
Universidade Federal de Santa Catarina
frank@inf.ufsc.br

Banca examinadora

Prof. Lau Cheuk Lung
Universidade Federal de Santa Catarina
lau.lung@inf.ufsc.br

Prof. Vitório Bruno Mazzola
Universidade Federal de Santa Catarina
mazzola@inf.ufsc.br

“O prazer no trabalho aperfeiçoa a obra.”
Aristóteles

AGRADECIMENTOS

Aos pais, por todo apoio, compreensão, e principalmente incentivo durante nossas vidas.

Aos outros membros de nossas famílias que nos deram força para chegar onde chegamos.

Aos nossos amigos que se mostraram ótimos companheiros mesmo nos momentos mais difíceis e sempre apoiaram nossas idéias.

Ao orientador Frank Siqueira, que sempre nos ajudou quando necessário e deu rumo ao nosso trabalho.

À Mariana Macedo Nora pela ajuda na correção textual e por sempre ter apoiado o trabalho.

À Substractum Tecnologia LTDA por ter nos tornado melhores profissionais e por ter sido nossa família durante todos os nossos dias de trabalho.

Resumo

Integração de sistemas é o processo que possibilita a troca de informações entre diferentes componentes de software.

Nas duas últimas décadas pode-se perceber que a integração de sistemas de grandes corporações tem se tornado um problema crônico. Boa parte disso se deve ao fato de que a maior parte do tempo de desenvolvimento é gasto com esforços para se conseguir tornar possível a comunicação entre sistemas de informações complexos.

É sabido que médios e grandes sistemas não podem sobreviver em silos isolados. A necessidade de troca de informações é algo que normalmente nasce junto com suas especificações. Porém, a maior parte dos problemas surge quando os sistemas que serão integrados possuem tecnologias ultrapassadas ou métodos não convencionais de interoperabilidade.

No mercado, pode-se observar uma grande gama de protocolos e tecnologias de comunicação, e não são difíceis de encontrar situações onde sistemas que utilizem tecnologias diferentes precisem trocar informações.

É nesse contexto que este trabalho propõe o desenvolvimento de um *framework* que aplique alguns padrões de integração bem difundidos no mercado, tentando assim diminuir o esforço de desenvolvimento de componentes de integração.

Abstract

System integration is a process that makes information and data exchange between different components of software possible. In the last two decades, it is visible that big corporations systems integration becomes a serious problem. Most of that because the major time spent in development is to make the communication possible between back ends and front ends complex components.

Medium and large sized systems can’t exist in isolated information silos. The need of information exchange usually comes together with systems specification. But, most of problems appear when systems that will be connected make use of outdated technology or non conventional interoperability methods.

A large variety of protocols and communication technology is available in the market, and it’s not difficult to find systems that use different types of technology but needs to exchange information.

In this context, this work looks forward to developing a framework that applies some well known integration patterns, trying to reduce the effort of integration components development.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

API	Application Programming Interface
B2B	Business-to-Business
EAI	Enterprise Application Integration
EIP	Enterprise Integration Patterns
ESB	Enterprise Service Bus
HTTP	Hypertext Transfer Protocol
JMS	Java Message Service
MOM	Message Oriented Middleware
RUP	Rational Unified Process
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
UML	Unified Modeling Language
VETO	Validate, Enrich, Transform and Operate
VETRO	Validate, Enrich, Transform, Route and Operate
XML	eXtensive Markup Language
XP	eXtreme Programming

Índice

1 INTRODUÇÃO

Nas ultimas décadas temos vivenciado uma grande proliferação da tecnologia da informação dentro das empresas. Os sistemas de informação vieram com a intenção de agilizar todos os processos, melhorar a qualidade do serviço ou produto fornecido pela organização aumentando com isso a satisfação dos seus clientes. Porém, em muitos casos, esta digitalização das informações foi ocorrendo de maneira desordenada chegando a um ponto em que existem tantos sistemas diferentes para se alimentar com dados que a eficiência e agilidade dos processos organizacionais foi fortemente prejudicada, afetando diretamente no resultado final do conjunto dos processos: a obtenção do produto ou serviço fornecido pela empresa.

Tornou-se impossível prover o nível de automatização de processos exigido pelo mercado, pois diferentes sistemas que existiam nas empresas utilizavam diferentes padrões de dados e comunicação, dificultando a interoperabilidade.

Interoperabilidade é a habilidade de dois ou mais sistemas (computadores, meios de comunicação, redes, software e outros componentes de tecnologia da informação) de interagir e de intercambiar dados de acordo com um método definido, de forma a obter os resultados esperados [ISO,2008].

Surgiu então um novo desafio: integrar de maneira racional os sistemas existentes, de maneira a seguir o fluxo dos processos das empresas adquirindo maior controle, rapidez, disponibilidade e, finalmente, qualidade nos diversos serviços fornecidos por aquela organização. Baseado neste cenário, construiu-se o conceito de *Enterprise Service Bus (ESB)*, que foi estruturado pela arquitetura orientada a serviços, e tornou-se uma ótima alternativa para aqueles que buscam solucionar os problemas de integração existentes.

O *Enterprise Service Bus* funda seus pilares no uso de padrões de comunicação estabelecidos no mercado, no uso da arquitetura orientada a serviços, na distribuição de carga de processamento entre diversos pontos e na utilização de um *middleware* para gerenciar as mensagens para seus respectivos destinatários. Desta maneira, ele basicamente centraliza o controle e roteamento de mensagens em um ponto, enquanto os outros componentes apenas executarão as regras de negócio aplicadas sobre os dados que por ali circularem sem se preocupar para quem ou onde realizar solicitações. Por isso, o ESB tornou-se uma excelente alternativa para as grandes e médias corporações que possuem grande volume de dados trafegando em seus sistemas.

1.1 Objetivo Geral

O objetivo geral deste trabalho é a criação de um *framework* que tenha as principais características de um ESB e consiga proporcionar a interoperabilidade entre dois ou mais sistemas que utilizem padrões de comunicação estabelecidos no mercado como *Web Services*, *sockets*, email e outros - fornecendo também uma estrutura para se realizar validações, enriquecimento, transformação e roteamento de mensagens de uma maneira simples.

1.2 Objetivos Específicos

- Fornecer uma plataforma de configuração do *framework* através de arquivos XML.
- A ferramenta deve ser estensível para que qualquer desenvolvedor possa aprimorá-la.
- Mensagens devem seguir um fluxo especificado no padrão VETRO.
- O *framework* deve ser capaz de operar com diversos protocolos de comunicação já conhecidos e estabelecidos no mercado.

1.3 Metodologia

A metodologia de desenvolvimento deste trabalho será baseada no XP (*Extreme Programming*), que busca de maneira ágil realizar um desenvolvimento sem tantas documentações quanto as encontradas em projetos que seguem outros padrões. XP é uma metodologia completa para desenvolvimento de software, que inclui estratégias para planejamento, levantamento de requisitos e tudo mais que for necessário para desenvolver aplicações inteiras [ERI2003]. A justificativa de se usar esta metodologia é simplesmente pelo fato de que não se trata de um projeto de grandes proporções e muito menos será desenvolvido por uma equipe com muitos membros.

Alguns princípios deste método que se aplicarão ao desenvolvimento deste trabalho são:

- **Simplicidade no projeto**: Este é principal pilar desta metodologia. O XP evita documentações demasiadas como vistos em outras metodologias de projeto como o RUP. Geralmente a documentação inicial do projeto são casos de uso e a análise de requisitos do sistema. Ao longo das interações e ao término do projeto podem ser anexados outros tipos de documentos que especificam o projeto, no entanto esta não é uma prática obrigatória. É importante ressaltar que esta é uma

metodologia voltada principalmente para a etapa de desenvolvimento, e pode ser inviável de se aplicar em projetos de maior dimensão.

- **Redefinição de requisitos durante o desenvolvimento**: Ao longo da implementação do projeto em si podem ser descobertos novos requisitos para o sistema. Como a fase de projeto deste método é menos detalhada, isso é muito comum de acontecer. Nestes casos, os requisitos devem ser adicionados à lista de implementações da equipe sem maiores burocracias.

- **Programação em pares**: A programação em pares é um principio que ajuda bastante no compreendimento do código por todos os membros da equipe, visto que a documentação no XP não é seu ponto forte, logo, o código deve ser o mais compreensível e auto-explicativo possível. Programando em pares os desenvolvedores buscam uma maneira em comum de como desenvolver o sistema e também trocam conhecimentos buscando sempre a melhor forma de se implementar o projeto.

- **Desenvolvimento Orientado a Testes**: Um dos valores pregados pelo XP que serão de grande valia para este projeto é o desenvolvimento orientado a testes. No decorrer do desenvolvimento dos casos de uso, são feitos os chamados "*Test Cases*" que realizarão os testes das funcionalidades implementadas. Desta maneira vão sendo realizados testes de unidade em cada um dos módulos produzidos, levando a uma grande redução de *bugs* no programa e aumentando a qualidade do software.

- **Desenvolvimento incremental**: Quando os objetivos finais do projeto são atingidos, ou seja, todos os requisitos estabelecidos foram cumpridos, pode haver uma definição de novos requisitos que buscam a evolução do projeto. Desta maneira pode-se dar continuidade ao projeto para que ele sempre esteja atualizado com as necessidades do cliente e tecnologias do mercado.

- **Refatoração**: A refatoração é um processo que visa a melhoria do código. Como foi citado anteriormente, o código é o principal documento e prioridade da metodologia XP. Por isso, ao fim de cada interação pode ser realizada uma re-estruturação de código, revisando-o para deixá-lo mais limpo, organizado,

documentado e até mesmo para encontrar falhas não notadas.

1.4 Organização do Texto

Durante o primeiro capítulo deste trabalho são apresentadas as motivações que deram origem a este trabalho, os seus respectivos objetivos e a metodologia aplicada para se realizar o desenvolvimento do mesmo.

No Segundo capítulo aprofunda-se na questão da motivação do trabalho, tratando a atual situação de integração corporativa nas empresas do mercado.

O Terceiro capítulo explica os *Enterprise Integration Patterns,* ou padrões para integração corporativa.

No quarto capitulo trata-se o conceito de *framework.*

O quinto capítulo conceitua e esclarece o que é um *Enterprise Service Bus*, como funciona e de que elementos ele é composto.

No sexto capítulo apresenta-se o *framework*, chamado Onix, desenvolvido durante a realização deste trabalho.

O sétimo capitulo demonstra um exemplo de utilização do Onix.

Durante o oitavo capítulo são apresentadas as conclusões e sugestões para trabalhos futuros.

2 ESTADO ATUAL E PROBLEMAS NA INTEGRAÇÃO CORPORATIVA

A necessidade de integração atualmente é algo quase onipresente. Não é difícil encontrar sistemas que precisem buscar informações, por menores que elas sejam, em outros sistemas da própria corporação ou de terceiros.

Esta tarefa pode parecer simples se pensarmos em uma empresa de pequeno ou médio porte, onde todos os seus sistemas são implementados pelo mesmo fornecedor, utilizam a mesma linguagem de programação e estão em desenvolvimento contínuo. Infelizmente essa é uma realidade que existe para a minoria. Como escrito em Hohpe et al [EIP2004], grandes corporações podem possuir centenas de sistemas, muitas vezes legados e sem suporte, dezenas de websites, bancos de dados dos mais diversos vendedores e plataformas de hardware de diferentes tipos, e é neste cenário que a integração começa a se tornar um verdadeiro desafio.

2.1 Problemas para a Integração de Sistemas Corporativos

“Quem disser que integração de sistemas é algo fácil deve ser alguém muito inteligente, incrivelmente ignorante ou tem algum interesse financeiro para tentar fazer você acreditar que integração é algo fácil”, assim escreveram Hohpe et al[EIP2004] sobre a Integração de Sistemas Corporativos.

Não há dúvidas que desenvolver uma integração com sistemas de diferentes vendedores é algo muitas vezes difícil, trabalhoso e desafiante. É uma tarefa que pode consumir muito tempo e, ainda assim, resultar em uma solução nem sempre esperada e passível de erros. Dentre as tarefas que envolvem a integração, podemos citar a conectividade entre os sistemas, a formatação e a validação dos dados intercambiados.

Abaixo seguem alguns dos problemas mais comuns relacionados ao assunto:

- **Falta de Documentação** – Ocorre quando o sistema com o qual se deseja conectar não possui qualquer tipo de documentação sobre o formato de dado que aceita para a troca de informação. Caso este sistema não possua uma equipe de suporte que ajude nesta questão a integração pode se tornar algo impossível.
- ***Sistema Legado e Imutável*** – Se o sistema com o qual se deseja trocar informações é um legado e não pode ser alterado, o desenvolvedor da integração pode ter que implementar uma solução radical de transformação na estrutura dos

seus dados internos para que eles sejam compatíveis com o formato aceito pelo sistema legado.

- **Falta de Interface de Integração** – Talvez este seja o pior problema em se tratando de integração de sistemas. Nesse caso, o sistema destino não tem uma interface para integração, seja ela um socket, um Web Service ou outra interface qualquer. Se este sistema destino estiver em constante desenvolvimento, pode-se solicitar uma alteração para adicionar esta interface de comunicação; caso contrário, pode-se tentar integrações em níveis mais baixos, como acesso compartilhado à base de dados.
- **Dificuldade de Manutenção** – Mesmo depois de implementada e em utilização, os problemas com a solução de integração não terminam. Como é sabido, todo software possui *bugs*, e detectar um *bug* em um sistema altamente distribuído pode se tornar um trabalho árduo e estressante. Questões de monitoramento também devem ser pensadas, como o envio de notificações caso o algum sistema falhe.

Acima foram citados apenas alguns dos muitos fatores que tornam a integração entre sistemas uma tarefa desafiante. No decorrer deste trabalho estes problemas serão comentados novamente e algumas soluções serão demonstradas, dentre elas o *framework* que aqui será proposto.

2.2 Estado da Integração de Sistemas

De acordo com uma pesquisa realizada pelo Gartner Inc em 2002 [GAR2002], somente 15% dos sistemas empresariais estavam integrados na época da pesquisa. Tendo em vista o resultado da pesquisa, fica a pergunta, por quê os outros 85% não estão integrados?

Chappel tenta explicar esta questão em seu livro *Enterprise Service Bus* [ESB2004]. Ele coloca a culpa no que chama de “arquitetura acidental”. A arquitetura acidental é o resultado de anos de poucos investimentos na área de tecnologia da informação por parte das organizações, o que acabou por produzir sistemas difíceis de gerenciar e modificar. Na maioria das empresas estes sistemas foram concebidos tendo-se em mente os negócios locais, não sendo preparados para qualquer tipo de interação extra-organizacional.

2.3 Padrões de Integração

Anteriormente, foram mostrados os pontos que oferecem mais obstáculos no que

se trata de integração de sistemas. Felizmente, muitos deles podem ser tratados de forma mais simples, fazendo o uso de padrões de integração. Estes padrões são soluções elaboradas por experientes engenheiros de software para solucionar determinados problemas que costumam ocorrer com certa freqüência.

O próximo capítulo sera destinado a destrinchar este tema e falar da importância dos padrões para o a integração de sistemas e, conseqüentemente, para o presente trabalho.

3 ENTERPRISE INTEGRATION PATTERNS

Para explicar o que são os padrões para integração corporativa, ou *Enterprise Integration Patterns*(EIP), deve-se primeiro explicar o que são os padrões de projetos da Engenharia de Software, pois seus conceitos e finalidades são muito semelhantes.

Fazendo uma analogia com a Engenharia Civil, onde um engenheiro tem a tarefa de projetar um grande prédio, para realizar este projeto, ele irá contar com algumas peças já criadas no passado para resolver alguns tipos de problemas, como uma viga de aço de um certo tamanho para suportar um determinado peso, livrando-o de ter que projetar todas as peças e componentes que farão parte da obra. Na Engenharia de software acontece de forma semelhante, nela existem os chamados padrões de projetos, conhecidos também como *Design Patterns*, estes padrões foram criados para auxiliar na resolução de alguns problemas que ocorrem frequentemente durante o projeto e desenvolvimento de um software.

Já no cenário da integração, existem os padrões para integração corporativa, os quais servem como guias para projetar e desenvolver uma solução robusta e confiável para integrar diferentes sistemas de grande porte. Cada padrão adotado é uma decisão de projeto tomada, por isso deve-se ter muito cuidado e se perguntar diversas vezes: “Devo usar troca de mensagens?”, “Este padrão pode realmente me ajudar nesta situação?”, antes de começar a efetivamente utilizá-los. Os padrões de integração são divididos na estrutura de árvore, onde na raiz da árvore estão os diferentes estilos de integração, e nas folhas estão algumas especializações destes estilos.

3.1 Estilos de Integração

Existem muitos obstáculos no que diz respeito a integração de sistemas empresariais, que tornam essa tarefa algo complexo e difícil de ser realizado. Um dos principais problemas é que cada caso possui seus próprios requisitos e detalhes que nunca são iguais a outro já realizado, e o tratamento destas peculiaridades exigem do grupo projetista atenção especial no momento da concepção do projeto.

Situações comuns de serem encontradas neste cenário são a integração de sistemas separados geograficamente, sistemas escritos em plataformas diferentes e sistemas de vendedores ou parceiros diferentes. Para tratar desta diferentes situações, os projetistas de sistemas podem fazer uso de um leque de opções chamadas estilos de integração, cada um especializado em resolver certo tipo de problema.

3.1.1 Transferência de Arquivos

A transferência de arquivos é a maneira mais simples e talvez a mais utilizada para a integração de sistemas. Os arquivos são estruturas universais que existem em qualquer sistema operacional, tornando essa alterativa interessante quando se precisa integrar sistemas em diferentes plataformas e linguagens e não existe a necessidade deles estarem sincronizados a todo momento[EIP2004] .

Se imaginarmos o cenário de um Sistema A integrando-se com um Sistema B utilizando a transferência de arquivos, o Sistema A, em algum momento exporta um arquivo em um formato qualquer, e um agente importador, automatizado ou não, importa este arquivo para o Sistema B, conforme ilustrado na figura 3.1.

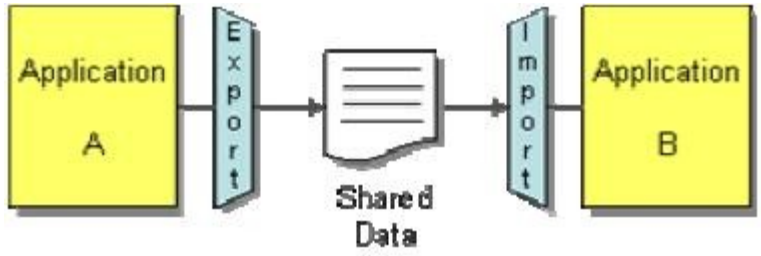

Figura 3.1: Troca de arquivos.

A grande vantagem desse estilo de integração é que os sistemas que participam do processo não precisam ter absolutamente nenhum conhecimento um do outro. No cenário descrito no parágrafo anterior, se o Sistema A for modificado ou até mesmo substituído, o Sistema B não ficará ciente destas mudanças, pois o único elo entre eles é o arquivo.

Este estilo de integração tem algumas desvantagens que podem ser decisivas para o sucesso de uma integração. Uma delas é o formato do arquivo que será compartilhado. Na maioria das vezes ele não é padronizado e cada sistema tem o seu próprio, então se um sistema for receber arquivos de vários outros sistemas, ele terá que ser capaz de manipular cada um desses formatos.

Outro problema é a freqüência em que ocorre a transferência dos arquivos. Se os sistemas precisam estar sincronizados a todo momento, este é um estilo que não deve ser adotado, existem outros mais adequados que serão explicados a seguir.

3.1.2 Bancos de Dados Compartilhados

A utilização de banco de dados compartilhados também é uma maneira simples de realizar a integração entre sistemas, pois os bancos de dados, em sua maioria, baseiam-

se no esquema relacional e utilizam a linguagem SQL para a manipulação dos dados, além disso, praticamente todas as plataformas são compatíveis com SQL[EIP2004] .

Neste estilo de integração, dois ou mais sistemas se integram realizando suas operações em uma base de dados comum, conforme ilustrado na figura 3.2. Como a comunicação com a base de dados é relativamente rápida se comparado com a troca de arquivos, elimina-se assim o problema da sincronização entre os sistemas.

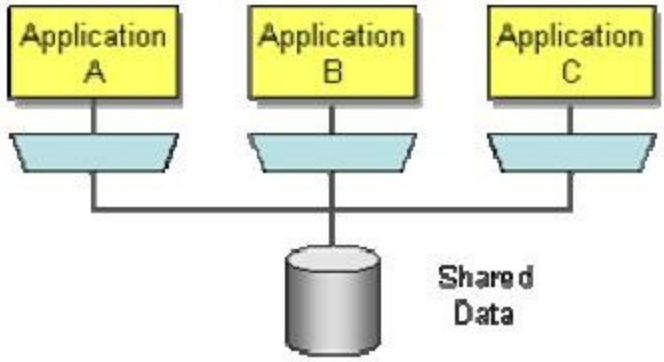

Figura 3.2: Banco de dados compartilhados.

As vantagens deste estilo de integração são a fácil implementação, já que quase todas as plataformas e linguagens de programação suportam SQL, e o sincronismo constante entre os sistemas, pois os seus estados atuais estarão armazenados em uma base de dados comum.

Porém, o compartilhamento de uma base de dados pode trazer algumas desvantagens, sendo uma delas a semântica contida nas tabelas e seus relacionamentos, já que todos os sistemas serão forçados a utilizar um mesmo esquema de banco de dados, o que nem sempre é desejado.

Outra desvantagem é que muitos sistemas lendo e escrevendo dados simultaneamente em um único local podem acabar transformando o banco de dados em um gargalo para suas performances.

Também deve-se levar em consideração que nesse estilo os sistemas ficam fortemente acoplados, devido ao compartilhamento da base de dados, e por isso mudanças na estrutura dela não são bem vindas, pois podem acarretar mudanças em todos os sistemas que a utilizam.

3.1.3 Invocação Remota de Procedimento

A invocação remota de procedimento ou chamada remota de procedimento, como

também é conhecida, é uma alternativa interessante a ser utilizada quando a simples troca de dados entre sistemas não é o suficiente e existe a necessidade de disparar algum procedimento no sistema destino[EIP2004] .

Nesse estilo de integração, a aplicação que deseja invocar uma operação faz uso de um objeto chamado *Stub*. O *Stub* funciona como uma representação local ou proxy remoto do sistema destino[RMI2004] e nele fica encapsulada toda a complexidade de acesso aos procedimentos desejados, como regras de nível de rede e serialização de objetos. Já na aplicação que disponibiliza o procedimento, a estrutura se encontra é o *Skeleton. O Skeleton é o* encarregado de receber as requisições dos Stubs e despachá-las para seu devido destino[RMI2004], também cabe a ele disponibilizar os procedimentos que serão passíveis de acesso remoto.

A idéia principal deste estilo é que se tenha um sistema destino o qual expõe sua interface com os procedimentos que estarão disponíveis para sistemas terceiros invocarem, como ilustrado na figura 3.3. Esta interface também descreve quais dados devem ser enviados e quais retornarão do procedimento. Os sistemas que desejam fazer requisições se utilizam dessa interface para saber como invocar os procedimentos, porém não ficam cientes de como eles foram implementados, pois estes dados ficam encapsulados no sistema destino.

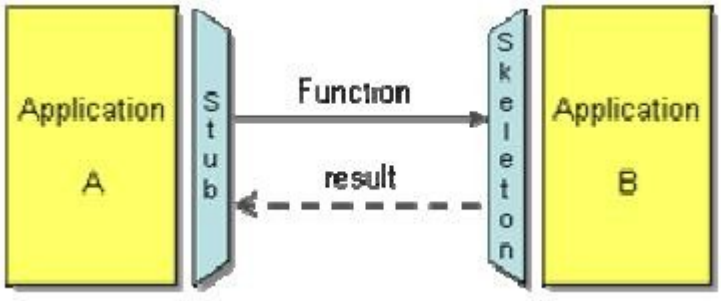

Figura 3.3: Invocação remota de procedimento.

A grande vantagem deste estilo, como já foi citado anteriormente, é o encapsulamento dos dados, que diminui o acoplamento entre as aplicações principalmente com a redução do compartilhamento de grandes estruturas de dados, como acontecia com o uso de bancos de dados compartilhados.

A utilização de uma interface com os serviços disponíveis também é considerada uma vantagem, pois ela especifica um tipo de contrato que deve ser obedecido tanto pelo sistema que está expondo seus procedimentos, quanto pelos sistemas que desejam fazer as requisições. Portanto qualquer alteração interna nesses sistemas não serão sentidas

no processo da integração, uma vez que ambos os lados estejam obedecendo as regras da interface.

Uma desvantagem da invocação remota de procedimento é a diferença de tempo que pode ocorrer entre uma chamada realizada em uma rede local e uma chamada remota, utilizando a Internet, por exemplo. A disponibilidade desta rede também deve ser levada em conta, pois na maioria das vezes não temos controle sobre o macro ambiente.

3.1.4 Troca de Mensagens

Algumas vezes o que se deseja é transferir dados entre as aplicações, como é feito na transferência de arquivos, porém com a garantia de que os sistemas estejam sincronizados a todo momento, e sem nenhum acoplamento entre os sistemas participantes, para este cenário pode-se utilizar o estilo de trocas de mensagens[EIP2004].

A troca de mensagens se baseia na utilização de mensagens – pequenos pacotes de dados – para a comunicação síncrona ou assíncrona entre os sistemas que participam deste processo, que é ilustrado na figura 3.4.

Em uma integração utilizando este estilo, existem os sistemas produtores de mensagens, os quais exportam os seus dados, e os sistemas consumidores de mensagens, os quais importam os dados produzidos por outros sistemas.

Utilizando a invocação remota de procedimento tem-se um certo acoplamento entre o sistemas, pois se utiliza o conceito de interfaces. Já com a troca de mensagens este acoplamento é totalmente eliminado, pois os sistemas que vão se integrar não precisam saber da existência um do outro, exatamente como acontece com a troca de arquivos, e com a vantagem de não ter a latência exacerbada da mesma. Isso acontece devido à existência de um terceiro elemento no ambiente, o qual fica responsável por todo o tratamento das mensagens, desde o recebimento dela de um sistema produtor, até a entrega da mensagem aos seus destinos finais. Estes artefatos são conhecidos como *Messages Brokers*.

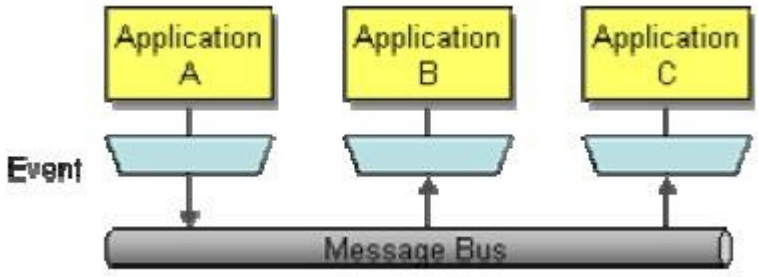

Figura 3.4: Troca de mensagens.

As vantagens deste estilo são inúmeras. Dentre as mais importantes estão o desacoplamento total entre as aplicações; a não necessidade de todos os sistemas que fazem parte da integração estarem rodando ao mesmo tempo, como acontece na invocação remota de procedimento, pois as mensagens podem ficar no Message Broker até que seja entregue aos destinos; a garantia de entrega da mensagem aos seus destinos; a possibilidade de uma comunicação assíncrona confiável, dentre outras que serão abordadas futuramente.

Existem algumas dificuldades para a adoção deste estilo, como a escolha correta do *Message Broker*, a escolha do formato adequado para as mensagens que serão utilizadas na comunicação, dentre outras decisões que devem ser tomados no momento do projeto do sistema[EIP2004].

Tendo em vista que a maioria dos *Enterprise Service Bus* que existem atualmente no mercado são baseados neste estilo, o presente trabalho irá tratar mais detalhadamente desse assunto nos próximos itens.

3.2 Sistemas de Troca de Mensagens

Esta seção tem por objetivo introduzir alguns dos conceitos mais básicos para o entendimento dos sistemas de troca de mensagens, descrever os problemas mais freqüentes que são encontrados durante o processo de construção de mensagens e propor soluções para resolvê-los.

Basicamente, a tecnologia de troca de mensagens se baseia em aplicações produtoras criando mensagens e enviando-as a um canal de mensagens, que são peças fundamentais em um sistema de troca de mensagens, são criados pelos desenvolvedores de acordo com suas necessidades. Por fim, as aplicações consumidoras se registram neste canal e ficam aguardando as mensagens serem entregues a elas. O processo de troca de mensagens é ilustrado na figura 3.5.

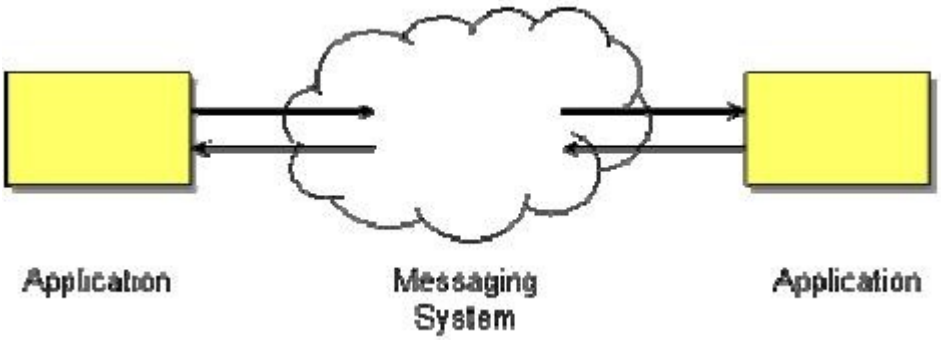

Figura 3.5: Processo de troca de mensagens.

3.2.1 Produtores e Consumidores

No contexto da troca de mensagens, pode-se classificar dois sujeitos ativos que participam de uma comunicação, os produtores e os consumidores.

A idéia é bastante simples e amplamente conhecida. Os produtores são responsáveis pela criação da mensagem propriamente dita e pelo seu envio ao canal de mensagem específico. Já os consumidores são responsáveis pela recepção das mensagens e realizar o devido processamento sobre essas mensagens[JMS2000].

3.2.2 Canais de Mensagens

Para realizar a ponte de ligação entre os produtores e consumidores, os sistemas de troca de mensagens utilizam o conceito de canais de mensagem.

Os canais de mensagens, conforme mostrado na figura 3.6, funcionam como encanamentos lógicos que ligam uma um sistema ao outro, ou diferentes partes de um mesmo sistema. Funcionam da seguinte maneira: o produtor envia uma mensagem ao canal, e na outra ponta existirá um ou mais consumidores que aguardam que mensagens serem enviadas ao canal para consumi-las. Com um canal de mensagem, as aplicações que fazem parte da integração ficam totalmente desacopladas umas das outras, bastando apenas estarem registradas no sistema de troca de mensagens e ouvindo o mesmo canal para conseguirem realizar a troca de informações[EIP2004].

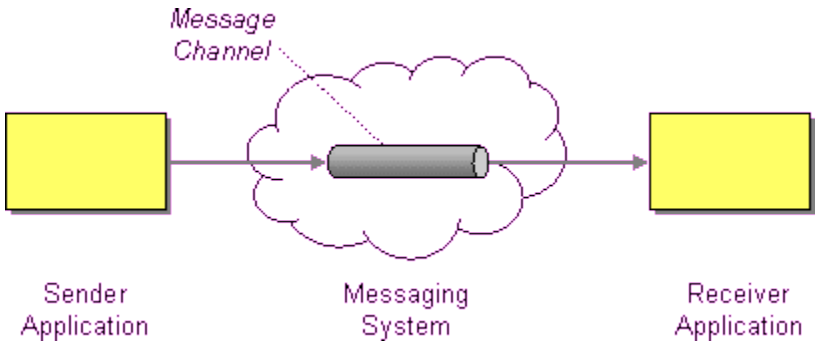

Figura 3.6: Canal de mensagens.

3.2.3 Construção de Mensagens

Esta é uma das etapas mais críticas no projeto de um sistema que utilize a troca de mensagens, pois envolve a elaboração da estrutura das mensagens que serão trafegadas pelos canais de mensagens. Um erro nesta fase pode causar um efeito totalmente indesejado onde os produtores e consumidores não se entendam durante a comunicação.

- **Qual é a intenção da mensagem?** - Mensagens são apenas pequenos pacotes que contém dados; porém, produtores podem precisar que os consumidores realizem diferentes tarefas ao receberem as mensagens. Podem desejar que o consumidor realize algum comando em seu sistema, enviando um tipo de mensagem conhecido como *Command Message;* podem desejar apenas enviar um dado seu a outro sistema, *Document Message;* ou podem, finalmente, desejar notificar os consumidores de que ocorreu alguma mudança no estado do sistema produtor, enviando mensagens do tipo *Event Message*[JMS2000].

- **A mensagem terá uma resposta? –** Muitas vezes ao enviar uma mensagem, o produtor pode desejar receber algum tipo de resposta, neste cenário, a maioria das mensagens enviadas serão do tipo *Command*, enquanto as respostas serão do tipo *Document*, contendo informações relativas à execução do comando enviado anteriormente[JMS2000].
- **Mensagens transportarão grandes massas de dados? -** Algumas vezes se deseja transportar grandes massas de dados entre as aplicações, neste caso, o melhor que o produtor tem a fazer é dividir estes dados em pedaços menores e enviá-los em seqüência ao canal de mensagens.
- **A mensagem deve expirar? -** O tempo que uma mensagem leva para ser recebida pelos seus consumidores pode ser um fator importante em

aplicações que utilizem este estilo de integração, porém os sistemas de troca de mensagens não podem oferecer muitas garantias sobre o tempo total que a mensagem levará para atingir seu destino final. Para resolver este problema pode-se utilizar o conceito de *dead-line*, onde tanto o sistema de troca de mensagens, quanto o consumidor da mensagem podem ignorá-la caso ela tenha ultrapassado o seu tempo de vida[JMS2000].

3.2.4 Considerações Finais

Nesta seção foram apresentados alguns conceitos básicos para o entendimento dos sistemas de mensagens. Outros conceitos mais avançados como roteamento e transformação de mensagens serão abordados no capitulo que trata sobre os *Enterprise Services Buses*.

3.3 Alguns Padrões para Integração Corporativa

Esta seção tem por objetivo apresentar alguns dos padrões arquiteturais mais utilizados no que se trata de integração de sistemas com ESB. O projeto que será criado com este trabalho vai procurar fazer uso exaustivo destes padrões, uma vez que são amplamente difundidos e já foram testados por grandes arquitetos da área.

3.3.1 VETO

O padrão VETO – acrônimo para Validação, Enriquecimento, Transformação e Operação – é o mais utilizado por corporações que fazem uso de um ESB, com ele é possível garantir que as mensagens que trafegam pelo barramento estarão sempre consistentes, graças a suas etapas de enriquecimento e validação. Este padrão é ilustrado na figura 3.7.

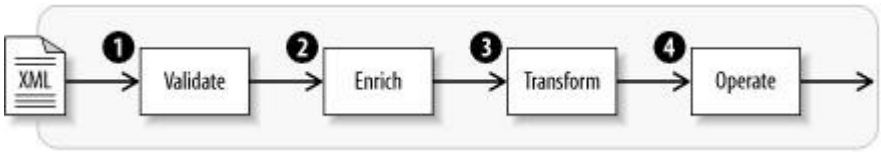

Figura 3.7: Padrão VETO.

3.3.1.1 Validação

A etapa de validação,ilustrada na figura 3.8, é, geralmente, a primeira a ser executada em um processo ESB e pode ser realizada de diferentes formas. É importante que esta etapa aconteça independentemente das outras, pois remove a sobrecarga de

validação nas etapas subseqüentes [ESB2004].

Para a implementação desta etapa, pode-se utilizar regras de validação de esquema XML e WSDL ou, caso a mensagem não seja no formato XML, uma regra customizada de validação deve ser criada.

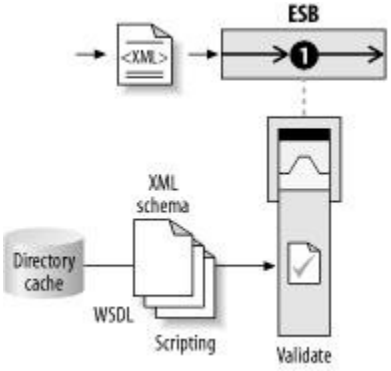

Figura 3.8: Etapa de validação do padrão VETO.

3.3.1.2 Enriquecimento

A etapa de enriquecimento consiste na adição de dados necessários para tornar a mensagem mais significativa para o seu receptor. Muitas vezes é necessário realizar algum complemento na mensagem que está trafegando, como adicionar dados de cabeçalho, meta-informações ou até mesmo alterar o seu corpo com algum dado oriundo de um serviço de enriquecimento ou banco de dados. É nesta etapa que estas alterações devem ser realizadas.

3.3.1.3 Transformação

Nesta etapa ocorre a transformação da mensagem para o formato esperado pelo sistema destino. É importante lembrar que o destino pode ter suas próprias regras de validação para decidir se aceita ou não a mensagem, então, para evitar que a mensagem seja rejeitada, após a etapa de transformação pode ser executada mais uma etapa de validação visando garantir que a mensagem transformada esteja em conformidade com as regras que o sistema destino exige. A etapa de transformação é ilustrada na figura 3.9.

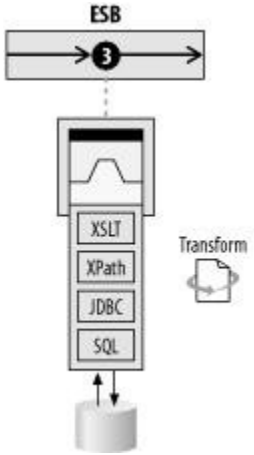

Figura 3.9: Etapa de transformação do padrão VETO.

3.3.1.4 Operação

A etapa de operação envolve a invocação do serviço destino ou alguma outra maneira de interação com a aplicação destino[ESB2004]. É fundamental nesta etapa que todas as etapas anteriores tenham sido executadas corretamente, para que não haja rejeição da mensagem por parte do sistema destino.

Com o uso do padrão VETO é possível implementar módulos separados que podem ser reutilizados por todo o sistema, porém o uso deste padrão, como foi concebido, não prevê nenhuma forma de interação mais sofisticada, como, por exemplo, a orquestração de serviços. Entretanto, existem diversas variações deste padrão para as mais diversas necessidades, e a variação VETRO é uma alternativa que pode ser utilizada para preencher esta lacuna.

3.3.2 VETRO

O padrão VETRO, ilustrado na figura 3.10, é uma variação do VETO que adiciona uma etapa de Roteamento antes da etapa da Operação. Com a adição desta etapa é possível elaborar um fluxo mais complexo que pode fazer uma busca inteligente e decidir em tempo de execução quais serão os serviços invocados pela etapa subseqüente.

Várias estratégias podem ser utilizadas para implementar a etapa de roteamento, sendo que a mais comum é o roteamento baseado em conteúdo, onde o serviço a ser invocado é descoberto a partir do conteúdo da mensagem que está trafegando no barramento.

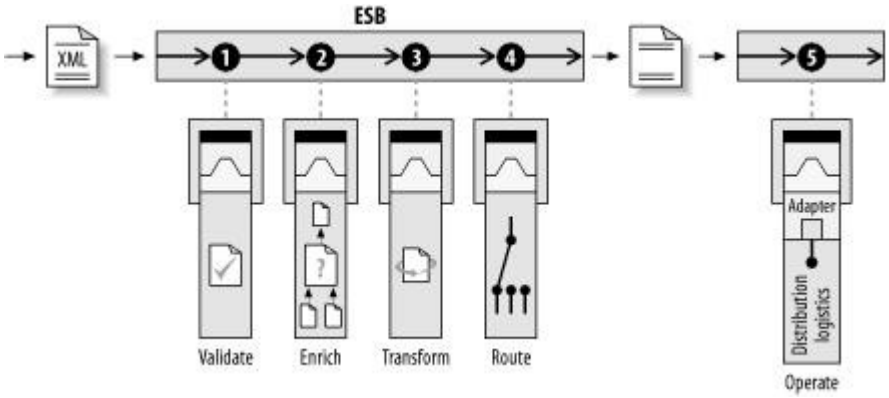

Figura 3.10: Padrão VETRO.

3.3.3 Two-Step XRef

Existem casos em que a integração de dados através de diferentes sistemas exigem tanto a transformação da estrutura da mensagem quanto a alteração de seu conteúdo[ESB2004]. Nessas situações, a simples criação de uma estrutura monolítica para a transformação do conteúdo pode não ser a melhor opção, pois acaba criando uma baixa coesão para estrutura, que além de ter que modificar o formato, terá também que modificar o conteúdo da mensagem.

Uma alternativa mais elegante para resolver este problema é a utilização do padrão Two-Step XRef, ilustrado na figura 3.11, que separa em duas partes a etapa de validação: uma parte fica responsável pela alteração da estrutura da mensagem, e a outra responsável por substituir seus dados.

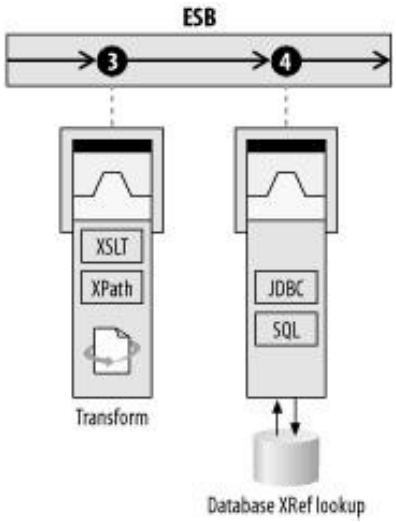

Figura 3.11: Padrão Two-Step XRef.

Com o uso deste padrão é possível observar uma série de vantagens para a arquitetura e funcionamento do sistema como um todo, como o aumento do nível de reuso

do serviços, já que serviços genéricos de transformação podem ser criados para modificar a estrutura, principalmente nas transformações XML-para-XML, e outros serviços podem ser criados para a busca de dados em bases de dados ou caches para as possíveis substituições de conteúdo.

Outra vantagem conseguida com o uso do Two-step XRef é a sua capacidade de ser “plugado” ao padrão VETO na etapa de transformação. Com este casamento, é possível elaborar um fluxo robusto para a mensagem e ao mesmo tempo ter um serviço de transformação bastante otimizado.

3.4 Considerações Finais

Este capítulo procurou dar uma visão geral sobre os padrões para a integração de sistemas corporativos, dando ênfase aos sistemas de mensagens, pois são estes os utilizados pela maioria dos ESB's presentes no mercado.

Foram mostrados também os padrões mais famosos utilizados pelos arquitetos de software para a integração de sistemas com ESB. O projeto que será criado neste trabalho fará uso dos conceitos vistos neste capítulo, também utilizará alguns conceitos sobre os *Enterprise Service Bus* que serão vistos no capítulo a seguir.

4 Frameworks

4.1 Conceito

Segundo a definição de Johnson [JOH1988] um *framework* é um conjunto de classes que incorpora um projeto que tem como foco solucionar uma gama de problemas relacionados. Ainda em outra definição, *frameworks* são um conjunto de classes relacionadas que criam uma estrutura reusável para um tipo específico de *software* [LAN1995]. Um *framework* é um conjunto de blocos de software pré fabricado que os programadores podem usar, estender, ou customizar para soluções computacionais específicas [TAL1994].

A razão pela qual mais empresas procuram utilizar *frameworks* nos dias atuais é evidente: a economia de tempo é visada em todo o projeto, e a qualidade do software é sempre importante. Não vale a pena “reinventar a roda” cada vez que for necessário utilizá-la e ainda correr risco de desenvolvê-la com falhas.

Um *framework* oferece muito mais do que recursos para o desenvolvedor, pois fornece uma arquitetura, uma plataforma, e geralmente faz uso de padrões estabelecidos no mercado, tudo isso para facilitar a integração e a sua extensão para quaisquer que sejam as necessidades do usuário. Um desenvolvedor que quer derivar uma nova classe para customizar o *framework* irá escrever seu código seguindo um padrão definido pela superclasse abstrata. O *framework* irá invocar os métodos da classe desenvolvida, e não o contrário: esta regra caracteriza o princípio de Hollywood, que é conhecido pela frase “*dont call us, we´ll call you*”, que significa “não nos chame, nós o chamaremos”.

4.2 Utilização de um Framework

Na utilização de um *framework* podem-se citar diversas vantagens:

- Aplicação mais atual: Quando se desenvolve utilizando versões atuais de *frameworks* garantimos que o projeto que está em desenvolvimento terá as tecnologias mais atuais do momento. É comum que durante a implantação de uma aplicação nova, surjam outras novas tecnologias que se mostrem interessantes para a mesma. Os *frameworks* reduzem este “período de desatualização”.
- Manutenção: Manutenção é a parte mais cara dos projetos. Quando se realiza manutenção de diversas aplicações que fazem uso de um mesmo *framework*, as mudanças necessitam ser feitas apenas em um lugar, e não em

diversos lugares, poupando tempo e garantindo que o erro foi corrigido em todas as aplicações.

- Confiança: Assim como qualquer outro software, um *framework* pode conter *bugs* e erros, mas como o seu principio é ser reusado por mais de uma aplicação, a tendência é que estes *bugs* e erros sejam descobertos mais cedo. Logo, em um espaço mais curto de tempo se obterá um *software* mais seguro e confiável.
- Padrões: Os *frameworks* têm como princípio fazer uso de padrões estabelecidos no mercado, que têm como grande vantagem serem de mais fácil acesso, suporte e atualização. Isso facilita na integração com a maioria das aplicações que queiram usar o *framework*.

Apesar destas vantagens, existem algumas dificuldades na utilização e desenvolvimento de *frameworks*. Existe certo aumento no tempo de implementação de um *framework*, porque todas as estruturas e a arquitetura como um todo do *software* deve ser pensada para ser reusável, e poder ser reaproveitada pelos outros usuários. Não é qualquer estrutura que atende este quesito. Em compensação, este tempo de desenvolvimento a mais é compensado. Uma estrutura completamente personalizada e compatível com os propósitos de projeto da empresa renderá muito mais ganho de tempo posteriormente no desenvolvimento das aplicações que farão uso deste *framework*. A utilização de um *framework* em geral é simples. Talvez a maior dificuldade seja encontrar um *framework* que atenda melhor as suas necessidades, de maneira compreensível, visto que atualmente existem diversas opções de *frameworks* para diversas áreas de software, sejam eles de código aberto ou fechado.

4.3 Classificações

Segundo a publicação *Building object-oriented frameworks,* feita por Taligent Inc., podemos classificar *frameworks* em três tipos de acordo com o tipo de problema que eles resolvem:

- *Frameworks* de aplicação: encapsulam funcionalidades aplicáveis para uma larga variedade de programas. Estes *frameworks* possuem funcionalidades específicas como componentes de interface, que podem ser aplicadas em diferentes softwares, não relacionados com o domínio de problema que o programa está destinado a solucionar.
- *Frameworks* de domínio: encapsulam funcionalidades para a resolução de um problema em particular. Os recursos disponíveis neste tipo de *framework* são

voltados diretamente para a área que ele abrange, não sendo aplicáveis a softwares de qualquer tipo. Por exemplo, um *framework* para controle de manufatura, ou até mesmo, como no caso deste trabalho, para a criação de um ESB.

- *Frameworks* de suporte fornecem serviços em nível de sistema que ajudam o desenvolvedor em tarefas como acesso de arquivos, computação distribuída e *drivers* de dispositivos.

5 ENTERPRISE SERVICE BUS

5.1 O que é?

O conceito de *Enterprise Service Bus* surgiu frente ao grande aumento da necessidade de integração dentro das empresas. As empresas, no inicio do novo milênio, se viram dotadas de diversos sistemas cada um com sua função de negócio e desenvolvidos para ambientes, e até por *software houses* diferentes. Houve a necessidade de integrar estas aplicações a fim de agilizar e facilitar processos dentro da organização. Alguns conceitos como SOA (*Service Oriented Architecture*), EAI (*Enterprise Application Integration*) e B2B (*Business-to-Business*) surgiram para suprir estas necessidades, e logo em seguida veio o ESB, que tenta unir todas as qualidades de cada uma destes conceitos em uma só coisa.

Enterprise Service Bus é uma plataforma de integração que combina troca de mensagens, *Web Services*, transformação de dados e roteamento inteligente para conectar e coordenar a integração de um número significativo de diferentes aplicações de uma organização com integridade transacional [ESB2004].

Desta maneira, um ESB não se constitui apenas da idéia de realizar a integração entre diferentes sistemas. Um ESB tem que ser capaz de gerenciar e disponibilizar os recursos das aplicações envoltas no barramento com segurança, capaz de atravessar fronteiras físicas e lógicas de uma empresa para garantir a integração.

No desenvolvimento deste tipo de aplicação devem ser utilizados padrões estabelecidos, como JMS, *Web Services*, JCA ou COM. A regra é fazer o uso de padrões que tenham compatibilidade entre plataformas e aplicações heterogêneas.

Distribuição também faz parte do conceito de ESB. Geralmente sistemas de integração são centralizados, mas neste caso deve-se dividir a aplicação fazendo uso de uma arquitetura como o SOA. Dividindo o sistema em serviços individuais é possível proporcionar esta distribuição e cooperação entre diferentes serviços.

Os serviços devem ser unidades feitas sem se pensar qual tipo de conexão será usada para acessá-los. Eles devem apenas receber mensagens como um evento, independente do meio de comunicação, garantindo assim a disponibilidade de qualquer serviço por qualquer meio de acesso e conseqüentemente para todas as aplicações que estão ligadas ao barramento.

Como podemos ver na figura 5.1, um ESB provê um barramento para conexão a diversas aplicações diferentes, no exemplo são ilustradas aplicações responsáveis por cada etapa da compra de um produto, desde sistemas que cuidam do cadastro de

produtos e clientes, até outras que realizam o processamento da compra. Os clientes destas aplicações apenas fazem a requisição ao ESB que realiza as validações, transformações, necessárias e repassa para o destinatário correto.

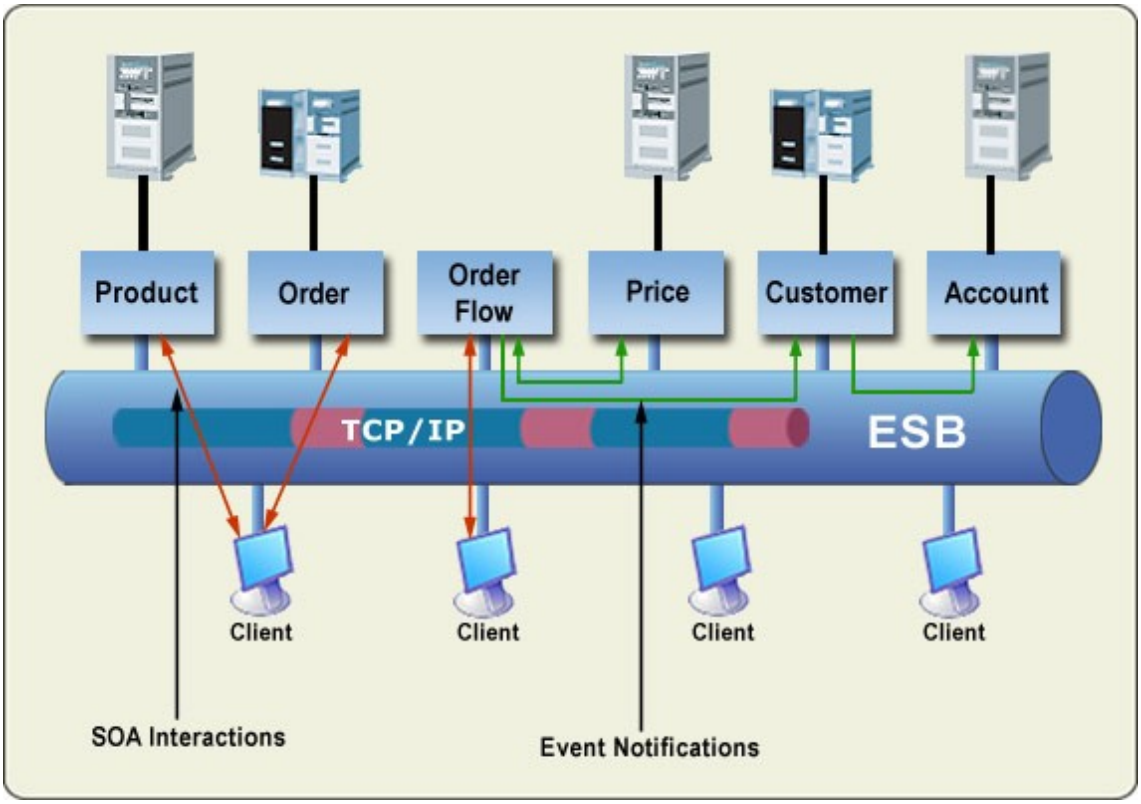

Figura 5.1: Ilustração de um ESB.

5.2 Message Oriented Middleware(MOM)

O MOM é uma parte chave de um ESB. Este conceito envolve a passagem dos dados entre as aplicações usando canais de comunicação que transportam unidades de informação(mensagens). A responsabilidade do MOM é levar as mensagens para o seu destino, os remetentes e os destinatários de mensagens não se conhecem, criando um ambiente de baixo acoplamento entre as aplicações. Usando este tipo de *middleware* fugimos de implementações de comunicação entre sistemas que podem causar muitos problemas. Ao invés de termos um sistema que abre canais de comunicação diretamente com diversas outras aplicações, todos os sistemas devem se comunicar apenas com o MOM, que faz parte do ESB. Este, por sua vez, será o responsável por levar a mensagem ao seu respectivo destino. Aumenta-se a dependência das aplicações com o ESB, mas obtém-se muito mais organização, tirando preocupações de comunicação e transformação de mensagens das aplicações, e passando este papel para o ESB. Basicamente, o MOM cria a abstração de um barramento em um *Enterprise Service Bus*.

A grande conquista ao se utilizar um MOM está na redução das interfaces de comunicação entre as aplicações. Tomemos por exemplo um ambiente pequeno com 3 aplicações, cada qual com um serviço a ser disponibilizado na rede, para que cada aplicação consiga acessar as outras duas aplicações serão necessárias 3 interfaces.

Se aumentarmos o numero de aplicações para 5 as interfaces sobem para 10, aumentando a 13 sistemas já serão 78 as interfaces a serem construídas. Um MOM faz com que este número seja muito menor e cresça de maneira linear com o acréscimo de novas aplicações ao ambiente, visto que cada aplicação deve apenas se comunicar com um barramento único ao invés de múltiplas fontes de serviços.

5.3 Service Container e Endpoints

Conforme descrito anteriormente, podemos generalizar a definição de um ESB e dizer que eles são compostos por quatro elementos: *Message Oriented Middleware*, *Web Services*, roteamento inteligente baseado no conteúdo da mensagens, e transformação de XML e dados. Um ESB provê uma metodologia de integração altamente distribuída e de baixo acoplamento. Mas como estes componentes trabalham em conjunto, considerando que devemos ter distribuição? Três componentes contribuem para que isso seja possível, são eles os *Abstract Endpoints*, o MOM e um *service container* distribuído de baixo peso.

5.4 Abstract Endpoints

Esta expressão de origem inglesa não tem nenhuma tradução muito elegante na nossa língua, mas seria algo como “pontos finais abstratos’.

Basicamente do ponto de vista da integração todas as aplicações e serviços em um ESB são *endpoints* abstratos. Portanto é algo bem genérico. Um *endpoint* pode representar um serviço especializado de cálculo de vendas ou uma chamada para um *Web Service* externo [ESB2004].

Esta abstração permite que o arquiteto da integração use ferramentas de maior nível para relacionar *endpoints* de serviços em fluxo de processos. Consideraremos então um *endpoint* todo e qualquer componente que faça parte do fluxo de mensagens, seja um sistema que está conectado ao barramento, ou qualquer serviço disponível no ESB, como transformações de dados.

5.5 Service Container

Um *service container* é a manifestação física de um a*bstract endpoint*, ou seja,

trata-se da implementação de uma interface de um serviço. Logo, podemos dizer ele é um processo remoto que pode hospedar componentes de software.

Como vimos na definição de *abstract endpoint*, o *service container* nada mais é do que a parte física de um *endpoint*. Então um ESB é formado por vários *endpoints* abstratos, no ponto de vista de um arquiteto de integração, e estas representações, do ponto de vista da implementação, serão em *services containers*.

5.6 Invocação de serviços no ESB

A invocação de serviços em um ESB deve se dar de uma maneira mais dinâmica, de modo que cada mensagem é tratada como um evento dentro do fluxo de processo. Um ESB deve conter registros como um diretório de serviços onde se guarda informações dos *endpoints* (destinatários) destes serviços. A ação de “procurar, conectar e invocar” é separada da parte de negócio/lógica dos serviços. Em um ESB assíncrono as mensagens são enviadas para uma fila de onde são enviadas aos seus destinatários e obtidas as suas respostas de processamento.

O código de implementação de um serviço é focado apenas na parte lógica e de negócio estabelecida, processa-se a mensagem recebida pelo *endpoint* de entrada gerenciada por um *service container*. Quando o serviço terminar sua tarefa ele retorna a resposta e esta é roteada para um próximo passo(caso haja mais de um passo de processamento de mensagens) ou para elemento remetente da mensagem de entrada. A operação de localizar e rotear as mensagens para o próximo serviço do fluxo é feita pelo próprio ESB e será explicada no tópico a seguir.

5.7 Roteamento de Mensagens

No modelo de ESB existem duas maneiras de se rotear as mensagens: por um itinerário e através do seu conteúdo. Estas modalidades serão explicadas a seguir

5.7.1 Roteamento de Mensagem Baseado em Itinerário

Esta é a maneira mais simples de se rotear uma mensagem. Imagine que um arquiteto de integração estabeleça uma ordem de serviços que uma mensagem passará em um fluxo, como por exemplo uma compra: verificação de crédito; checagem de estoque; preenchimento da compra; e solicitação de compra. Cada uma destas etapas pode se traduzir em um serviço, ou em um *endpoint* para o ESB. Em cada etapa que a requisição de compra passar, os serviços irão validar seu conteúdo e, caso esteja tudo certo, repassar para o próximo passo até obter-se um erro (então a compra não será

solicitada) ou atingir o fim do fluxo (a compra será solicitada). Repare que o fluxo de mensagem é algo não muito dinâmico, pois já se sabe exatamente quais serão as etapas a serem percorridas e inclusive onde se encontra o serviço responsável pelo processamento de cada etapa, por isso podemos dizer que este é um roteamento baseado em itinerário. Assim como um ônibus de linha onde se tem um percurso pré-estabelecido por onde o ônibus passará e recolherá passageiros até atingir seu ponto final.

5.7.2 Roteamento de Mensagens Baseado no Conteúdo

Podemos considerar este tipo de roteamento de mensagens mais dinâmico que o outro visto que o próximo passo a ser invocado dentro de um fluxo depende de informações contidas dentro da mensagem, e o ESB através de serviços de regras, filtros, estabelecerá qual será o próximo destino da mensagem. Por exemplo: o ESB, ao receber uma mensagem XML, a passa por um filtro de entrada (serviço de regra) no qual estarão implementadas as regras que ditarão seu próximo destino. Caso ela seja uma mensagem SOAP inválida, a encaminha para um fluxo de erro ou exceção; caso perceba que a mensagem pertence a determinado *namespace,* a encaminha para um outro serviço, que pode ser outro serviço, outro *endpoint*, ou até mesmo um W*ebservice* remoto.

Existem casos em que se verifica algum dado identificador do destinatário dentro da mensagem enviada na invocação. Um exemplo simples seria uma rede de lojas que possui diversos estabelecimentos. O gerente de vendas da rede envia uma requisição para o ESB(de maneira transparente) requisitando informações de venda da unidade de Florianópolis. O ESB relaciona a identificação da loja com o respectivo diretório cadastrado da unidade no sistema e invoca um W*ebservice* que está localizado no sistema local desta loja requisitando os relatórios desejados pelo gerente de vendas.

As mensagens podem variar de formato, e isso pode ser um quesito que motive o roteamento dela para outros fluxos ou serviços. Ainda é muito comum nas empresas sistemas legados que utilizam formato de arquivos de texto para trafegar informações pela rede, e o filtro de roteamento pode verificar se a mensagem é do tipo XML ou texto plano e encaminhar para diferentes serviços que tratam estas duas mensagens, e quando temos divergências em formatos de mensagens pode ocorrer a necessidade de transformações desta mensagem para que esta possa ser aceita por outros serviços ou receptores externos dentro do ESB, trataremos deste assunto no próximo item.

5.8 Transformação de Mensagens

Quando temos um ambiente heterogêneo dentro de uma empresa onde sistemas diferentes foram desenvolvidos em diferentes plataformas ou por diferentes fabricantes é comum que tenhamos divergência no formato de dados destes sistemas. Um caso muito comum são organizações que utilizam sistemas legados que fazem uso de antigos arquivos de texto para comunicação, e que pela sua eficiência e confiabilidade não substituem estes sistemas por novas implementações que utilizem novas estruturas de dados como o próprio XML. Mas muitas vezes surge a necessidade que um sistema legado se comunique com algum sistema mais atual que faz uso do XML, e este seria um exemplo de caso em que o ESB poderia transformar a mensagem de entrada para um padrão compatível com o sistema destinatário.

Em alguns casos, mesmo quando temos estruturas de dados do mesmo tipo, se faz necessária a transformação destes dados. Um sistema de cadastro de clientes tem diferente relacionamento ao objeto “Cliente” comparado a um sistema de gerenciamento de relacionamento com clientes. Logo, o ESB, ao estabelecer ligação entre estes dois sistemas, deve modificar os dados do cliente para suprir as necessidades de dados de cada aplicação, seja enriquecendo a mensagem com informações não presentes na concepção de “cliente” de uma das aplicações, ou removendo informações para que seja compreendida pela aplicação destino.

Estas transformações são implementadas como um serviço dentro do fluxo da mensagem no ESB, tendo a mesma facilidade de utilização e implementação dos outros componentes do barramento. A figura 5.2 ilustra uma mensagem sendo transformada.

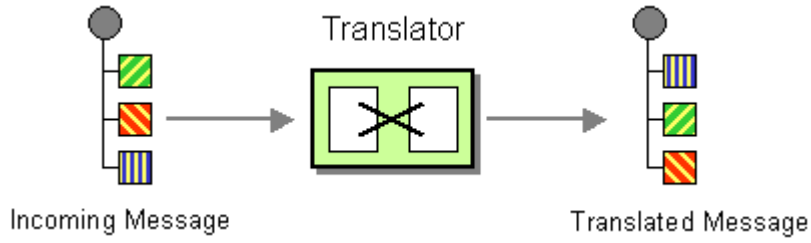

Figura 5.2: Tradução de uma mensagem em diferentes formatos.

5.9 Mule ESB

O Mule é um *framework* desenvolvido na linguagem Java, que foi produzido se pensando em implementar os conceitos de um ESB sob a licença CPAL (*Common Public Attribution License*), caracterizada por ter código aberto. A ideia é proporcionar a integração fazendo uso de protocolos de comunicação como JMS, TCP, HTTP, E-mail

fazendo a interação entre elas todas. Trata-se de um *framework* altamente escalável, podendo ser usado por poucas aplicações de maneira mais simples e ao mesmo tempo sendo robusto para a utilização mais massiva de acordo com as necessidades do usuário. Este *software* é capaz de gerenciar todas as interações feitas entre as aplicações e componentes da camada negócio de maneira transparente, exatamente como o conceito de ESB propõe. Pelo fato de ser bem conceituado e ser aberto para a comunidade decidiu-se usa-lo para a implementação do projeto descrito neste documento, a fim de demonstrarmos na prática a composição e o funcionamento de um ESB.

5.9.1 Funcionamento do Mule ESB

A ideia geral do Mule é que ele possa proporcionar ao usuário a separação da lógica de negócio da lógica de tratamento e manuseio de mensagens, deixando esta ultima parte para o *framework* gerenciar. Uma das grandes vantagens do Mule é que ele é capaz de manusear mensagens enviadas em uma grande variedade de tipos de protocolos [MUL2008]. Por isso ele é capaz de facilitar em muito a vida dos desenvolvedores de ESBs. Um exemplo simples: Uma solicitação de compra pode ser recebida através de JMS ou *Web Service*, como fazer para que o serviço disponível na sua aplicação consiga tratar diferentes formatos de mensagem? A resposta para esta pergunta é simples. Um serviço não deve ter nenhuma relação com o formato da requisição ou resposta. O Mule será o responsável por receber esta mensagem não importa por que meio ou formato que ela venha e transforma-la para que o serviço possa ser executado em cima das informações que foram enviadas.

Para que possamos fazer coisas como esta anteriormente citada utilizando o Mule é necessário conhecer um componente chave dentro do *framework*, o *endpoint*. *Enpoints* são elementos que devem ser especificados na parte de roteamento de entrada e saída dos fluxos que são definidos no Mule para dizer qual protocolo deve-se utilizar, para onde deve ser enviada a mensagem e quais mensagens um componente do serviço deve receber. A principal parte de um *endpoint* é a especificação da URL. Por ela determina-se o protocolo, a localização e possíveis parâmetros adicionais dependendo do protocolo.

Por exemplo, caso seja especificado um *enpoint* de entrada com a URL “http://onix.org/service” para um determinado serviço, o transporte http do Mule despachará mensagens para aquele serviço sempre que alguma mensagem for enviada para aquela URL. Da mesma maneira simples podem ser especificados uma diversidade de outros protocolos, como o de arquivos, que pode ser uma URL como “file://arquivos/upload”. Toda a vez que algum arquivo for colocado nesta pasta local será

iniciado um fluxo de mensagem para aquele serviço que contiver este *endpoint* especificado na sua entrada. Com a mesma facilidade pode-se especificar a saída da mensagem do serviço.

Além dos elementos de entrada e saída, outros componentes podem ser adicionados ao serviço, os quais cuidarão da lógica do negócio em si, verificar se os dados dentro das mensagens são válidos, buscar outras informações relativas ao conteúdo em bancos de dados, enriquecer os dados de alguma maneira. Este tipo de componente deve ser implementado pelo usuário do *framework*, por ser algo muito específico de cada sistema, e pode ser facilmente associado a qualquer serviço bastando fazer referência para esta implementação nas configurações do serviço.

Todas as configurações do Mule podem ser feitas através de um ou mais arquivos XML que conterão todas as informações dos serviços que se desejam criar. Estes arquivos de configuração devem ser especificados no "*start-up*" do *framework* e tudo será automaticamente carregado no servidor de aplicação.

5.9.2 Utilização do Mule ESB no Onix

A proposta de *framework* contida neste trabalho fará uso do Mule, mas não utilizará todos os recursos que ele dispõe. As principais funcionalidades aproveitadas serão as implementações dos protocolos de entrada e saída de mensagens e os *endpoints*. Protocolos são padrões já conhecidos pela comunidade, e não é o foco do trabalho desenvolver componentes de conectividade. O projeto será focado na especificação de um fluxo padrão de tratamento de mensagens que conterá validadores, enriquecedores e transformadores de mensagem de acordo com o padrão VETRO.

A estrutura do *framework*, batizado de Onix, fará o gerenciamento de fluxo por onde a mensagem deve passar e será completamente desenvolvida neste projeto, deixando a parte de entrada e saída de dados para aplicações externas como responsabilidade do Mule. Para fazer o controle de saída e entrada de mensagens serão criados *endpoints* de acordo com a especificação do usuário. Quando um *endpoint* for invocado através do envio de alguma mensagem, o Mule repassará esta mensagem para o Onix, que assumirá o controle a partir deste ponto, executando as operações configuradas pelo usuário e no final novamente passando a mensagem modificada para um *endpoint* de saída para que ela possa ser enviada para o receptor que foi declarado nas configurações.

6 Onix *Framework*

O Onix é um *framework* desenvolvido utilizando a linguagem de programação Java que tem por objetivo auxiliar engenheiros de software, analistas e programadores a desenvolverem soluções de integração de uma maneira mais rápida e robusta.

O *framework* pode ser considerado como uma implementação do padrão de integração VETRO, mostrado nos capítulos anteriores, e caberá aos desenvolvedores que o utilizarem, apenas estenderem as interfaces de validação, enriquecimento e transformação para adicionarem as funcionalidades desejadas ao fluxo de integração.

De acordo com a classificação de *frameworks* criada por Taligent Inc. [TAL1994], o Onix se enquadra na classe de *frameworks* de domínio, por só poder se utilizado por um domínio especifico, neste caso a integração de sistemas.

A figura 6.1 demonstra um ambiente de integração que utiliza o Onix *Framework*.

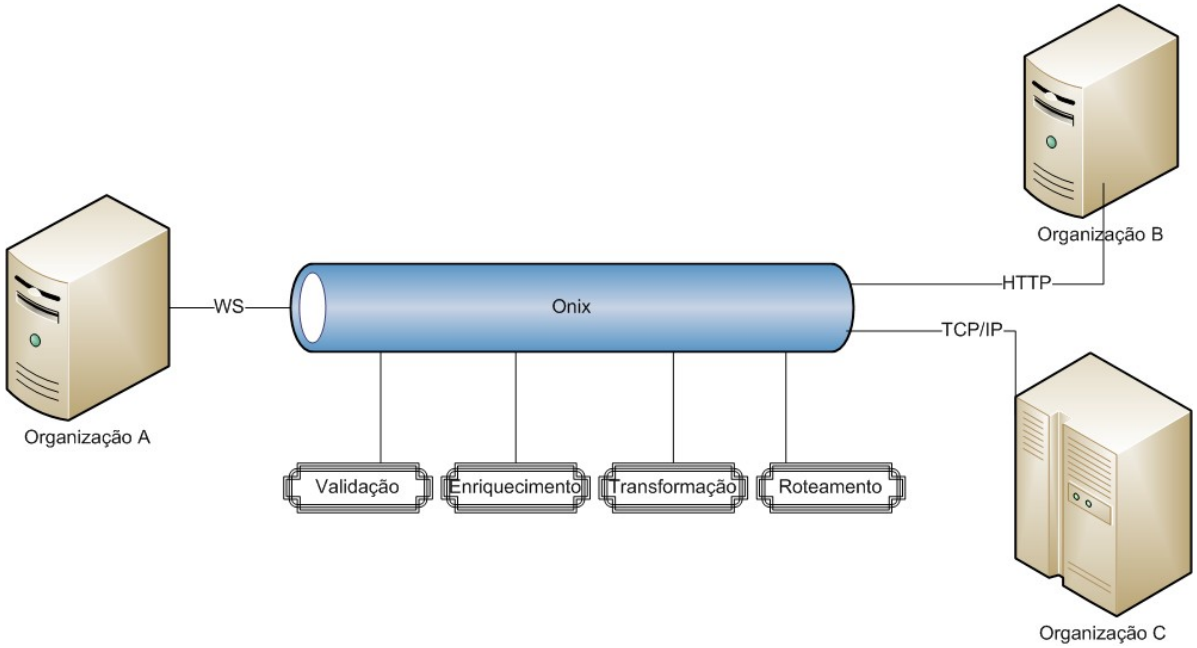

Figura 6.1: Ilustração de um ambiente de integração com o Onix.

6.1 Requisitos

O principal requisito deste *framework* de integração é facilitar a implementação da troca de informações entre sistemas diferentes. O Onix também deve levar em consideração aspectos de confiabilidade, desempenho e facilidades de uso e de configuração.

Abaixo segue uma lista mais detalhada com os requisitos levantados para a

primeira versão:

- **Configuração utilizando arquivos XML**: A configuração do *framework* deve se dar na forma de arquivos de configuração no formato XML para facilitar tanto a compreensão humana, quanto a validação do arquivo e a tradução do mesmo para objetos internos do *framework*.
- **Capacidade de executar as regras descritas no padrão VETRO:** O *framework* deve ser capaz de executar as cinco regras descritas no padrão VETRO – validação, enriquecimento, transformação, roteamento e operação – para todas as mensagens recebidas. As etapas de validação, enriquecimento e transformação são consideradas opcionais.
- **Capacidade de enviar uma mensagem para um único destinatário:** O Onix deve ser capaz de receber uma mensagem e enviá-la para um destinatário de acordo com as configurações em seu arquivo XML.
- **Capacidade de enviar uma mensagem para vários destinatários:** O Onix deve ser capaz de receber uma mensagem e enviá-la para vários destinatários simultaneamente, sem necessidade que seja retornada uma resposta destes destinatários.
- **Capacidade de trabalhar com diversos tipos de protocolos:** O *framework* deve ser capaz de aceitar conexões nos seguintes protocolos em sua primeira versão: TCP, HTTP, SOAP.
- **Capacidade de realizar a troca informações entre sistemas que utilizam protocolos diferentes:** O *framework* deve ser capaz de realizar a troca de informações entre sistemas que utilizam protocolos de comunicação diferentes. Por exemplo: um sistema X que utilize *sockets* para trocar informações poderá se comunicar com um sistema Y que utilize Web Services com o protocolo SOAP.
- **Capacidade de utilizar o ESB que desejar:** O Onix deve possuir uma arquitetura de alta coesão e baixo acoplamento que permita a utilização de qualquer ESB existente no mercado.

6.2 Arquitetura

A arquitetura proposta para o Onix possui três camadas de alta coesão, cada qual

com sua devida responsabilidade.

A primeira camada é a API, ou *Application Programming Interface* em inglês, essa camada representa uma especificação do *framework* e nela encontram-se apenas as interfaces que devem ser implementadas pelas camadas subseqüentes.

A camada abaixo da API se chama Camada Abstrata. Nesta camada encontram-se as regras que independem do ESB utilizado, como o gerenciamento das regras propostas pelo padrão VETRO, o carregamento do arquivo de configuração, entre outros. A Camada Abstrata foi criada com o intuito de permitir que o ESB utilizado possa ser mais facilmente trocado, apenas reimplementando a camada abaixo, chamada Camada ESB.

Na camada ESB encontram-se todas as regras específicas do ESB utilizado, como a criação dos endpoints, e a utilização dos protocolos implementados pelo barramento de serviços. A ilustração das camadas do Onix pode ser vista na figura 6.2.

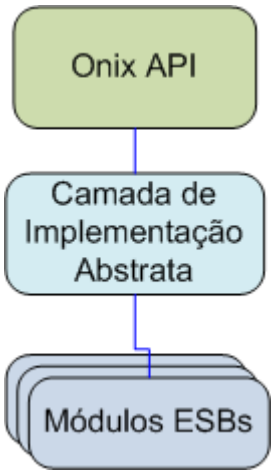

Figura 6.2: Ilustração das camadas do Onix Framework

A idéia geral do *framework*, como já foi dito anteriormente, é que ele funcione como um barramento de mensagens. Além disso, todas as mensagens que entrarem neste barramento sofrerão as regras previstas no padrão VETRO, sendo que cada *endpoint* de entrada possuirá seu próprio conjunto de regras. Estas regras serão configuradas através de arquivos XML, que devem obedecer ao esquema XML que será detalhado no item a seguir.

6.2.1 Esquema do XML de Configuração

O arquivo de configuração do Onix deve estar em conformidade com o seu esquema XML. Um esquema XML é uma gramática baseada em XML para documentos

XML[JXML2002]. Ou seja, é um conjunto de regras pré-definidas que documentos XML devem obedecer para estarem sintática, léxica e semanticamente corretos.

Os principais elementos do esquema serão explicados detalhadamente nos itens a seguir.

6.2.1.1 Elemento onix-config

Elemento base para a configuração do *framework*, no qual são adicionadas todas as regras de validação, enriquecimento, transformação e tratamento de erros que serão utilizadas no sistema. Também são adicionados os fluxos de integração, chamados de *integrations chains*. O diagrama deste elemento é mostrado na figura 6.3.

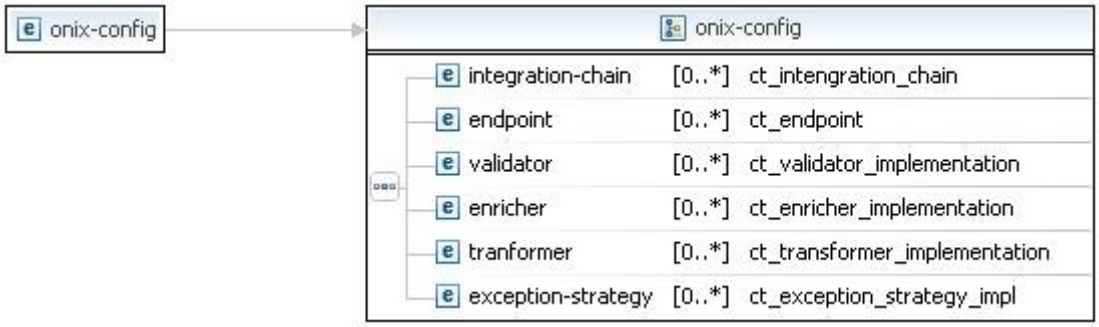

Figura 6.3: Diagrama do elemento onix-config.

6.2.1.2 Elemento ct_integration_chain

O ct_integration_chain descreve os fluxo de integração do sistema por onde passaram as mensagens. Cada fluxo possui um sub fluxo de envio(ct_send) e outro de resposta (ct_receive). Fluxo de notificação deve possuir só um sub fluxo de envio, já os fluxos que possuem respostas deverão ter um sub fluxo de enviou outro de resposta. Neste elemento também se encontra o elemento que trata exceções do fluxo, o *exception strategy*. O diagrama deste elemento é mostrado na figura 6.4.

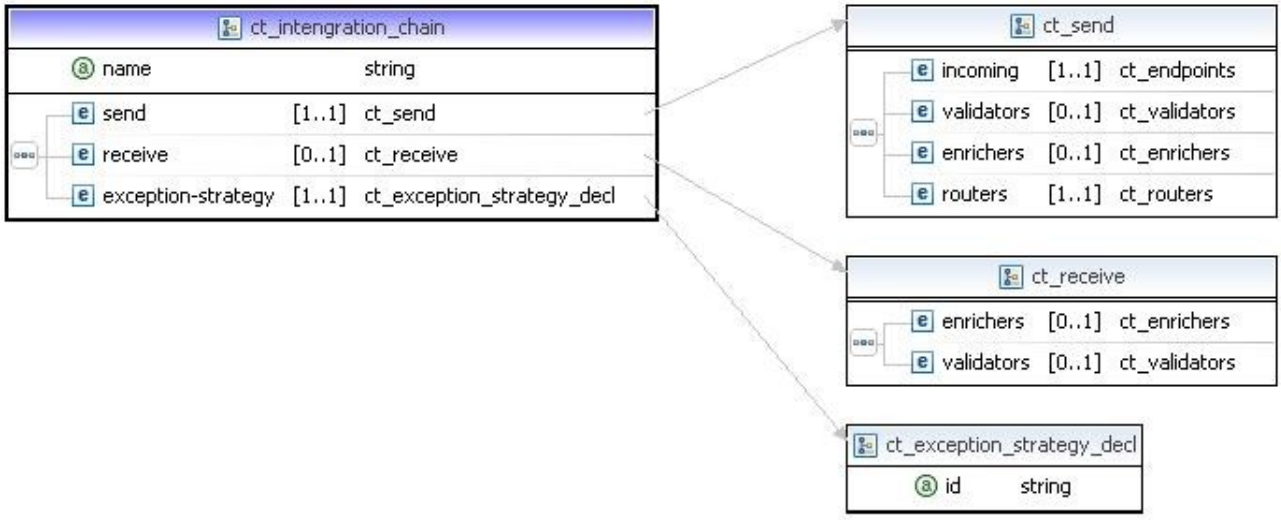

Figura 6.4: Diagrama do elemento ct_integration_chain

6.2.1.3 Elemento ct_send

O elemento ct_send representa o fluxo de envio da mensagem, este fluxo contém as referências para as regras de validação, transformação, enriquecimento e roteamento da mensagem. Também contém a referência para os endpoints de entrada do fluxo, os "incoming endpoints" e para as regras de roteamento da mensagem. O diagrama deste elemento é mostrado na figura 6.5.

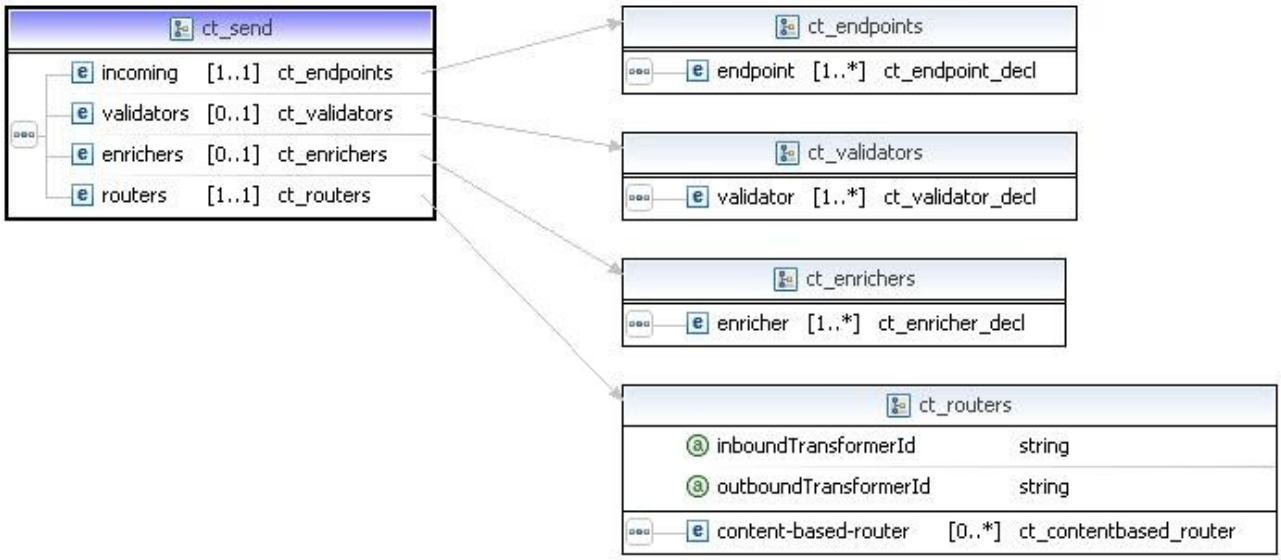

Figura 6.5: Diagrama do elemento ct_send.

6.2.1.4 Elemento ct_receive

O ct_receive representa o fluxo de recebimento da mensagem, ele é executado quando o sistema destino retorna alguma resposta para o sistema origem. Neste elemento são declaradas as regras de enriquecimento e de validação para a mensaagem

de volta. O diagrama deste elemento é mostrado na figura 6.6.

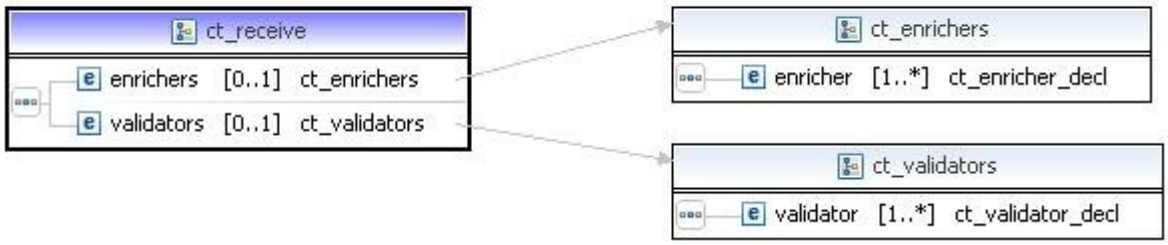

Figura 6.6: Diagrama do elemento ct_receive.

6.2.1.5 Elemento ct_exception_strategy_impl

No ct_exception_strategy são declaradas as regras para o tratamento das exceções que ocorrem durante o fluxo de integração. Cada fluxo de integração tem direito a uma estratégia de exceção

O diagrama deste elemento é mostrado na figura 6.7.

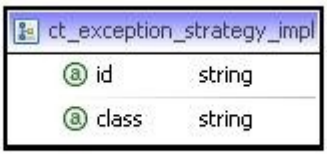

Figura 6.7: Diagrama do elemento ct_exception_strategy_impl.

6.2.1.6 Elemento ct_endpoint

O elemento ct_endpoint representa os pontos de entrada e saída do sistema. Ele possui a declaração do protocolo, endereço e porta utilizados. Caberá à camada do ESB interpretar esse elemento e criar um ponto de escuta para aceitar ou enviar novas requisições. O diagrama deste elemento é mostrado na figura 6.8.

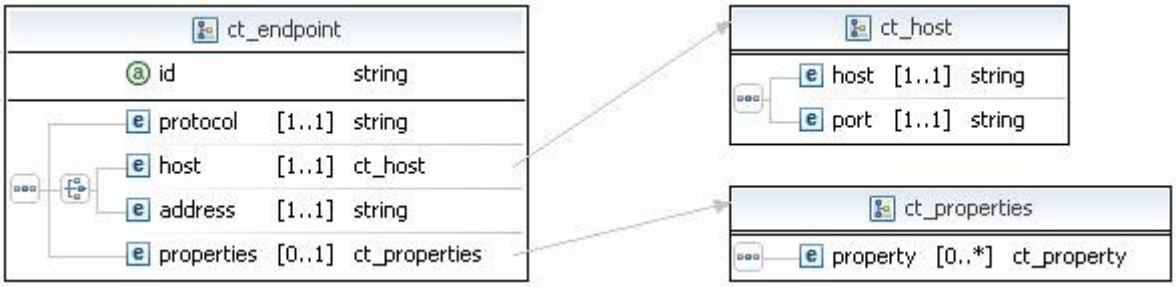

Figura 6.8: Diagrama do elemento ct_endpoint.

6.2.1.7 Elemento ct_contentbased_router

O elemento ct_contentbased_router representa um roteador de mensagens baseado em conteúdos. Neste elemento devem ser declarados a classe do roteador, as propriedades contendo o conteúdo a ser filtrado e, o endpoint para o qual o roteador enviará a mensagem caso o conteúdo dela seja equivalente ao esperado e os *transformers* a serem utilizados para enviar e receber a mensagem. A figura 6.9 demonstra o diagrama deste elemento.

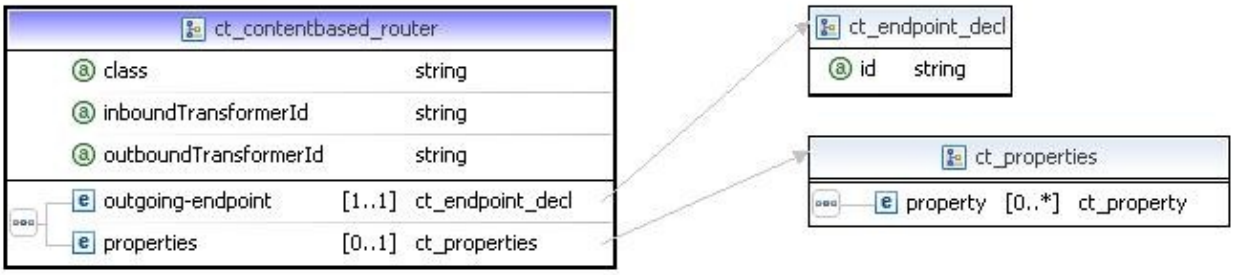

Figura 6.9: Diagrama do elemento ct_content_based_router.

6.2.1.8 Elemento ct_broadcasting_router

O elemento ct_broadcasting_router declara um roteador que envia a mensagem recebida para todos os seus *endpoints*, não utilizando nenhum critério de filtragem. No *broadcasting router* devem ser declarados uma lista com os *endpoints* que a mensagem será enviada e o *transformer* a ser utilizado.

A figura 6.10 demonstra o diagrama deste elemento.

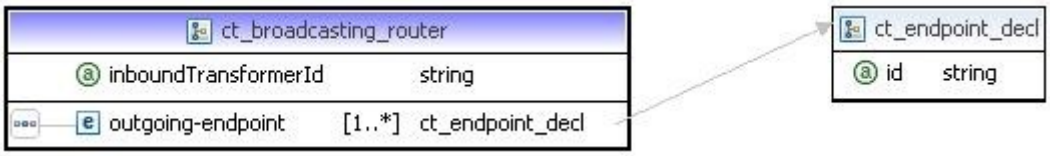

Figura 6.10: Diagrama do elemento ct_broadcasting_router

6.2.2 Classes de Tipos

As classes de tipos são os elementos do esquema XML convertidos para objetos Java. Estas classes são compartilhadas por todas as camadas do Onix e não possuem nenhuma regra de negócio, apenas métodos *getters* e *setter* para adicionar e recuperar informações. Estas classes servem basicamente para representar através de objetos Java

as configurações que são realizadas via XML.

As classes de tipos se encontram no pacote *br.inf.ufsc.onix.config.types*.

6.2.3 Interfaces da API

A API, ou *Application Programming Interface* como é conhecida no mundo do desenvolvimento de software, é uma camada que declara os métodos públicos do *framework*. Sendo assim, todos os métodos que os usuários do *framework* terão acesso estarão declarados nessas interfaces.

A criação de uma camada de API também auxilia na construção de classes de baixo acoplamento, tendo em vista que as referências para os objetos do sistema serão todas realizadas através de suas interfaces.

6.2.3.1 Message e Header

As interfaces *Message* e *Header* declaram os métodos de acesso e modificação dos elementos das mensagens que trafegam pelo Onix.

O *Header* contém os métodos de acesso aos elementos do cabeçalho da mensagem. A primeiro elemento é o *inboundEndpoint*, no qual está declarado o endereço do *endpoint* de entrada da mensagem, que é preenchido de forma automática pelo Onix e é mantido durante todo o processo de integração. O segundo elemento é um mapa chave-valor para inserir e recuperar propriedades que o usuário do *framework* deseje utilizar durante o processo.

A *Message* é uma interface simples que apenas declara os métodos para o seu *Header* e para o seu *Body*, que é o conteúdo da mensagem.

O diagrama de classe dessas interfaces é mostrado a seguir na figura 6.11.

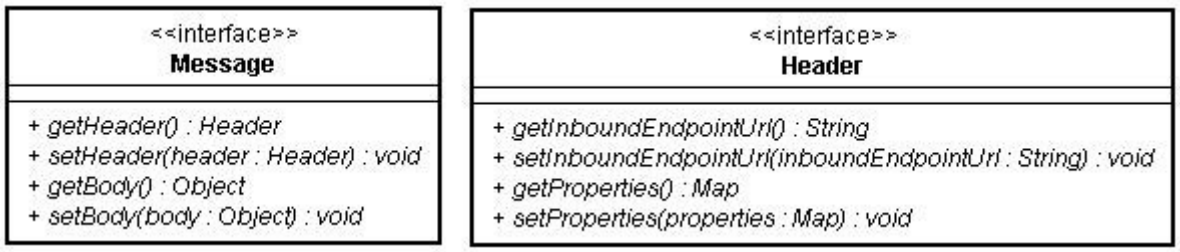

Figura 6.11: Diagrama de Classes das interfaces Message e Header.

6.2.3.2 LifeCycle e OnixEngine

As interfaces *LifeCycle* e *OnixEngine* são as estruturas com as quais os desenvolvedores que utilizarem o *framework* terão mais contato, pois nelas estão

declarados os métodos para gerenciamento do estado do Onix(iniciado ou parado), carregamento de arquivos XML de configuração, dentre outros.

A *LifeCycle* define os métodos para componentes que possuam um ciclo de vida padrão para o sistema. Nesta primeira versão no Onix apenas o OnixEngine possuirá um ciclo de vida padrão.

Os métodos do LifeCycle são:

- **start**: Inicializa o componente.
- **stop**: Finaliza a execução do componente.
- **restart**: Reinicializa o componente.

A *OnixEngine* define os métodos do coração do *framework*, desde o método de envio dos arquivos de configuração, passando pelos métodos para registro e recuperação dos *integrations chains* configurados, chegando nos métodos de processamento de mensagens.

Também deve-se considerar que esta interface estende a *LifeCycle*. Isso significa que o desenvolvedor acessará os métodos para gerenciamento do ciclo de vida do Onix através da interface *OnixEngine*.

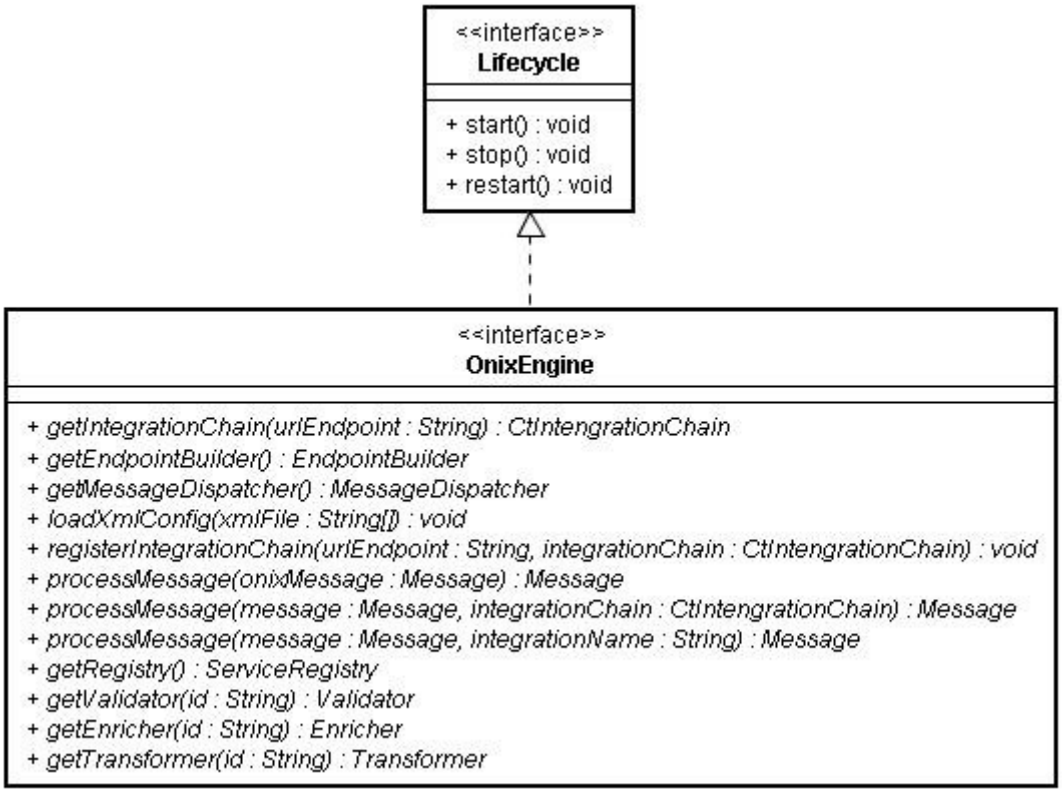

Figura 6.12: Diagrama de Classes das interfaces LifeCycle e OnixEngine.

6.2.3.3 ServiceRegistry

A interface *ServiceRegistry* especifica a estrutura de métodos de um repositório de serviços. Os repositórios de serviços, ou *registry* como são chamados internamente, são as estruturas responsáveis pelo armazenamento de todos os *integrations chains* carregados pelo Onix.

Os métodos declarados pelo *ServiceRegistry* são os seguintes:

- ***getServiceByEndpoint***: Retorna um *integration chain* relacionado ao *endpoint* enviado como parâmetro.
- ***getServiceByName***: Retorna um *integration chain* cadastrado com o nome enviado como parâmetro.
- ***register:*** Registra um *integration chain* no repositório de serviços relacionando com o *endpoint* enviado.
- ***hasService***: Verifica se existe o *integration chain* cadastrado no Onix com o *endpoint* enviado.

Uma implementação concreta do ServiceRegistry pode ser conseguida através do método *getRegistry* do *OnixEngine*. A figura 6.13 demonstra o diagrama de classes deste elemento.

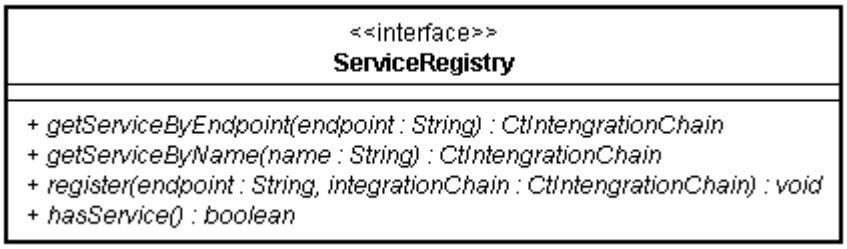

Figura 6.13: Diagrama de Classes da interface ServiceRegistry.

6.2.3.4 EndpointBuilder

A *EndpointBuilder* é uma interface que declara os métodos que, em suas implementações, terão a função de criar *endpoints* de entrada e saída no ESB utilizado, e, por conseguinte, no Onix. Os *endpoints* de entrada são chamados de *inboundEndpoint* e os de saída de *outboundEndpoint.* Uma implementação concreta do *EndpointBuilder* pode ser conseguida através do método *getEndpointBuilder* do *OnixEngine*.

Os métodos declarados pelo EndpointBuilder são os seguintes:

- ***buildEsbInboundEndpoint****:* Cria um *endpoint* de entrada no ESB a partir dos dados do *CtEndpoint* enviado como parâmetro. Este *CtEndpoint* é configurado via XML no elemento ct_endpoint.

- ***BuildEsbOutboundEndpoint****:* Cria um *endpoint* de saída no ESB a partir dos dados do *CtEndpoint* enviado como parâmetro.
- ***getEndpointAddress****:* Retorna o endereço no ESB referente ao *CtEndpoint* enviado como parâmetro.

O diagrama de classes da interface *EndpointBuilder* pode ser visto na figura 6.14.

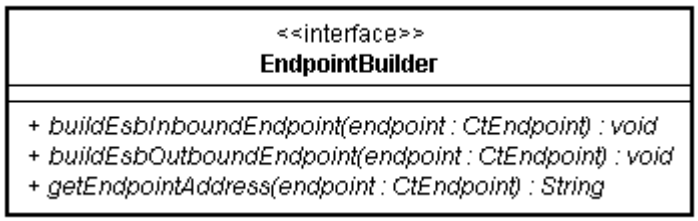

Figura 6.14: Diagrama de Classes da interface EndpointBuilder

6.2.3.5 MessageDispatcher

O usuário do Onix *framework* tem a opção de utilizar um *MessageDispatcher* para enviar mensagens diretamente aos *integrations chains* carregados, em vez de ter que mandar para o *endpoint* de entrada para que este as encaminhem aos respectivos *chains*. Uma referência a uma implementação concreta do *message dispatcher* pode ser conseguida através do método *getMessageDispatcher* do *OnixEngine*.

Os métodos que devem ser implementados pela classe que estender esta interface são:

- ***sendMessage***: Envia a mensagem recebida como parâmetro ao *integration chain* recebida também como parâmetro e retorna a resposta do sistema destino.
- ***dispatchMessage***: Despacha a mensagem recebida como parâmetro ao *integration chain* também recebido como parâmetro, este método não aguarda a resposta do sistema destino.
- ***sendMessageToEsb:*** Envia a mensagem para o *endpoint* do ESB recebido de parâmetro.
- ***dispatchMessageToEsb:*** Despacha uma mensagem de notificação para o *endpoint* do ESB recebido como parâmetro.

A figura 6.15 demonstra o diagrama de classes desta interface.

Figura 6.15: Diagrama de Classes da interface MessageDispatcher.

6.2.3.6 Validator, Enricher, Transformer, Router

As interfaces *Validator*, *Enricher* e *Transformer* são alguns dos pontos de extensão do Onix *Framework*. Os desenvolvedores usuários do Onix devem implementar estas interfaces e registrar essas implementações no XML de configuração para que o *framework* possa realizar as regras previstas no padrão VETRO.

Classes que implementam a interface *Validator* são as responsáveis por executar regras de validação em cima das mensagens que trafegam pelo *framework*. Cada *integration chain* pode ter de zero ou muitos *Validators* para cada etapa, *send* e *receive*. A cardinalidade zero ou muitos incentiva que as classes de validação sejam implementadas com alta coesão, realizando apenas uma validação especifica por classe, facilitando assim testes e manutenção dos artefatos criados.

Classes que implementam a interface *Enricher* são as responsáveis por executar procedimentos de enriquecimento das mensagens que passa pelo *framework*. Como acontece com as classes de validação, cada *integration chain* pode conter zero ou muitos enriquecedores para cada etapa, permitindo, novamente, a alta coesão dos artefatos.

Já as classes que implementam a interface *Transformer* são as responsáveis pelas regras de transformação, tanto para saída quanto para a entrada das mensagens.

Por fim, as classes que implementam a interface *Router*, são responsáveis pela lógica de roteamento das mensagens para os seus *endpoints* de saída. Estes *endpoints* representam o ponto de entrada do sistema destino, como já foi visto neste trabalho.

O diagrama de classe destas interfaces podem ser vistos na figura 6.16.

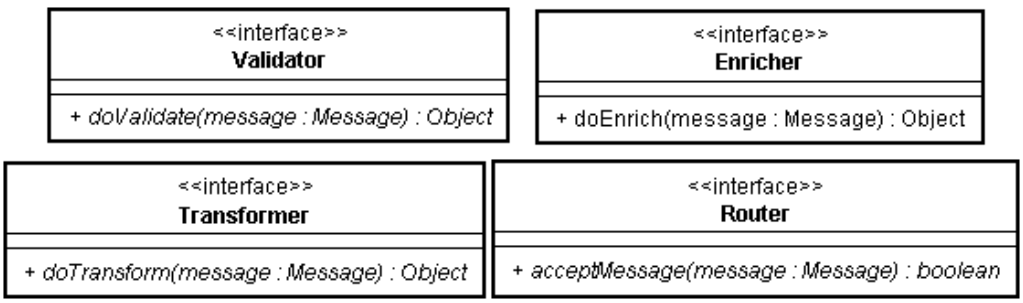

Figura 6.16: Diagrama de Classes das interfaces Validator, Enricher, Transformer e Router.

6.2.3.7 ExceptionHandler

A interface *ExceptionHandler* define outro ponto de extensão do *framework*, os componentes para tratamento de exceções. Cada *integration chain* pode possuir um *ExceptionHandler* que, caso ocorra alguma exceção durante o processo de integração, pode realizar o devido tratamento e retornar uma mensagem mais amigável a quem realizou a requisição.

A figura 6.17 demonstra o digrama de classes da interface *ExceptionHandler*.

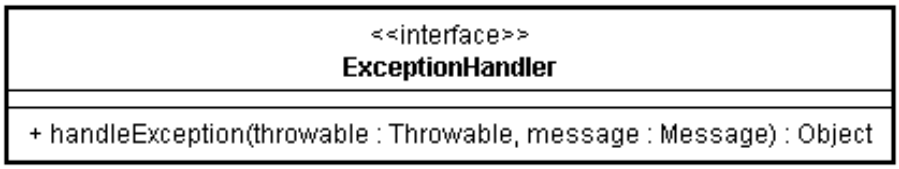

Figura 6.17: Diagrama de Classes da interface ExceptionHandler.

6.2.4 Classes da Camada Abstrata

A camada abstrata fornece uma implementação padrão dos métodos das interfaces da API que independem do ESB que está sendo utilizado.

Nesta camada está contida a lógica da *engine* de execução das regras do padrão VETRO, também existe uma implementação padrão do *ServiceRegistry* e do *MessageDispatcher* que serão mostradas a seguir.

Uma ilustração das principais classes da camada abstrata pode ser vista na figura 6.18.

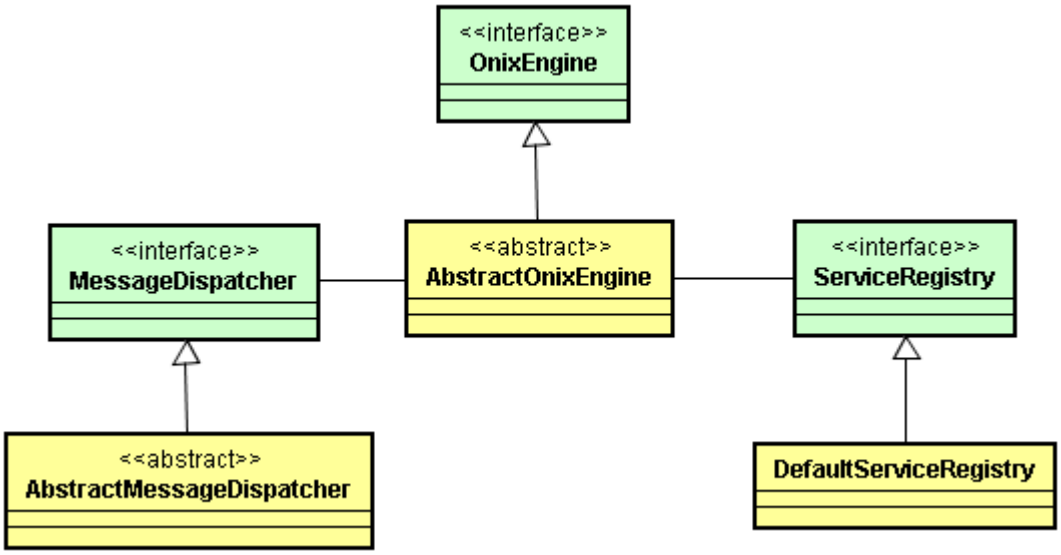

Figura 6.18: Ilustração das principais classes da camada abstrata em amarelo, suas interfaces da camada da API em verde.

6.2.4.1 AbstractMessageDispatcher

A interface *MessageDispatcher* fornece quatro métodos de envio de mensagens, porém apenas dois deles são independentes do ESB utilizado, o *sendMessage* e o *dispatchMessage*, pois eles servem apenas para enviar a mensagem para dentro do *Onix Framework.* Estes dois métodos são implementados pela classe abstrata *AbstractMessageDispatcher*. Já os métodos *sendMessageToEsb* e *dispatchMessageToEsb* dependem da camada ESB e devem ser implementados por uma classe filha.

6.2.4.2 AbstractOnixEngine

O *AbstractOnixEngine* pode ser considerado o coração do Onix *Framework*. É nele que está contida a engine que executa as regras previstas pelo padrão VETRO – validação, enriquecimento, transformação, roteamento e operação. Também está implementado nesta classe o método que carrega as configurações dos arquivos XML e transforma em objetos internos do Onix.

6.2.4.3 DefaultServiceRegistry

A classe *DefaultServiceRegistry* fornece uma implementação padrão para a interface *ServiceRegistry*, que nada mais é do que um repositório com todos os *integrations chains* carregados pelo framework. Esta é uma classe concreta e, por conseguinte, implementa todos os métodos descritos na sua interface.

6.2.4.4 OnixHeader e OnixMessage

As classes *OnixHeader* e *OnixMessage* são implementações padrão das interfaces *Header* e *Message*. Estas duas classes são concretas e utilizadas por padrão pelo *framework* para representar as mensagens que trafegam internamente. Entretanto, nada impede que o desenvolvedor reimplemente as interfaces supracitadas e as use no lugar das implementações providas por padrão. A figura 6.19 demonstra um diagrama com a estrutura destas classes.

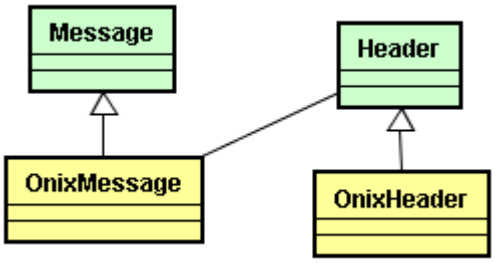

Figura 6.19: Diagrama de classe de alto nível das implementações padrões das interfaces Header e Message.

6.2.5 Classes da Camada ESB Utilizando Mule ESB

A camada ESB é a última da pilha de camadas do Onix *Framework*. Nesta camada está presente toda a lógica ligada ao ESB sendo utilizado. No Onix *Framework*, o ESB é o artefato responsável pela criação e gerenciamento dos *endpoints* de saída e de entrada do sistema, sendo assim, a maior parte do código contido nesta camada está intimamente relacionado ao tema *endpoint*.

A figura 6.20 mostra uma ilustração da arquitetura das classes do Onix, incluindo a camada ESB.

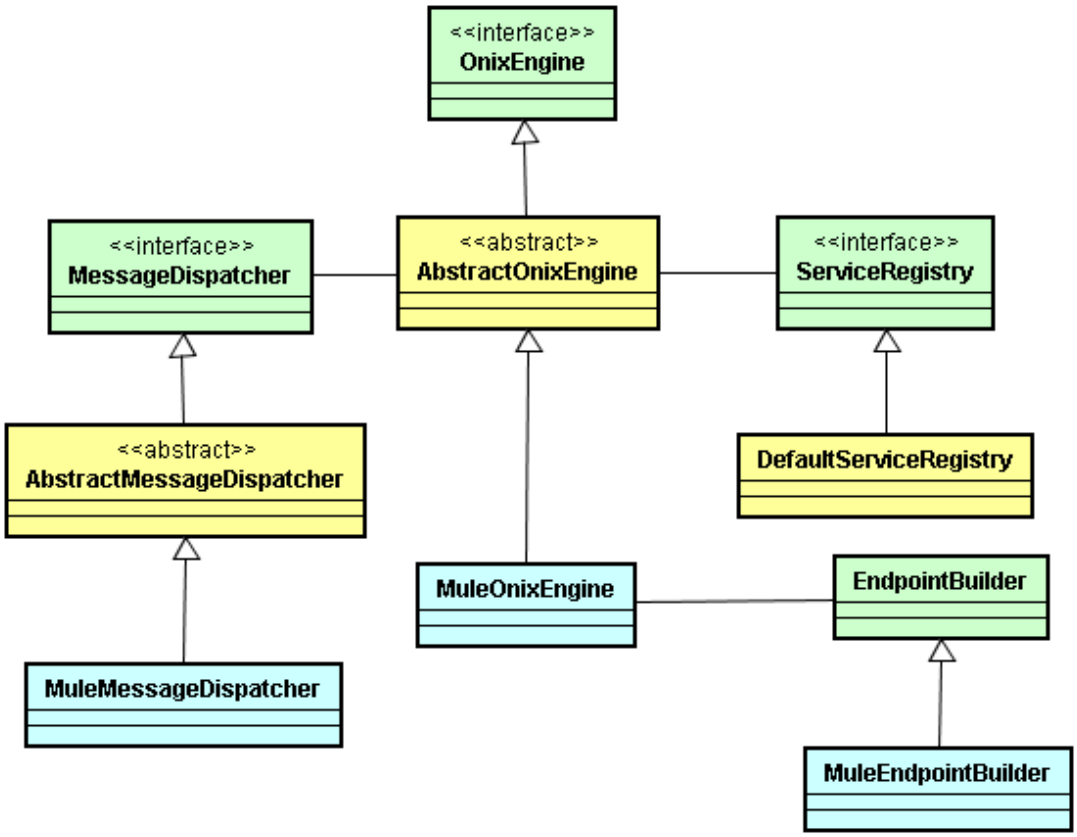

Figura 6.20: Em verde as interfaces da API, em amarelo as classes da camadas abstrata e em azul as classes da camada do Mule ESB.

6.2.5.1 MuleEndpointBuilder

A classe *MuleEndpointBuilder* é a responsável por criar os *endpoints* no Mule ESB. Ela é utilizada internamente pelo Onix para criar os *endpoints* configurados via XML, independente do protocolo que eles utilizem, seja TCP/IP, HTTP, File ou outros.

6.2.5.2 MuleMessageDispatcher

A *MuleMessageDispatcher* contém as regras para o envio ou despacho das mensagens para algum endpoint do ESB, diferentemente da *AbstractMessageDispatcher*, que contém a lógica para enviar ou despachar mensagens para os *integrations chains* configurados no Onix.

6.2.5.3 MuleOnixEngine

A classe *MuleOnixEngine* tem dois objetivos principais, implementar os métodos do ciclo de vida do Onix, definidos pela interface *LifeCycle*; e instanciar os objetos das outras classes da camada ESB, como *MuleEndpointBuilder* e *MuleMessageDispatcher*, para que sejam acessíveis ao resto do framework.

6.2.6 Inicialização do Framework

A inicialização do *framework* acontece de forma simples e rápida. Primeiramente o usuário necessita de uma referência à alguma implementação do *OnixEngine.* A implementação padrão utiliza o Mule como ESB e uma referência pode ser obtida através do método *getInstance* da classe *MuleOnixEngine*, que é um *singleton*.

De posse de uma instância do *OnixEngine*, o próximo passo é enviar ao *framework* os arquivos de configuração, os quais contém os *Validators*, *Enrichers*, *Transformers*, dentre outros componentes previstos pela especificação do *framework*. O envio dos arquivos de configurações é realizado através do método *loadXmlConfig* da *Engine*, que pode receber um ou muitos caminhos de arquivos de configuração.

O último passo para a inicialização do Onix acontece com a invocação do método *start*, que é o responsável por inicializar o ESB, os *endpoints* e alterar o estado do Onix para ativo, tornando-o apto a realizar o processamento de mensagens.

A figura 6.21 demonstra o diagrama de seqüência da inicialização do Onix *Framework*.

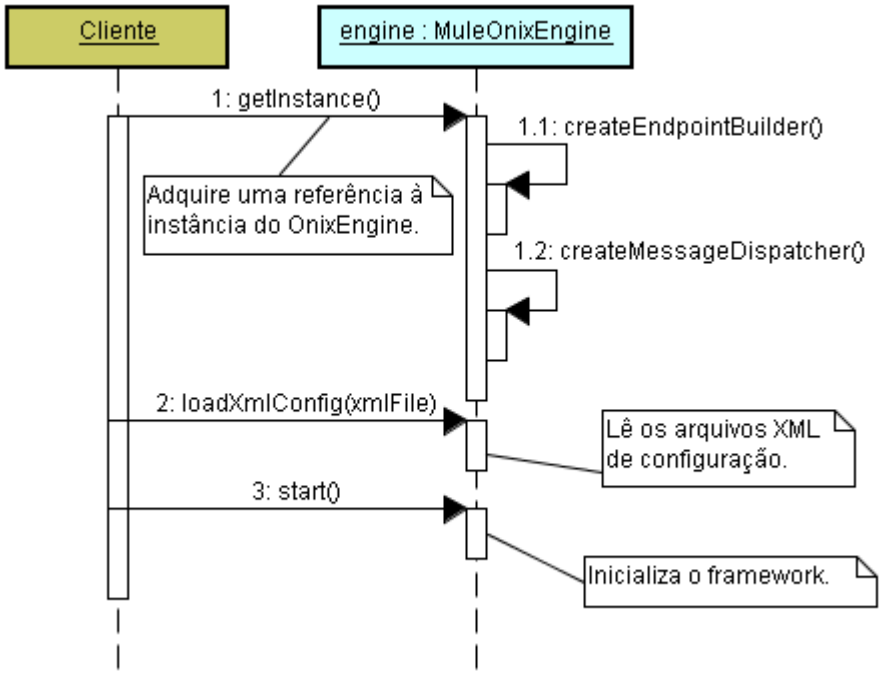

Figura 6.21: Seqüência de inicialização do Onix Framework.

6.2.7 Processamento da Mensagens

O processamento de mensagens pode ser considerado a “razão de existir” do Onix *Framework*, por este motivo ele foi planejado de forma a ser facilmente extensível, com

classes de baixo acoplamento, com amplo uso de interfaces, e de alta coesão, pois cada classe tem seu papel bem definido dentro do Onix.

Esta funcionalidade pode ser iniciada de duas maneiras distintas, o *framework* pode receber uma mensagem em um de seus *endpoints* ativos, por exemplo, receber uma mensagem via *socket* em uma porta que o Onix esteja ouvindo; ou, a mensagem pode ser enviada ao *framework* através dos métodos *sendMessage* e *dispatchMessage* do *MessageDispatcher*, como é mostrado na figura 6.22.

Após a mensagem chegar a um dos dois pontos de entrada explicados acima, ela é enviada ao *OnixEngine*, o qual executa as regras previstas no padrão VETRO. Por fim a mensagem é despachada até o seu destino através do *MessageDispatcher* novamente.

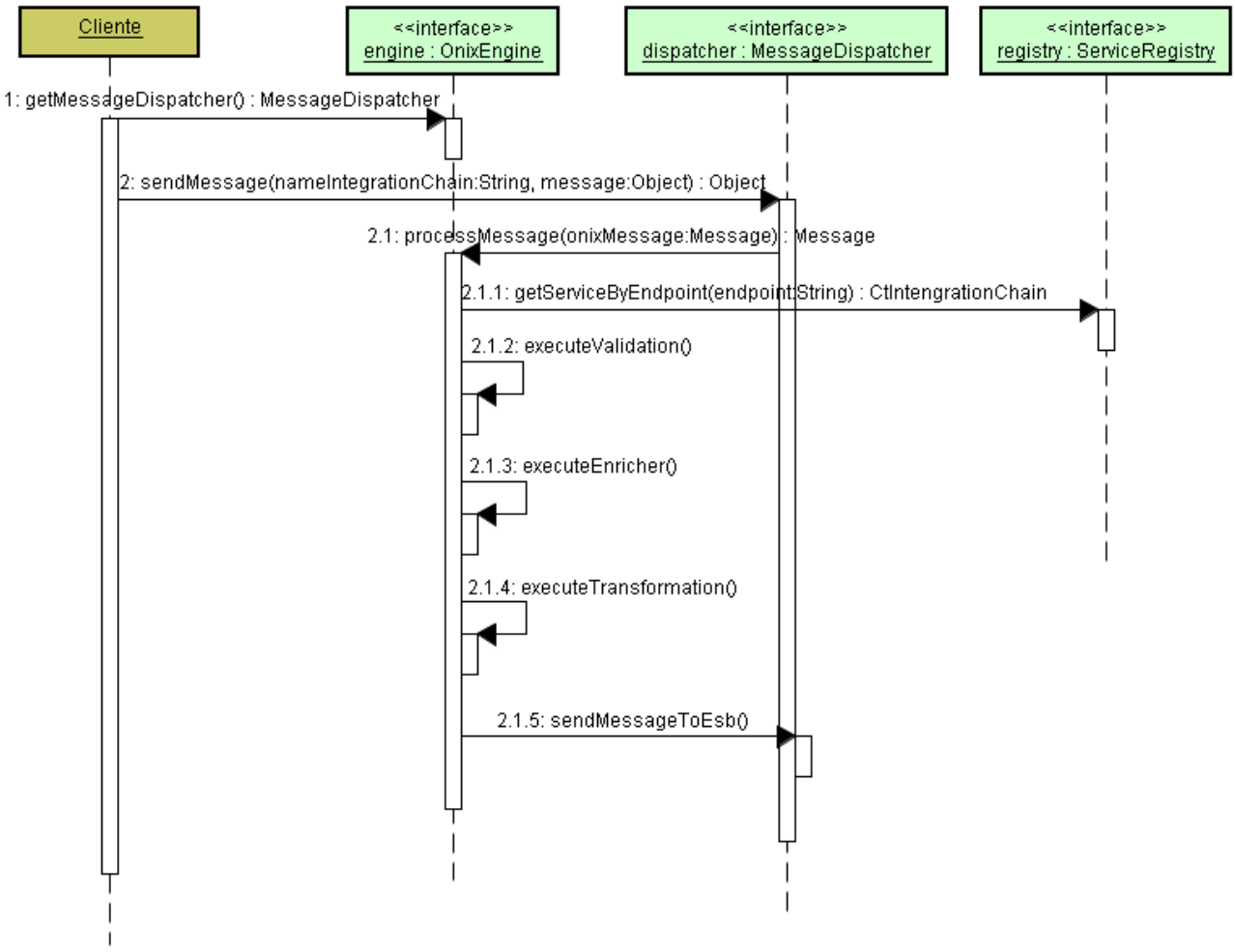

Figura 6.22: Seqüência do processamento de mensagens do Onix a partir do MessageDispatcher.

6.3 Resumo

Este capítulo procurou mostrar aspectos estruturais e comportamentais do Onix *Framework*.

A primeira parte demonstrou a arquitetura do framework, suas camadas e classes mais importantes. Isto foi feito especialmente para que os desenvolvedores que se interessem em colaborar no desenvolvimento de mais funcionalidades possam se familiarizar mais rapidamente com o Onix.

Já a segunda parte procurou mostrar alguns aspectos comportamentais, que auxiliará os usuários desenvolvedores que desejem adotar o Onix em seus projetos, mostrando como configurar e inicializar o *framework*.

7 Estudo de Caso

Este estudo de caso tem como objetivo validar a utilização do Onix Framework e demonstrar como o mesmo pode ser usado em uma situação real de aplicação. Será aplicado um exemplo simples com a intenção de apenas atender aos fatores antes citados.

7.1 Aplicação

A proposta de implementação de aplicação consiste em um sistema de aprovação de empréstimos bancários. Sendo assim, o sistema invocaria o ESB, implementado utilizando o Onix, que realizaria processos de validação, enriquecimento e transformação da mensagem, e faria o roteamento da mesma em função do valor do empréstimo. Caso o valor seja inferior a 10 mil reais, a mensagem seria roteada para o Banco A, e caso seja superior a 10 mil, ao Banco B.

O diagrama de atividades da imagem 7.1 ilustra o funcionamento do sistema.

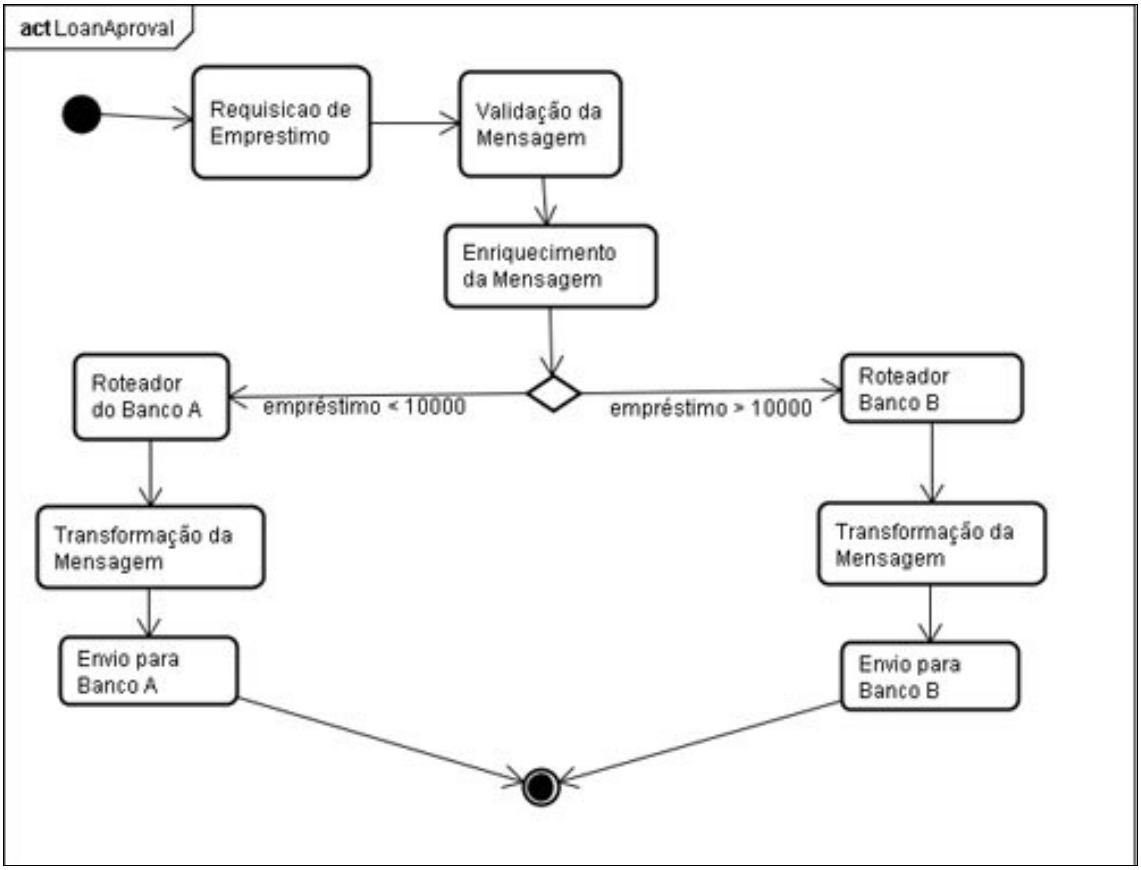

Figura 7.1: Diagrama de atividades para a aprovação de empréstimos.

7.2 Implementação

Primeiro deve-se construir o XML de configuração do Onix para o funcionamento desta aplicação. Neste arquivo será incluído um elemento "*integration-chain*", declarando também um nome para o mesmo. Dentro deste elemento serão declarados os componentes do fluxo que se deseja construir, ressaltando que a ordem dos tipos de componentes segue o padrão VETRO, conforme explicado nos capítulos anteriores.

O Fluxo da mensagem será feito em apenas um passo: a requisição. Portanto se faz necessária apenas a declaração da *tag* "*send*". A resposta da aprovação de empréstimo se dará de modo assíncrono, logo, em um fluxo diferente. A invocação do serviço será feita via W*ebservice,* neste caso sendo utilizado o Apache CXF para a implementação do mesmo, e os bancos destinatários receberão dados via *socket* TCP. A figura 7.2 mostra a estrutura do *integration-chain* no arquivo de configuração em XML desenvolvido para este exemplo.

```
<onx:integration-chain name="loanAprover">
    <onx:send>
        <!-- Endpoint de entrada -->
        <onx:incoming>
            <onx:endpoint id="entradaLoan" />
        </onx:incoming>
        <!-- Procedimentos de validacao-->
        <onx:validators>
            <onx:validator id="cpfValidator" />
        </onx:validators>
        <!-- Procedimentos de enriquecimento da mensagem -->
        <onx:enrichers>
            <onx:enricher id="msgEnricher" />
        </onx:enrichers>
        <!-- Roteamento -->
        <onx:routers>
            <onx:content-based-router
                class="br.inf.ufsc.testonix.router.BancoAContentBasedRouter"
                inboundTransformerId="bancoATransformerIn"
                outboundTransformerId="bancoATransformerOut">
                <onx:outgoing-endpoint id="bancoAEndpoint" />
            </onx:content-based-router>
            <onx:content-based-router
                class="br.inf.ufsc.testonix.router.BancoBContentBasedRouter"
                inboundTransformerId="bancoBTransformerIn"
                outboundTransformerId="bancoBTransformerOut">
                <onx:outgoing-endpoint id="bancoBEndpoint" />
            </onx:content-based-router>
        </onx:routers>
    </onx:send>
    <!-- Em caso de erro -->
    <onx:exception-strategy id="exceptionHandlerLoanAprover" />
</onx:integration-chain>
```

Figura 7.2: Configuração do integration-chain no arquivo XML.

Dentro do *integration-chain* foram especificados vários identificadores de componentes. É preciso então fazer a declaração destes componentes referenciados pelos identificadores, e desta maneira o Onix saberá o que é cada elemento e onde encontrá-lo. No *endpoint* de entrada, do tipo W*ebservice*, foi necessária a declaração da classe que implementa o servidor do serviço, enquanto nos *endpoints* de saída não foi necessário, pois isso não se faz necessário no protocolo TCP. Na imagem 7.3 estes componentes são especificados dentro do arquivo de configuração do Onix

```
<onx:endpoint id="entradaLoan">
    <onx:protocol>WS_CXF</onx:protocol>
    <onx:address>http://localhost/loanAprover</onx:address>
    <properties>
        <property name="implClass">
            br.com.ufsc.LoanAproverImpl
        </property>
    </properties>
</onx:endpoint>
<onx:endpoint id="bancoAEndpoint">
    <onx:protocol>TCP</onx:protocol>
    <onx:host>
        <onx:host>192.168.0.1</onx:host>
        <onx:port>60000</onx:port>
    </onx:host>
</onx:endpoint>
<onx:endpoint id="bancoBEndpoint">
    <onx:protocol>TCP</onx:protocol>
    <onx:host>
        <onx:host>192.168.0.1</onx:host>
        <onx:port>60040</onx:port>
    </onx:host>
</onx:endpoint>

<onx:validator id="cpfValidator"
    class="br.inf.ufsc.testonix.validator.CpfValidator" />
<onx:enricher id="msgEnricher"
    class="br.inf.ufsc.testonix.enricher.LoanRequestEnricher" />
<onx:exception-strategy id="exceptionHandlerLoanAprover"
    class="br.inf.ufsc.testonix.exception.LoanApproverExceptionHandler" />
```

Figura 7.3: Especificação dos componentes utilizados no integration-chain.

Depois de declarados os componentes, eles devem ser implementados na prática. Cada um dos tipos destes componentes deve obedecer a um padrão, implementando uma interface específica para cada tipo de elemento. Neste exemplo, precisa-se de um validador que verifique a validade do CPF do cliente que está pedindo aprovação de empréstimo. Esta classe já foi especificada no XML que foi determinado anteriormente. A implementação desta classe esta ilustrada na figura 7.4.

```java
package br.inf.ufsc.testonix.validator;

import br.inf.ufsc.onix.api.Message;

public class CpfValidator implements Validator {

    @Override
    public void doValidate(Message message) throws Exception {
        CtLoanRequest req = (CtLoanRequest) message.getBody();
        String cpf = req.getCustomerData().getCustomerId();
        boolean isCpfValido = false;
        // Realizar validacoes;

        if (!isCpfValido) {
            throw new Exception("CPF do cliente é invalido.");
        }

    }

}
```

Figura 7.4: Implementação da classe de validação do CPF.

Assim como o validador e os enriquecedores, transformadores também precisam de classes que realizem as suas implementações para serem invocadas de maneira dinâmica pelo Onix. Neste caso o enriquecedor vai verificar se o cliente tem algum registro que o caracterize como um mal pagador de contas e enriquecer a mensagem com esta informação, como pode-se ver na figura 7.5.

```java
package br.inf.ufsc.testonix.enricher;

import br.inf.ufsc.onix.api.Enricher;

public class LoanRequestEnricher implements Enricher {

    @Override
    public Object doEnrich(Message message) {
        CtLoanRequest req = (CtLoanRequest) message.getBody();
        String cpf = req.getCustomerData().getCustomerId();
        boolean isClienteMalPagador = false;
        // Realizar validacoes;

        req.getCustomerData().setIsBadCustomer(isClienteMalPagador);

        return req;
    }

}
```

Figura 7.5: Implementação da classe que realiza o enriquecimento da mensagem.

É preciso implementar os *content-based-routers* para que o Onix saiba para qual

destinatário encaminhar esta mensagem. Neste caso, se o empréstimo for inferior a 10 mil deve-se encaminhar a mensagem para o banco A, e caso seja superior a 10 mil, envia-se a mensagem ao banco B. Na figura 7.6 temos a implementação da classe que verifica para qual banco a mensagem deve ser enviada.

```
1  package br.inf.ufsc.testonix.router;
2
3  import br.inf.ufsc.onix.api.Message;
6
7  public class BancoAContentBasedRouter implements Router {
8
9      @Override
10     public boolean acceptMessage(Message onixMessage) {
11         CtLoanRequest req = (CtLoanRequest) onixMessage.getBody();
12
13         if (req.getLoanData().getLoanValue() < 10000) {
14             return true;
15         }
16
17         return false;
18     }
19
20 }
```

Figura 7.6: Implementação do roteador baseado em conteúdo.

Cada um destes dois bancos pode operar em diferentes tipos de mensagens, e por isso existem os *transformers.* Antes de enviar os dados ao destinatário, o Onix passará a mensagem pelo transformador para obter o formato de dados adequado ao destino. Na imagem 7.7 foi implementado um transformador de mensagens simples que constrói um *array* de *bytes* a partir da concatenação dos dados contidos na mensagem de entrada.

```java
package br.inf.ufsc.testonix.transformer;

import br.inf.ufsc.onix.api.Message;

public class BancoAMessageTransformer implements Transformer {

    @Override
    public Object doTransform(Message message) throws Exception {
        CtLoanRequest req = (CtLoanRequest) message.getBody();

        String x = req.getCustomerData().getCustomerId() + ";"
                + req.getCustomerData().getCustomerName() + ";"
                + req.getCustomerData().isIsBadCustomer() + "/n";
        x += req.getLoanData().getLoanValue() + ";"
                + req.getLoanData().getNumberOfPayments();

        return x.getBytes();
    }
}
```

Figura 7.7: Implementação do transformador de mensagens para o Banco A.

O componente de *exception strategy* será invocado caso ocorra alguma exceção nas etapa do fluxo da mensagem. Nestes casos o objetivo principal é tratar esta exceção de alguma maneira conveniente para o sistema e também informar o usuário que realizou a requisição da ocorrência deste erro. No exemplo demonstrado na figura 7.8 foi implementado um *exception strategy* que apenas retorna uma simples mensagem de erro.

```java
package br.inf.ufsc.testonix.exception;

import br.inf.ufsc.onix.api.ExceptionHandler;

public class LoanApproverExceptionHandler implements ExceptionHandler {

    @Override
    public Object handleException(Throwable throwable, Message message) {
        String errorMessage = "Erro ao processar mensagem: "
                + throwable.getMessage();
        return errorMessage;
    }

}

```

Figura 7.8: Implementação do componente de exception strategy.

Por fim, deve-se implementar a classe que iniciará o Onix. Basta apenas criar uma classe com um método estático para execução e especificar o arquivo de configuração construído para que sejam carregados todos os componentes desejados pelo usuário. Caso haja algum erro no arquivo, o usuário será alertado. Na figura 7.9 mostra-se o código de uma classe inicializadora do Onix para este exemplo.

```java
package br.inf.ufsc.testonix.main;

import br.inf.ufsc.onix.api.OnixEngine;

public class StartApp {

    /**
     * @param args
     * @throws OnixException
     */
    public static void main(String[] args) throws OnixException {
        String xmlConfig = StartApp.class.getResource("onixConfig.xml")
                .getPath();
        try {
            OnixEngine engine = MuleOnixEngine.getInstance();
            engine.loadXmlConfig(xmlConfig);
            engine.start();
        } catch (Exception e) {
            e.printStackTrace();
            System.exit(-1);
        }

    }
}
```

Figura 7.9: Implementação da classe que inicia o Onix.

8 CONCLUSÃO

8.1 Resultados Alcançados

Para alcançar os resultados deste trabalho uma série de tecnologias e padrões tiveram que ser estudados mais profundamente, como os conceitos de *Enterprise Integration Patterns* e *Enterprise Service Bus*. Com isso, os autores puderam ter uma base mais sólida para o desenvolvimento do Onix.

A partir do estudo envolvendo integração empresarial e padrões de integração, tivemos contato com o que se chama de arquitetura acidental, que provavelmente foi a principal razão pelos sistemas empresariais possuírem tamanha complexidade para integração. Esta foi nossa principal motivação para o desenvolvimento do *framework*.

Sendo assim, projetamos o Onix para que possa se enquadrar nesse cenário acidental, tendo compatibilidade com uma grande gama de aplicações já existentes e protocolos.

A tabela abaixo apresenta o conjunto de requisitos do projeto Onix, e a forma no qual foram ou não atingidos.

Requisito	Detalhe
Configuração utilizando arquivos XML	Foi criado um esquema XML que define o formato do arquivo XML aceito pelo *framework*. Este arquivo pode ser enviado ao *OnixEngine* através do método **loadXmlConfig()**.
Capacidade de executar as regras descritas no padrão VETRO	Foram criadas as interfaces *Validator*, *Enricher*, *Transformer* e *Router* para que sejam estendidas e implementem as quatro primeiras fases do padrão VETRO. A fase de operação se dá quando a mensagem é encaminhada ao sistema destino.
Capacidade de enviar uma mensagem para um único destinatário	Com o componente *Router*, é possível especificar um *endpoint* de saída para um único destinatário.
Capacidade de enviar uma mensagem para vários destinatários	Foi criada uma implementação do *Router* chamada *BroadcastingRouter*, que é capaz de enviar a mesma mensagem para vários destinatários diferentes.
Capacidade de trabalhar com diversos tipos de protocolos	Nesta versão inicial, o Onix aceita a declaração dos *endpoints* de entrada e saída nos seguintes protocolos: SOAP, TCP/IP, File, SMTP e POP3.
Capacidade de realizar a troca informações entre sistemas que utilizam protocolos diferentes	É possível criar um *integration chain* com um *endpoint* de entrada usando um protocolo, e o de saída utilizando outro.
Capacidade de utilizar o ESB que desejar	O Onix utiliza um arquitetura em camadas que possibilita a fácil substituição da camada ESB.

Tabela 8.1: Tabela de requisitos e detalhes do Onix.

8.2 Trabalhos Futuros

Pode-se considerar que o presente trabalho serviu como um impulso inicial para o

desenvolvimento do Onix *Framework*, construindo uma base sólida e capaz de resolver os problemas em questão. Apesar disso, para se tornar competitivo no ambiente empresarial, existem uma série de funcionalidades básicas que ainda precisam ser implementadas.

O desenvolvimento de um módulo para gerenciamento de *logging* e auditoria é algo essencial para um *framework* que deseje fazer parte do meio corporativo, principalmente em ambientes altamente distribuídos, cenário no qual o Onix se enquadra.

Além disso, é fundamental a criação de um módulo de segurança. Assim será possível garantir que o sujeito que envia as mensagens é realmente quem diz ser e se possui permissão para enviá-las.

Futuramente, a intenção dos autores é que também se associe ao Onix um módulo de gerenciamento, com alertas e notificações. Este módulo seria responsável por monitorar recursos de hardware (memória RAM, uso de CPU, etc) e recursos de software(memória da máquina virtual, notificação de exceções, etc).

Como se pode observar, há muito trabalho a ser feito para que o Onix se torne um *framework* competitivo e confiável para o uso empresarial. A próxima etapa dos autores será a disponibilização do código fonte do Onix para a comunidade, através de portais como *Devnet* ou *SourceForge*.

8.3 Considerações Finais

Com o desenvolvimento deste projeto foi possível adquirir um conhecimento mais profundo tanto na área de integração de sistemas, quanto no assunto *Enterprise Service Bus* em si.

A complexidade envolvida no desenvolvimento de projetos deste tipo é grande. A começar pela especificação do projeto, que deve ser realizado de forma a criar uma arquitetura suficientemente flexível. Também foram enfrentados problemas para a utilização do Mule ESB, já que tivemos que fazer uma utilização não muito trivial configurando os *endpoints* manualmente, o que normalmente é feito via arquivos de configuração XML.

Acreditamos que o Onix atingiu o principal objetivo ao qual se propôs – a construção de um *framework* que implemente o padrão VETRO. Apesar das muitas dificuldades encontradas durante todo o desenvolvimento, estamos satisfeitos com os resultados alcançados. A partir de agora esperamos que a comunidade desenvolvedora possa nos ajudar a melhorar este projeto de forma incremental, pois sabemos das dificuldades de continuar no desenvolvimento do mesmo sem este auxílio.

9 BIBLIOGRAFIA

[EIP2004] HOHPE, Gregor; WOOLF, Bobby. **Enterprise Integrations Patterns – Designing, Building and Deploying Messaging Solutions.** Addison-Wesley, 2004.

[EAA2002] FOWLER, Martin; **Patterns of Enterprise Application Architecture.** Addison-Wesley, 2002.

[ESB2004] CHAPPELL, David. **Enterprise Service Bus**, O'Reilly, 2004.

[JMS2000] MONSON-HAEFEL, Richard; CHAPPEL, David. **Java Message System**. O'Reilly, 2000.

[JXML2002] GABRICK, Kurt; WEISS, David. **J2EE XML Development**. Manning, 2002.

[ERI2003] BURKE, Eric M.; COYNER, Brian M. **Java Extreme Programming Cookbook**.O'Reilly 2003.

[JOH1988] R.E Johnson, B. Foote. **Designing Reusable Classes**. Journal of Object-Oriented Programming, 1988.

[TAL1994] Taligent Inc. **Building object-oriented frameworks**. Disponível em: <http://lhcb-comp.web.cern.ch/lhcb-comp/Components/postscript/buildingoo.pdf>. Acessado em: 03/04/2009.

[LAN1995] LANDIN, Nicklas; NIKLASSON, Axel. **Development Of Object Oriented Frameworks**. Ericsson Software Technology,1995. Disponível em: <http://www.dsc.ufcg.edu.br/~jacques/cursos/map/recursos/developing-frame.pdf>. Acesso em: 21/03/2009.

[GAR2002] SCHULTE, Roy; LHEUREUX, Benoit; NATIS, Yefim; PEZZINI, Massimo; THOMPSON, Jess. **Integration Brokers, Application Servers and APSs**. Gartner, 2002. Disponível em: <http://www.bus.umich.edu/KresgePublic/Journals/Gartner/research/111000/111072/111072.html>. Acesso em: 19/03/2009.

[RMI2004] Sun Microsystems. **Java Remote Method Invocation**. 2004. Disponível em: <http://java.sun.com/j2se/1.5.0/docs/guide/rmi/spec/rmiTOC.html>. Acesso em: 31/03/2009.

[MUL2008] Mule Source. **Mule 2 Getting Started Guide**. Disponível em: <http://www.mulesource.org/display/MULE2INTRO/Separating+Business+Logic+from+Messaging>. Acesso em: 19/03/2009.

10 Anexos e Apêndices

10.1 Apêndice 1 – Artigo sobre o Projeto

10.2 Anexo 1 - Código Fonte do Onix Framework

Interface EndpointBuilder

```
package br.inf.ufsc.onix.api;

import br.inf.ufsc.onix.config.types.CtEndpoint;
import br.inf.ufsc.onix.exception.OnixEndpointException;
import br.inf.ufsc.onix.exception.OnixInitializationException;

/**
 * Interface generica com os metodos para a criacao de endpoints utilizado pelo
 * Onix.
 *
 * @author Gabriel Nunes Menezes<bielmenezes@gmail.com>
 *
 * @since 15/01/2009
 */
public interface EndpointBuilder {
	/**
	 * Constroi um endpoint no ESB sendo utilizado com as configuracoes do
	 * CtEndpoint enviado.
	 *
	 * @param endpoint
	 * @throws OnixInitializationException
	 * @throws OnixEndpointException
	 */
	public void buildEsbInboundEndpoint(CtEndpoint endpoint)
			throws OnixInitializationException, OnixEndpointException;

	/**
	 * Constroi um outbound endpoint no ESB sendo utilizado com as configuracoes
	 * do CtEnpoind enviado.
	 *
	 * @param endpoint
	 * @throws OnixInitializationException
	 * @throws OnixEndpointException
	 */
	public void buildEsbOutboundEndpoint(CtEndpoint endpoint)
			throws OnixInitializationException, OnixEndpointException;

	/**
	 * Retorna a representação do endereço do endpoint no ESB sendo utilizado
	 * representado pelo CtEndpoint enviado de parâmetro.
	 *
```

```
         * @param endpoint
         * @return
         */
        public String getEndpointAddress(CtEndpoint endpoint);

}
```

Interface Enricher

```
package br.inf.ufsc.onix.api;

/**
 *
 * @author Gabriel Nunes Menezes<bielmenezes@gmail.com>
 *
 * @since 29/11/2008
 */
public interface Enricher {

        Object doEnrich(Message message) throws Exception;

}
```

Interface ExceptionHandler

```
package br.inf.ufsc.onix.api;


/**
 * Interface para os componentes que iraao tratar as excecoes que ocorrem
 * durante o fluxo.
 *
 * @author Gabriel Nunes Menezes<bielmenezes@gmail.com>
 *
 * @since 22/01/2009
 */
public interface ExceptionHandler {

        Object handleException(Throwable throwable, Message message);

}
```

Interface Header

```
package br.inf.ufsc.onix.api;

import java.io.Serializable;
import java.util.Map;

public interface Header extends Serializable{
```

```
	public abstract String getInboundEndpointUrl();

	public abstract void setInboundEndpointUrl(String inboundEndpointUrl);

	public abstract Map<Object, Object> getProperties();

	public abstract void setProperties(Map<Object, Object> properties);

}
```

Interface LifeCycle

```
package br.inf.ufsc.onix.api;

import br.inf.ufsc.onix.exception.OnixException;

/**
 * Ciclo de vida para componentes do Onix.
 *
 * @author Gabriel Nunes Menezes<bielmenezes@gmail.com>
 *
 * @since 15/01/2009
 */
public interface LifeCycle {

	void start() throws OnixException;

	void stop() throws OnixException;

	void restart() throws OnixException;

}
```

Interface Message

```
package br.inf.ufsc.onix.api;

import java.io.Serializable;


public interface Message extends Serializable {

	public abstract Header getHeader();

	public abstract void setHeader(Header header);

	public abstract Object getBody();

	public abstract void setBody(Object body);

}
```

Interface MessageDispatcher

```
package br.inf.ufsc.onix.api;

import br.inf.ufsc.onix.exception.OnixEndpointException;
import br.inf.ufsc.onix.exception.OnixException;

/**
 * Interface com a assinatura dos metodos utilizados para enviar mensagens ao
 * Onix.
 *
 * @author Gabriel Nunes Menezes<bielmenezes@gmail.com>
 *
 * @since 17/01/2009
 */
public interface MessageDispatcher {

	/**
	 * Envia uma mensagem ao Onix que contém resposta.
	 *
	 * @throws OnixEndpointException -
	 *             caso não exista o integration chain com o nome enviado.
	 * @param message
	 * @return
	 * @throws OnixException
	 */
	Object sendMessage(String nameIntegrationChain, Object message)
			throws OnixEndpointException, OnixException;

	/**
	 * Envia uma mensagem sem resposta ao Onix.
	 *
	 * @throws OnixEndpointException -
	 *             caso não exista o integration chain com o nome enviado.
	 *
	 * @param message
	 */
	void dispatchMessage(String nameIntegrationChain, Object message)
			throws OnixEndpointException, OnixException;

	/**
	 * Envia uma mensagem ao ESB que contém resposta.
	 *
	 * @throws OnixEndpointException -
	 *             caso não exista o integration chain com o nome enviado.
	 * @param message
	 * @return
	 * @throws OnixException
	 */
	Object sendMessageToEsb(String url, Object message)
			throws Exception;
```

```
	/**
	 * Envia uma mensagem sem resposta ao ESB.
	 *
	 * @throws OnixEndpointException -
	 *             caso não exista o integration chain com o nome enviado.
	 *
	 * @param message
	 */
	void dispatchMessageToEsb(String url, Object message)
			throws Exception;

}
```

Interface OnixEngine

```
package br.inf.ufsc.onix.api;

import br.inf.ufsc.onix.config.types.CtIntengrationChain;
import br.inf.ufsc.onix.exception.OnixException;

/**
 * Interface contendo os metodos com as principais funcionalidades do framework.
 *
 * @author Gabriel Nunes Menezes<bielmenezes@gmail.com>
 *
 * @since 29/11/2008
 */
public interface OnixEngine extends LifeCycle {

	/**
	 * Retorna o Integration Chain que possui o endpoint de entrada enviado como
	 * argumento.
	 *
	 * @param urlEndpoint
	 *            - endpoint de entrada do integration chain.
	 *
	 * @return
	 */
	CtIntengrationChain getIntegrationChain(String urlEndpoint);

	/**
	 * Retorna uma instancia do endpointbuilder da implementacao utilizada.
	 *
	 * @return
	 */
	EndpointBuilder getEndpointBuilder();

	/**
	 * Retorna uma instancia do message dispatcher da implementacao do Onix
	 * utilizada.
	 *
	 * @return
```

```
 */
MessageDispatcher getMessageDispatcher();

/**
 * Carrega a lista de arquivos de configuracao enviadas como parametro.
 *
 * @param xmlFile
 * @throws OnixException
 * @throws IllegalArgumentException
 */
void loadXmlConfig(String... xmlFile) throws OnixException;

/**
 * Adiciona o integrationChain ao framework.
 *
 * @param urlEndpoint
 * @param integrationChain
 */
void registerIntegrationChain(String urlEndpoint,
                CtIntengrationChain integrationChain);

/**
 * Executa o fluxo de processamento configurado para o Integration chain da
 * mensagem enviada.
 *
 * @param onixMessage
 * @return
 * @throws OnixException
 */
Message processMessage(Message onixMessage) throws OnixException;

/**
 * Processa a mensagem enviada no integration chain recebido.
 *
 * @param onixMessage
 * @return
 * @throws OnixException
 */
Message processMessage(Message onixMessage, CtIntengrationChain chain)
                throws OnixException;

/**
 * Busca um integrationChain com o nome enviado e envia a mensagem para ser
 * processado por ele.
 *
 * @param onixMessage
 * @param chainName
 * @return
 * @throws OnixException
 */
Message processMessage(Message onixMessage, String chainName)
                throws OnixException;
```

```
        ServiceRegistry getRegistry();

        Validator getValidator(String id);

        Enricher getEnricher(String id);

        Transformer getTransformer(String id);

}
```

Interface Router

```
package br.inf.ufsc.onix.api;

/**
 *
 * @author Pedro Castello Branco Galoppini<pgaloppini@gmail.com>
 * @since 20/04/2009
 *
 */
public interface Router {

        public boolean acceptMessage(Message onixMessage) throws Exception;

}
```

Interface ServiceRegistry

```
package br.inf.ufsc.onix.api;

import br.inf.ufsc.onix.config.types.CtIntengrationChain;

/**
 * Interface contendo os metodos que devem ser implementados para um repositorio
 * com os servicos que serao cadastrados.
 *
 * @author Gabriel Nunes Menezes<bielmenezes@gmail.com>
 *
 * @since 15/01/2009
 */
public interface ServiceRegistry {

        /**
         * Retorna um IntegrationChain do endpoint enviado, null se não existir.
         *
         * @param endpoint
         * @return
         */
        CtIntengrationChain getServiceByEndpoint(String endpoint);

        /**
```

```
	 * Retorna um IntegrationChain que possui o name enviado, null se não
	 * existir.
	 *
	 * @param name
	 * @return
	 */
	CtIntengrationChain getServiceByName(String name);

	/**
	 * Registra um IntegrationChain com o endpoint enviado.
	 *
	 * @param endpoint
	 * @param integrationChain
	 */
	void register(String endpoint, CtIntengrationChain integrationChain);

	/**
	 * Verifica se existe um IntegrationChain registrado com o endpoint enviado.
	 *
	 * @param endpoint
	 * @return
	 */
	boolean hasService(String endpoint);

}
```

Interface Transformer

```
package br.inf.ufsc.onix.api;

/**
 *
 * @author Gabriel Nunes Menezes<bielmenezes@gmail.com>
 *
 * @since 29/11/2008
 */
public interface Transformer {

	Object doTransform(Message message) throws Exception;

}
```

Interface Validator

```
package br.inf.ufsc.onix.api;



/**
 *
 * @author Gabriel Nunes Menezes<bielmenezes@gmail.com>
 *
```

```
 * @since 29/11/2008
 */
public interface Validator {

	void doValidate(Message message) throws Exception;

}
```

Class ConfigLoader

```
package br.inf.ufsc.onix.config.jaxb;

import java.io.File;
import java.io.InputStream;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBElement;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;
import javax.xml.transform.Source;
import javax.xml.transform.stream.StreamSource;
import javax.xml.validation.Schema;
import javax.xml.validation.SchemaFactory;

import org.xml.sax.SAXException;

import br.inf.ufsc.onix.config.types.OnixConfig;

/**
 * This class unmarshall files from onix schema and gets the Jaxb types from the
 * source file
 *
 *
 * @author Pedro Galoppini<pgaloppini@gmail.com>
 *
 */
public class ConfigLoader {

	private Unmarshaller unmarshaller;

	private static ConfigLoader singleton;

	private static final String PATH_SCHEMA_XSD = "onixComplexTypes.xsd";

	private static String PACKAGE_ONIX_SCHEMA = "br.inf.ufsc.onix.config.types";

	private ConfigLoader() {
		try {
			JAXBContext context =
JAXBContext.newInstance(PACKAGE_ONIX_SCHEMA);
			unmarshaller = context.createUnmarshaller();
			InputStream schemaInputStream = ConfigLoader.class
```

```
                                        .getResourceAsStream(PATH_SCHEMA_XSD);
                        Source inputSource = new StreamSource(schemaInputStream);
                        Schema schema = SchemaFactory.newInstance(
                                        "http://www.w3.org/2001/XMLSchema").newSchema(inp
utSource);
                        unmarshaller.setSchema(schema);

                } catch (JAXBException e) {
                        System.out.println("Initialize error on ConfigLoader: " + e);
                        throw new RuntimeException(e);
                } catch (SAXException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        }

        public static ConfigLoader getInstance() {
                if (singleton == null) {
                        singleton = new ConfigLoader();
                }
                return singleton;
        }

        /**
         * Gets OnixConfig from source file
         *
         * @param source
         * @return
         * @throws Exception
         */
        public OnixConfig loadConfig(File source) throws Exception {
                try {
                        JAXBElement<?> obj = (JAXBElement<?>)
unmarshaller.unmarshal(source);
                        if (!(obj.getValue() instanceof OnixConfig)) {
                                throw new Exception("Expected <onx:onix-config> at File "
                                                + source.getAbsolutePath());
                        }
                        OnixConfig config = (OnixConfig) obj.getValue();

                        return config;
                } catch (Exception e) {
                        // TODO tratamento de excecao;
                        throw new Exception(e);
                }

        }

        public static void main(String[] args) {
                try {
                        OnixConfig teste = ConfigLoader.getInstance().loadConfig(
                                        new File(ConfigLoader.class
```

```
                                                        .getResource("exemploConfig.xml").getPat
h()));
                        System.out.println(teste.getIntegrationChain().get(0).getName());
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

Class OnixConfigLoader

```
package br.inf.ufsc.onix.config.loader;

import java.io.File;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.apache.log4j.Logger;

import br.inf.ufsc.onix.api.Enricher;
import br.inf.ufsc.onix.api.ExceptionHandler;
import br.inf.ufsc.onix.api.Router;
import br.inf.ufsc.onix.api.Transformer;
import br.inf.ufsc.onix.api.Validator;
import br.inf.ufsc.onix.config.jaxb.ConfigLoader;
import br.inf.ufsc.onix.config.types.CtBroadcastingRouter;
import br.inf.ufsc.onix.config.types.CtContentbasedRouter;
import br.inf.ufsc.onix.config.types.CtEndpoint;
import br.inf.ufsc.onix.config.types.CtEndpointDecl;
import br.inf.ufsc.onix.config.types.CtEnricherDecl;
import br.inf.ufsc.onix.config.types.CtEnricherImplementation;
import br.inf.ufsc.onix.config.types.CtExceptionStrategyDecl;
import br.inf.ufsc.onix.config.types.CtExceptionStrategyImpl;
import br.inf.ufsc.onix.config.types.CtIntengrationChain;
import br.inf.ufsc.onix.config.types.CtRouters;
import br.inf.ufsc.onix.config.types.CtTransformerImplementation;
import br.inf.ufsc.onix.config.types.CtValidatorDecl;
import br.inf.ufsc.onix.config.types.CtValidatorImplementation;
import br.inf.ufsc.onix.config.types.OnixConfig;
import br.inf.ufsc.onix.config.types.StProtocols;
import br.inf.ufsc.onix.exception.OnixException;

/**
 * This class validates all config files seted at Onix startup Class
 *
 * @author Pedro Galoppini <pgaloppini@gmail.com>
 * @author Gabriel Nunes Menezes<bielmenezes@gmail.com>
 * @since 16/02/2009
 */
public class OnixConfigLoader {
```

```
	private Map<String, CtTransformerImplementation> transformers;
	private Map<String, CtEnricherImplementation> enrichers;
	private Map<String, CtValidatorImplementation> validators;
	private Map<String, CtExceptionStrategyImpl> exceptionStrategy;
	private Map<String, CtEndpoint> endpoints;
	private Map<String, CtEndpoint> inboundEndpoints;
	private Map<String, CtEndpoint> outboundEndpoints;

	private Map<String, CtIntengrationChain> integrationChain;

	private Logger logger = Logger.getLogger(OnixConfigLoader.class);

	public OnixConfigLoader() {
		transformers = new HashMap<String, CtTransformerImplementation>();
		enrichers = new HashMap<String, CtEnricherImplementation>();
		validators = new HashMap<String, CtValidatorImplementation>();
		exceptionStrategy = new HashMap<String, CtExceptionStrategyImpl>();
		endpoints = new HashMap<String, CtEndpoint>();
		integrationChain = new HashMap<String, CtIntengrationChain>();
		inboundEndpoints = new HashMap<String, CtEndpoint>();
		outboundEndpoints = new HashMap<String, CtEndpoint>();
	}

	public void loadConfigs(String[] configFilePaths) throws Exception {
		logger
				.info("Iniciando procedimento que carrega os arquivos de
configuração.");
		if (configFilePaths == null || configFilePaths.length == 0) {
			logger
					.warn("Nenhum arquivo de configuração para carregar,
abortando procedimento.");
			throw new OnixException("No config files to load.");
		}

		this.getData(configFilePaths);

		this.validateComponents();

	}

	private void getData(String[] configFilePaths) throws Exception {
		for (int i = 0; i < configFilePaths.length; i++) {
			logger.info("Carregando configuração presente no arquivo '"
					+ configFilePaths[i] + "'.");
			OnixConfig config = ConfigLoader.getInstance().loadConfig(
					new File(configFilePaths[i]));

			if (config == null) {
				continue;
			}
			try {
				for (CtIntengrationChain chain : config.getIntegrationChain()) {
```

```
                                        if (chain.getName() == null
                                                        || chain.getName().trim().equals(
                                                                        "Integration Chain Name is
invalid.")) {

                                        }

                                        if (integrationChain.get(chain.getName()) != null) {
                                                throw new OnixException(

                                                                "Integration Chain "
                                                                                + chain.getName()
                                                                                + " with two or more
references in config files"
                                                                                + ", choose other id.");
                                        }
                                        logger.info("Incluindo IntegrationChain " +
chain.getName()
                                                                        + " a lista de carregamento.");
                                        integrationChain.put(chain.getName(), chain);
                                }
                                // Carrega declaracao das implementacoes
                                logger.info("Carregando componentes declarados.");
                                this.loadTransformers(config.getTranformer());
                                this.loadEndpoints(config.getEndpoint());
                                this.loadEnrichers(config.getEnricher());
                                this.loadExceptions(config.getExceptionStrategy());
                                this.loadValidators(config.getValidator());
                                logger.info("Carregamento do arquivo '" + configFilePaths[i]
                                                + "' feita com sucesso.");
                        } catch (OnixException o) {
                                throw new OnixException(o.getMessage() + " - at File: "
                                                + configFilePaths[i]);
                        }
                }

        }

        private void validateComponents() throws OnixException {
                for (CtIntengrationChain chain : integrationChain.values()) {
                        try {
                                logger.info("Validando componentes do integration Chain: "
                                                + chain.getName());
                                this.validateEndpoints(chain.getSend().getIncoming()
                                                .getEndpoint());
                                this.validateValidators(chain.getSend().getValidators()
                                                .getValidator());
                                this.validateEnrichers(chain.getSend().getEnrichers()
                                                .getEnricher());
                                this.validateException(chain.getExceptionStrategy());
                                this.validateRouters(chain.getSend().getRouters());
                                logger.info("Validação do integration chain '"
```

```
                                        + chain.getName() + "' efetuada sem erros.");
                        } catch (OnixException o) {
                                throw new OnixException(o.getMessage() + " - at "
                                        + chain.getName() + " chain.");
                        }

                }

        }

        /**
         * Loads transformers from file and validates if theirs IDs have already
         * been declared
         *
         * @param tranformersList
         * @throws OnixException
         */
        private void loadTransformers(
                        List<CtTransformerImplementation> tranformersList)
                        throws OnixException {
                if (tranformersList != null && !tranformersList.isEmpty()) {
                        logger
                                        .info("Iniciando inclusao de lista de declaraçoes de
tranformers.");
                        for (CtTransformerImplementation impl : tranformersList) {
                                if (impl.getId() == null || impl.getId().trim().equals("")) {
                                        throw new OnixException("Transformer with class '"
                                                        + impl.getClazz()
                                                        + "' has a invalid ID declaration.");
                                }

                                if (!isIdValid(impl.getId())) {
                                        throw new OnixException("Transformer '" + impl.getId()
                                                        + "' with two or more references in config
files"
                                                        + ", choose other id.");
                                }

                                this.isClassValid(impl.getClazz(), Transformer.class);
                                logger.info("Transformer: '" + impl.getId()
                                                + "' é valido e esta sendo incluido.");
                                transformers.put(impl.getId(), impl);
                        }
                }
        }

        /**
         * Loads enrichers from file and validates if theirs IDs have already been
         * declared
         *
         * @param tranformers
         * @throws OnixException
```

```
	 */
	private void loadEnrichers(List<CtEnricherImplementation> enrichersList)
			throws OnixException {
		if (enrichersList != null && !enrichersList.isEmpty()) {
			logger
					.info("Iniciando inclusao de lista de declaraçoes de
enrichers.");
			for (CtEnricherImplementation impl : enrichersList) {
				if (impl.getId() == null || impl.getId().trim().equals("")) {
					throw new OnixException("Enricher with class '"
							+ impl.getClazz()
							+ "' has a invalid ID declaration.");
				}

				if (!isIdValid(impl.getId())) {
					throw new OnixException("Enricher '" + impl.getId()
							+ "' with two or more references in config
files"
							+ ", choose other id.");
				}

				this.isClassValid(impl.getClazz(), Enricher.class);
				logger.info("Enricher: '" + impl.getId()
						+ "' é valido e esta sendo incluido.");
				enrichers.put(impl.getId(), impl);
			}
		}
	}

	/**
	 * Loads endpoints from file and validates if theirs IDs have already been
	 * declared
	 *
	 * @param tranformers
	 * @throws OnixException
	 */
	private void loadEndpoints(List<CtEndpoint> endpointsList)
			throws OnixException {
		if (endpointsList != null && !endpointsList.isEmpty()) {
			logger
					.info("Iniciando inclusao de lista de declaraçoes de
endpoints.");
			for (CtEndpoint impl : endpointsList) {
				if (impl.getId() == null || impl.getId().trim().equals("")) {
					throw new OnixException("Endpoint with host '"
							+ impl.getHost()
							+ "' has a invalid ID declaration.");
				}

				if (!isIdValid(impl.getId())) {
					throw new OnixException("Endpoint '" + impl.getId()
							+ "' with two or more references in config
```

```
files"
                                                + ", choose other id.");
                                }

                                if (impl.getProtocol().equals(StProtocols.TCP)) {
                                        if (impl.getHost() == null) {
                                                throw new OnixException(
                                                                "TCP Endpoint '"
                                                                                + impl.getId()
                                                                                + "' needs HOST
declaration instead of ADDRESS.");
                                        }
                                } else {
                                        if (impl.getAddress() == null
                                                        || impl.getAddress().equals("")) {
                                                throw new OnixException(
                                                                "NOT TCP Endpoint '"
                                                                                + impl.getId()
                                                                                + "' needs ADDRESS
declaration instead of HOST.");
                                        }
                                }
                                logger.info("Endpoint: '" + impl.getId()
                                                + "' é valido e esta sendo incluido.");
                                endpoints.put(impl.getId(), impl);
                        }
                }
        }

        /**
         * Loads validators from file and validates if theirs IDs have already been
         * declared
         *
         * @param tranformers
         * @throws OnixException
         */
        private void loadValidators(List<CtValidatorImplementation> validatorsList)
                        throws OnixException {
                if (validatorsList != null && !validatorsList.isEmpty()) {
                        logger
                                        .info("Iniciando inclusao de lista de declaraçoes de
validators.");
                        for (CtValidatorImplementation impl : validatorsList) {
                                if (impl.getId() == null || impl.getId().trim().equals("")) {
                                        throw new OnixException("Validator with class '"
                                                        + impl.getClazz()
                                                        + "' has a invalid ID declaration.");
                                }

                                if (!isIdValid(impl.getId())) {
                                        throw new OnixException("Validator '" + impl.getId()
                                                        + "' with two or more references in config
```

```
files"
                                                + ", choose other id.");
                        }

                        this.isClassValid(impl.getClazz(), Validator.class);
                        logger.info("Validator: '" + impl.getId()
                                        + "' é valido e esta sendo incluido.");
                        validators.put(impl.getId(), impl);
                    }
                }
        }

        /**
         * Loads validators from file and validates if theirs IDs have already been
         * declared
         *
         * @param tranformers
         * @throws OnixException
         */
        private void loadExceptions(List<CtExceptionStrategyImpl> exceptionList)
                        throws OnixException {
                if (exceptionList != null && !exceptionList.isEmpty()) {
                        logger
                                        .info("Iniciando inclusao de lista de declaraçoes de
exceptionStrategies.");
                        for (CtExceptionStrategyImpl impl : exceptionList) {
                                if (impl.getId() == null || impl.getId().trim().equals("")) {
                                        throw new OnixException("Exception with class '"
                                                        + impl.getClazz()
                                                        + "' has a invalid ID declaration.");
                                }

                                if (!isIdValid(impl.getId())) {
                                        throw new OnixException("ExceptionStrategy " +
impl.getId()
                                                        + " with two or more references in config
files"
                                                        + ", choose other id.");
                                }

                                this.isClassValid(impl.getClazz(), ExceptionHandler.class);
                                logger.info("ExceptionStrategy: '" + impl.getId()
                                                + "' é valido e esta sendo incluido.");
                                exceptionStrategy.put(impl.getId(), impl);
                        }
                }
        }

        /**
         * Validate endpoints properties
         *
         * @param endpoint
```

```
     * @throws OnixException
     */
    private void validateEndpoints(List<CtEndpointDecl> endpoint)
                    throws OnixException {
            if (endpoint != null && !endpoint.isEmpty()) {
                    logger.info("Validando enpoints.");
                    for (CtEndpointDecl decl : endpoint) {
                            if (endpoints.get(decl.getId()) == null) {
                                    throw new OnixException("No declaration for endpoint: "
                                                    + decl.getId());
                            } else {
                                    inboundEndpoints.put(decl.getId(), endpoints.get(decl
                                                    .getId()));
                            }
                    }
            }
    }

    /**
     * Validate Validators properties
     *
     * @param validator
     * @throws OnixException
     */
    private void validateValidators(List<CtValidatorDecl> validator)
                    throws OnixException {
            if (validator != null && !validator.isEmpty()) {
                    logger.info("Validando validators.");
                    for (CtValidatorDecl decl : validator) {
                            if (validators.get(decl.getId()) == null) {
                                    throw new OnixException("No declaration for validator: "
                                                    + decl.getId());
                            }
                    }
            }
    }

    /**
     * Validate Enrichers properties
     *
     * @param enricher
     * @throws OnixException
     */
    private void validateEnrichers(List<CtEnricherDecl> enricher)
                    throws OnixException {
            if (enricher != null && !enricher.isEmpty()) {
                    logger.info("Validando enrichers.");
                    for (CtEnricherDecl decl : enricher) {
                            if (enrichers.get(decl.getId()) == null) {
                                    throw new OnixException("No declaration for Enricher: "
                                                    + decl.getId());
                            }
```

```
                }
            }
        }

        /**
         * Validate Exception properties
         *
         * @param decl
         * @throws OnixException
         */
        private void validateException(CtExceptionStrategyDecl decl)
                throws OnixException {
            if (decl != null) {
                logger.info("Validando Exception Strategy.");
                if (exceptionStrategy.get(decl.getId()) == null) {
                    throw new OnixException(
                            "No declaration for Exception strategy : "
                                    + decl.getId());
                }
            }

        }

        /**
         * Validate Routers properties
         *
         * @param routers
         * @throws OnixException
         */
        private void validateRouters(CtRouters routers) throws OnixException {
            logger.info("Validando Routers");
            if (routers.getContentBasedRouter() != null
                    && !routers.getContentBasedRouter().isEmpty()) {
                for (CtContentbasedRouter decl : routers.getContentBasedRouter()) {

                    this.isClassValid(decl.getClazz(), Router.class);

                    // Valida enpoint
                    if (endpoints.get(decl.getOutgoingEndpoint().getId()) == null) {
                        throw new OnixException(
                                "No declaration for outbound endpoint: "
                                        +
decl.getOutgoingEndpoint().getId());
                    } else {
                        if (inboundEndpoints
                                .get(decl.getOutgoingEndpoint().getId()) !=
null) {
                            throw new OnixException(
                                    "OutboundEndpoint '"
                                            +
decl.getOutgoingEndpoint().getId()
                                            + "' is already been
```

```
used as a inbound enpoint.");
                                        }

        outboundEndpoints.put(decl.getOutgoingEndpoint().getId(),

        endpoints.get(decl.getOutgoingEndpoint().getId()));
                                }

                                if (decl.getInboundTransformerId() != null
                                        && !
decl.getInboundTransformerId().trim().equals("")) {
                                        if (transformers.get(decl.getInboundTransformerId()) ==
null) {
                                                throw new OnixException(
                                                        "No declaration for inbound
transformer '"
                                                                +
decl.getInboundTransformerId() + "'.");
                                        }
                                }

                                if (decl.getOutboundTransformerId() != null
                                        && !
decl.getOutboundTransformerId().trim().equals("")) {
                                        if (transformers.get(decl.getOutboundTransformerId()) ==
null) {
                                                throw new OnixException(
                                                        "No declaration for Outbound
transformer '"
                                                                +
decl.getOutboundTransformerId()
                                                                + "'.");
                                        }
                                }

                        }
                } else {
                        // Caso nao tenha nenhuma declaracao de router de qualquer tipo,
                        // lança excecao
                        if (routers.getBroadcastingRouter() == null
                                        || routers.getBroadcastingRouter().isEmpty()) {
                                throw new OnixException(
                                        "You must declare at least one
contentbasedRouter or broadcastingRouter");
                        }
                }

                if (routers.getBroadcastingRouter() != null
                                || !routers.getBroadcastingRouter().isEmpty()) {
                        for (CtBroadcastingRouter decl : routers.getBroadcastingRouter()) {

                                // Valida enpoints do broadcasting
```

```
                        for (CtEndpointDecl endpoint : decl.getOutgoingEndpoint()) {
                                if (endpoints.get(endpoint.getId()) == null) {
                                        throw new OnixException(
                                                        "No declaration for outbound
endpoint: "
                                                                        + endpoint.getId());
                                } else {
                                        if (inboundEndpoints.get(endpoint.getId()) != null) {
                                                throw new OnixException(
                                                                "OutboundEndpoint '"
                                                                                +
endpoint.getId()
                                                                                + "' is already
been used as a inbound enpoint.");
                                        }
                                        outboundEndpoints.put(endpoint.getId(),
endpoints
                                                        .get(endpoint.getId()));
                                }
                        }
                        if (decl.getInboundTransformerId() != null
                                        && !
decl.getInboundTransformerId().trim().equals("")) {
                                if (transformers.get(decl.getInboundTransformerId()) ==
null) {
                                        throw new OnixException(
                                                        "No declaration for inbound
transformer '"
                                                                        +
decl.getInboundTransformerId() + "'.");
                                }
                        }
                }
        }

        }

        private boolean isIdValid(String id) {
                if (exceptionStrategy.get(id) != null) {
                        return false;
                } else if (enrichers.get(id) != null) {
                        return false;
                } else if (endpoints.get(id) != null) {
                        return false;
                } else if (validators.get(id) != null) {
                        return false;
                } else if (transformers.get(id) != null) {
                        return false;
                }
                return true;
        }
```

```
/**
 * This method validates if the parameters class path is valid
 *
 * @param clazz
 * @return
 */
private void isClassValid(String clazz, Class<?> type) throws OnixException {
	try {
		logger.info("Validando implementação da classe: " + clazz);
		if (!type.isAssignableFrom(Class.forName(clazz))) {
			throw new OnixException("Class '" + clazz
					+ "' does not extends " + type.getName());
		}
	} catch (ClassNotFoundException e) {
		throw new OnixException("Class '" + clazz + "' does not exists.");
	}
}

// TODO - USA JUNIT PRA FAZER ISSO!!!!
public static void main(String[] args) {
	try {

		String[] list = new String[1];
		list[0] = ConfigLoader.class.getResource("exemploConfig.xml")
				.getPath();
		OnixConfigLoader loader = new OnixConfigLoader();
		loader.loadConfigs(list);
	} catch (Exception e) {
		System.out.println("pau");
		e.printStackTrace();
	}
}

public Map<String, CtTransformerImplementation> getTransformers() {
	return transformers;
}

public void setTransformers(
		Map<String, CtTransformerImplementation> transformers) {
	this.transformers = transformers;
}

public Map<String, CtEnricherImplementation> getEnrichers() {
	return enrichers;
}

public void setEnrichers(Map<String, CtEnricherImplementation> enrichers) {
	this.enrichers = enrichers;
}

public Map<String, CtValidatorImplementation> getValidators() {
	return validators;
```

```
	}

	public void setValidators(Map<String, CtValidatorImplementation> validators) {
		this.validators = validators;
	}

	public Map<String, CtExceptionStrategyImpl> getExceptionStrategy() {
		return exceptionStrategy;
	}

	public void setExceptionStrategy(
			Map<String, CtExceptionStrategyImpl> exceptionStrategy) {
		this.exceptionStrategy = exceptionStrategy;
	}

	public Map<String, CtIntengrationChain> getIntegrationChain() {
		return integrationChain;
	}

	public void setIntegrationChain(
			HashMap<String, CtIntengrationChain> integrationChain) {
		this.integrationChain = integrationChain;
	}

	public Map<String, CtEndpoint> getInBoundEndpoints() {
		return inboundEndpoints;
	}

	public void setInBoundEndpoints(Map<String, CtEndpoint> inBoundEndpoints) {
		this.inboundEndpoints = inBoundEndpoints;
	}

	public Map<String, CtEndpoint> getOutboundEndpoints() {
		return outboundEndpoints;
	}

	public void setOutboundEndpoints(Map<String, CtEndpoint> outboundEndpoints) {
		this.outboundEndpoints = outboundEndpoints;
	}

}
```

Class EnricherException

```
package br.inf.ufsc.onix.exception;

/**
 * Exception thrown when a enricher throws any kind of Exception
 *
 * @author Pedro Castello Branco Galoppini<pgaloppini@gmail.com>
 *
 */
```

```
public class EnricherException extends OnixException {

	private static final long serialVersionUID = 1L;

	private String enricherId;

	public String getEnricherId() {
		return enricherId;
	}

	public void setEnricherId(String enricherId) {
		this.enricherId = enricherId;
	}

	public EnricherException() {
		super();
	}

	public EnricherException(Throwable cause) {
		super(cause);
	}

	public EnricherException(String message) {
		super(message);
	}
}
```

Class OnixEndpointException

```
package br.inf.ufsc.onix.exception;

/**
 * Excecao utilizada em casos que não é encontrado o endpoint destino para a
 * mensagem.
 *
 * @author Gabriel Nunes Menezes<bielmenezes@gmail.com>
 *
 * @since 22/01/2009
 */
public class OnixEndpointException extends OnixException {

	private String message = "Endpoint não encontrado.";

	public OnixEndpointException() {
	}

	public OnixEndpointException(Throwable cause) {
		super(cause);
	}

	public OnixEndpointException(String message) {
		this.message = message;
```

```
	}

	@Override
	public String getMessage() {
		return message;
	}
}
```

Class OnixException

```
package br.inf.ufsc.onix.exception;

/**
 * Excecao pai de todas as excecoes que podem ocorrer no framework.
 *
 * @author Gabriel Nunes Menezes<bielmenezes@gmail.com>
 *
 * @since 15/01/2009
 */
public class OnixException extends Exception {

	private static final long serialVersionUID = 1L;

	public OnixException() {
		super();
	}

	public OnixException(Throwable cause) {
		super(cause);
	}

	public OnixException(String message) {
		super(message);
	}

}
```

Class OnixInitializationException

```
package br.inf.ufsc.onix.exception;

/**
 * Excecao que representa erro durante inicializacoes.
 *
 * @author Gabriel Nunes Menezes<bielmenezes@gmail.com>
 *
 * @since 15/01/2009
 */
public class OnixInitializationException extends OnixException{

	public OnixInitializationException() {
	}
```

```
	public OnixInitializationException(Throwable cause) {
		super(cause);
	}
}
```

Class RoutingException

```
package br.inf.ufsc.onix.exception;

/**
 * Exception thrown when a router throws any kind of Exception
 *
 * @author Pedro Castello Branco Galoppini<pgaloppini@gmail.com>
 *
 */
public class RoutingException extends OnixException {
	private static final long serialVersionUID = 1L;

	public static final int CONTENT_BASED_ROUTER = 1;

	public static final int BROADCASTING_ROUTER = 2;

	private int routerType;

	private String outboundEndpointId;

	public RoutingException() {
		super();
	}

	public RoutingException(Throwable cause) {
		super(cause);
	}

	public RoutingException(String message) {
		super(message);
	}

	public String getOutboundEndpointId() {
		return outboundEndpointId;
	}

	public void setOutboundEndpointId(String outboundEndpointId) {
		this.outboundEndpointId = outboundEndpointId;
	}

	public int getRouterType() {
		return routerType;
	}

	public void setRouterType(int routerType) {
```

```
		this.routerType = routerType;
	}
}
```

Class TransformException

```
package br.inf.ufsc.onix.exception;

/**
 * Exception thrown when a transformer throws any kind of Exception
 *
 * @author Pedro Castello Branco Galoppini<pgaloppini@gmail.com>
 *
 */
public class TransformException extends OnixException {

	private static final long serialVersionUID = 1L;

	private String transformerId;

	public String getTransformerId() {
		return transformerId;
	}

	public void setTransformerId(String transformerId) {
		this.transformerId = transformerId;
	}

	public TransformException() {
		super();
	}

	public TransformException(Throwable cause) {
		super(cause);
	}

	public TransformException(String message) {
		super(message);
	}
}
```

Class ValidationException

```
package br.inf.ufsc.onix.exception;

/**
 * Exception thrown when a Validator throws any kind of Exception
 *
 * @author Pedro Castello Branco Galoppini<pgaloppini@gmail.com>
 *
 */
public class ValidationException extends OnixException {
```

```
	private static final long serialVersionUID = 1L;

	private String validatorId;

	public ValidationException() {
		super();
	}

	public ValidationException(Throwable cause) {
		super(cause);
	}

	public ValidationException(String message) {
		super(message);
	}

	public String getValidatorId() {
		return validatorId;
	}

	public void setValidatorId(String validatorId) {
		this.validatorId = validatorId;
	}
}
```

Class ClassUtils

```
package br.inf.ufsc.onix.util;

import org.apache.log4j.Logger;

/**
 *
 * @author Gabriel Nunes Menezes<bielmenezes@gmail.com>
 *
 * @since 22/01/2009
 */
public class ClassUtils {

	private static Logger logger = Logger.getLogger(ClassUtils.class);

	@SuppressWarnings("unchecked")
	public static Object instanciateClass(String className)
			throws InstantiationException, IllegalAccessException,
			ClassNotFoundException {
		return Class.forName(className).newInstance();
	}

	/**
	 * This method validates if the parameters class path is valid
	 *
```

```
	 * @param clazz
	 * @return
	 * @throws Exception
	 */
	public static void implementsOrExtends(String clazz, Class<?> type)
			throws Exception {
		logger.info("Validando implementação da classe: " + clazz);
		if (!Class.forName(clazz).isAssignableFrom(type)) {
			throw new Exception("Class '" + clazz
					+ "' does not extends nor implements " +
type.getName());
		}

	}
}
```

Class MessageConverter

```
package br.inf.ufsc.onix.util;

import java.util.HashMap;
import java.util.Map;

import br.inf.ufsc.onix.api.Message;
import br.inf.ufsc.onix.impl.OnixHeader;
import br.inf.ufsc.onix.impl.OnixMessage;

/**
 * Classe com utilidades para se realizar sobre as mensagens do framework.
 *
 * @author Gabriel Nunes Menezes<bielmenezes@gmail.com>
 *
 * @since 18/02/2009
 */
public class MessageConverter {

	public static Message objectToMessage(Object obj,
			Map<Object, Object> properties) {
		Message onixMessage = new OnixMessage();
		// setando o header
		onixMessage.setHeader(new OnixHeader());
		onixMessage.getHeader().setInboundEndpointUrl(null);

		if (properties == null) {
			properties = new HashMap<Object, Object>();
		}

		onixMessage.getHeader().setProperties(properties);

		// setando o body
		onixMessage.setBody(obj);
```

```
		// retornando mensagem criada
		return onixMessage;
	}

	public static Message objectToMessage(Object obj) {
		return objectToMessage(obj, null);
	}

}
```

Class AbstractMessageDispatcher

```
package br.inf.ufsc.onix.impl;

import org.apache.log4j.Logger;

import br.inf.ufsc.onix.api.Message;
import br.inf.ufsc.onix.api.MessageDispatcher;
import br.inf.ufsc.onix.api.OnixEngine;
import br.inf.ufsc.onix.api.ServiceRegistry;
import br.inf.ufsc.onix.config.types.CtIntengrationChain;
import br.inf.ufsc.onix.esb.mule.MuleMessageDispatcher;
import br.inf.ufsc.onix.exception.OnixEndpointException;
import br.inf.ufsc.onix.exception.OnixException;
import br.inf.ufsc.onix.util.MessageConverter;

/**
 * Implementaçao abstrata do MessageDispatcher do Onix.
 *
 * @author Gabriel Nunes Menezes<bielmenezes@gmail.com>
 *
 * @since 18/02/2009
 */
public abstract class AbstractMessageDispatcher implements MessageDispatcher {

	protected static Logger logger = Logger
			.getLogger(MuleMessageDispatcher.class);

	private ServiceRegistry registry;

	private OnixEngine onixEngine;

	public AbstractMessageDispatcher(OnixEngine onixEngine) {
		this.onixEngine = onixEngine;
		registry = onixEngine.getRegistry();
	}

	@Override
	public void dispatchMessage(String nameIntegrationChain, Object message)
			throws OnixEndpointException, OnixException {
		// TODO - Implementar
	}
```

```
	@Override
	public Object sendMessage(String nameIntegrationChain, Object message)
			throws OnixEndpointException, OnixException {
		logger.info("Despachando mensagem para o " + nameIntegrationChain);
		CtIntengrationChain integrationChain = registry
				.getServiceByName(nameIntegrationChain);
		if (integrationChain == null) {
			logger.warn("Integration chain com o nome " + nameIntegrationChain
					+ " não encontrado, abortando despacho.");
			throw new OnixEndpointException(
					"Não foi encontrado nenum integration chain com o
nome "
							+ nameIntegrationChain
							+ ", verifique se ele existe e tente
novamente.");
		}

		Message sendMessage = MessageConverter.objectToMessage(message);
		Message result = onixEngine.processMessage(sendMessage);

		return result.getBody();
	}

}
```

Class AbstractOnixEngine

```
package br.inf.ufsc.onix.impl;

import java.util.List;
import java.util.Map;

import org.apache.log4j.Logger;

import br.inf.ufsc.onix.api.Enricher;
import br.inf.ufsc.onix.api.ExceptionHandler;
import br.inf.ufsc.onix.api.Message;
import br.inf.ufsc.onix.api.OnixEngine;
import br.inf.ufsc.onix.api.Router;
import br.inf.ufsc.onix.api.ServiceRegistry;
import br.inf.ufsc.onix.api.Transformer;
import br.inf.ufsc.onix.api.Validator;
import br.inf.ufsc.onix.config.loader.OnixConfigLoader;
import br.inf.ufsc.onix.config.types.CtBroadcastingRouter;
import br.inf.ufsc.onix.config.types.CtContentbasedRouter;
import br.inf.ufsc.onix.config.types.CtEndpoint;
import br.inf.ufsc.onix.config.types.CtEndpointDecl;
import br.inf.ufsc.onix.config.types.CtEnricherDecl;
import br.inf.ufsc.onix.config.types.CtEnricherImplementation;
import br.inf.ufsc.onix.config.types.CtEnrichers;
import br.inf.ufsc.onix.config.types.CtExceptionStrategyDecl;
```

```
import br.inf.ufsc.onix.config.types.CtExceptionStrategyImpl;
import br.inf.ufsc.onix.config.types.CtIntengrationChain;
import br.inf.ufsc.onix.config.types.CtRouters;
import br.inf.ufsc.onix.config.types.CtTransformerImplementation;
import br.inf.ufsc.onix.config.types.CtValidatorDecl;
import br.inf.ufsc.onix.config.types.CtValidatorImplementation;
import br.inf.ufsc.onix.config.types.CtValidators;
import br.inf.ufsc.onix.exception.EnricherException;
import br.inf.ufsc.onix.exception.OnixEndpointException;
import br.inf.ufsc.onix.exception.OnixException;
import br.inf.ufsc.onix.exception.OnixInitializationException;
import br.inf.ufsc.onix.exception.RoutingException;
import br.inf.ufsc.onix.exception.TransformException;
import br.inf.ufsc.onix.exception.ValidationException;
import br.inf.ufsc.onix.util.ClassUtils;
import br.inf.ufsc.onix.util.MessageConverter;

/**
 * Implementação abstrata da interface OnixEngine contendo as regras basicas
 * para utilizacao do framework.
 *
 * @author Gabriel Nunes Menezes<bielmenezes@gmail.com>
 * @author Pedro Castello Branco Galoppini<pgaloppini@gmail.com>
 *
 * @since 15/01/2009
 */
public abstract class AbstractOnixEngine implements OnixEngine {

	protected ServiceRegistry registry;

	protected Map<String, CtTransformerImplementation> transformers;

	protected Map<String, CtEnricherImplementation> enrichers;

	protected Map<String, CtValidatorImplementation> validators;

	protected Map<String, CtExceptionStrategyImpl> exceptionStrategy;

	protected Map<String, CtEndpoint> inboundEndpoints;

	protected Map<String, CtEndpoint> outboundEndpoints;

	private Logger logger = Logger.getLogger(this.getClass());

	@Override
	public void registerIntegrationChain(String endpoint,
			CtIntengrationChain integrationChain) {
		if (endpoint == null || endpoint.equals("")) {
			throw new IllegalArgumentException(
					"Url do endpoint deve ser diferente de nulo e nao pode
ser vazia.");
		}
```

```
		if (integrationChain == null) {
			throw new IllegalArgumentException(
					"Integration Chain não pode ser nulo.");
		}

		registry.register(endpoint, integrationChain);
	}

	@Override
	public Message processMessage(Message onixMessage) throws OnixException {
		String inboundEndpointUrl = onixMessage.getHeader()
				.getInboundEndpointUrl();
		CtIntengrationChain chain = registry
				.getServiceByEndpoint(inboundEndpointUrl);

		if (chain == null) {
			logger
					.warn("IntegrationChain recebido é nulo, abortando
processamento..");
			throw new OnixException(
					"A mensagem recebida não possui um IntegrationChain
configurado para tratá-la. InboundEndpoint = "
							+
onixMessage.getHeader().getInboundEndpointUrl());
		}

		return processMessage(onixMessage, chain);
	}

	public Message processMessage(Message onixMessage, String chainName)
			throws OnixException {
		CtIntengrationChain chain = registry.getServiceByName(chainName);
		if (chain == null) {
			logger
					.warn("IntegrationChain recebido é nulo, abortando
processamento..");
			throw new OnixException(
					"A mensagem recebida não possui um IntegrationChain
configurado para tratá-la. InboundEndpoint = "
							+
onixMessage.getHeader().getInboundEndpointUrl());
		}

		return processMessage(onixMessage, chain);
	}

	public Message processMessage(Message onixMessage, CtIntengrationChain
chain)
			throws OnixException {
		logger.info("Mensagem recebida na engine, iniciando processamento..");
```

```
		if (chain == null) {
			logger
					.warn("IntegrationChain recebido é nulo, abortando
processamento..");
			throw new OnixException(
					"O IntegrationChain enviado não pode ser null.");
		}

		logger.info("Processando mensagem no chain " + chain.getName());

		try {
			// send
			this.executeValidations(chain.getSend().getValidators(),
					onixMessage);
			onixMessage = this.executeEnrichers(chain.getSend().getEnrichers(),
					onixMessage);

			onixMessage = this.routeContentBasedMessage(chain.getSend()
					.getRouters(), onixMessage);

			this.broadcastMessage(chain.getSend().getRouters(), onixMessage);

			// receive
			if (chain.getReceive() != null) {
				this.executeValidations(chain.getReceive().getValidators(),
						onixMessage);
				onixMessage = this.executeEnrichers(chain.getReceive()
						.getEnrichers(), onixMessage);

			}
		} catch (OnixException t) {
			logger
					.warn(
							"Erro ao executar integration chain,
passando para Exception Strategy",
							t);

			try {
				onixMessage = this.executeExceptionStrategy(chain
						.getExceptionStrategy(), onixMessage, t);
			} catch (Exception e) {
				logger.fatal("Erro Fatal ao executar excepion strategy: '"
						+ chain.getExceptionStrategy().getId(), e);
			}

		} catch (Exception e) {
			logger.fatal("Erro inesperado ao executar integration chain: '"
					+ chain.getName(), e);
		}

		logger
				.info("Processamento da mensagem realizado com sucesso,
```

```
retornando ao cliente.");
                return onixMessage;
        }

        @Override
        public CtIntengrationChain getIntegrationChain(String urlEndpoint) {
                return registry.getServiceByEndpoint(urlEndpoint);
        }

        @Override
        public void loadXmlConfig(String... xmlFile) throws OnixException {
                try {
                        logger.info("Carregando configurações do framework..");

                        OnixConfigLoader configLoader = new OnixConfigLoader();
                        configLoader.loadConfigs(xmlFile);

                        validators = configLoader.getValidators();
                        enrichers = configLoader.getEnrichers();
                        transformers = configLoader.getTransformers();
                        exceptionStrategy = configLoader.getExceptionStrategy();
                        inboundEndpoints = configLoader.getInBoundEndpoints();
                        outboundEndpoints = configLoader.getOutboundEndpoints();

                        // registra no ESB os endpoints que estarão ativos..
                        logger
                                        .info("Configurações carregadas com sucesso,
registrando endpoints no ESB..");
                        registerEndpointListeners(configLoader.getInBoundEndpoints());
                        registerOutBoundEndpoints(configLoader.getOutboundEndpoints());
                        logger.info("Regitro de endpoints realizado com sucesso.");

                        // registra na engine os integrations chains configurados no xml
                        logger.info("Registrando integrationsChains no framework..");
                        registerIntegrationsChains(configLoader.getIntegrationChain());
                        logger.info("IntegrationsChains registrados com sucesso.");
                } catch (Exception e) {
                        OnixException oex = new OnixException(
                                        "Erro ao carregar arquivos de configuração do
framework.");
                        oex.initCause(e);
                        throw oex;
                }
        }

        /**
         * Registra os integrations Chains no registry.
         *
         * @param integrationChains
         */
        protected void registerIntegrationsChains(
                        Map<String, CtIntengrationChain> integrationChains) {
```

```
		if (integrationChains == null) {
			throw new IllegalArgumentException(
					"The value of integrationChains cannot be null.");
		}

		// para cada chain, lista os endpoints de entrada e registra no
		// registry
		for (CtIntengrationChain chain : integrationChains.values()) {
			List<CtEndpointDecl> endpointsDecl = chain.getSend().getIncoming()
					.getEndpoint();
			for (CtEndpointDecl endpointDecl : endpointsDecl) {
				CtEndpoint endpoint = inboundEndpoints
						.get(endpointDecl.getId());
				if (endpoint != null) {
					// registra
					registerIntegrationChain(getEndpointBuilder()
							.getEndpointAddress(endpoint), chain);
				}

			}

		}

	}

	/**
	 * Registra os endpoints no ESB.
	 *
	 * @param endpoints
	 * @throws OnixEndpointException
	 * @throws OnixInitializationException
	 */
	protected void registerEndpointListeners(Map<String, CtEndpoint> endpoints)
			throws OnixInitializationException, OnixEndpointException {
		for (CtEndpoint endpoint : endpoints.values()) {
			getEndpointBuilder().buildEsbInboundEndpoint(endpoint);
		}
	}

	/**
	 * Registra os outbound endpoints no ESB.
	 *
	 * @param endpoints
	 * @throws OnixEndpointException
	 * @throws OnixInitializationException
	 */
	protected void registerOutBoundEndpoints(Map<String, CtEndpoint> endpoints)
			throws OnixInitializationException, OnixEndpointException {
		for (CtEndpoint endpoint : endpoints.values()) {
			getEndpointBuilder().buildEsbOutboundEndpoint(endpoint);
		}
	}
```

```
	private Message executeExceptionStrategy(
			CtExceptionStrategyDecl exStrategy, Message onixMessage,
Throwable e)
			throws Exception {

		CtExceptionStrategyImpl impl = this.exceptionStrategy.get(exStrategy
				.getId());
		ExceptionHandler exHandler = (ExceptionHandler) Class.forName(
				impl.getClazz()).newInstance();
		logger.debug("Executando exception strategy: " + impl.getId());

		Object payLoad = exHandler.handleException(e, onixMessage);

		return MessageConverter.objectToMessage(payLoad);
	}

	/**
	 * Realiza roteamento da mensagem e transformação caso seja necessário e
	 * esteja declarado dos roteadores baseados no conteudo
	 *
	 * @param routers
	 * @param onixMessage
	 * @return
	 */
	private Message routeContentBasedMessage(CtRouters routers,
			Message onixMessage) throws OnixException {
		// TODO router deve analisar a mensagem transformada ou mensagem
normal?
		for (CtContentbasedRouter content : routers.getContentBasedRouter()) {
			if (content.getInboundTransformerId() != null) {
				try {
					logger.warn("Executando transformer de entrada: "
							+ content.getInboundTransformerId());

					Transformer trans = getTransformer(content
							.getInboundTransformerId());
					if (trans == null) {
						throw new Exception("Erro ao instanciar
transformer "
								+
content.getInboundTransformerId());
					}
					Object resp = trans.doTransform(onixMessage);

					if (resp instanceof OnixMessage) {
						onixMessage = (Message) resp;
					} else {
						onixMessage =
MessageConverter.objectToMessage(resp);
					}
				} catch (Exception e) {
```

```
                                        logger
                                                .warn("Exceção ocorrida ao executar
transformer de entrada: "
                                                        +
content.getInboundTransformerId());
                                        TransformException ex = new TransformException(e);
                                        ex.setTransformerId(content.getInboundTransformerId());
                                        throw ex;
                                }
                        }
                        Message returnMessage = null;

                        try {
                                Router router = (Router) ClassUtils.instanciateClass(content
                                                .getClazz());

                                // TODO colocar as properties dos routers
                                // Verifica se deve aceitar a mensagem para este router
                                if (!router.acceptMessage(onixMessage)) {
                                        continue;
                                }

                                returnMessage =
this.sendMessage(content.getOutgoingEndpoint(),
                                                onixMessage);
                        } catch (Throwable e) {
                                logger
                                                .warn("Exceção ocorrida ao fazer roteamento para
saida no endpoint: "
                                                                +
content.getOutgoingEndpoint().getId());
                                RoutingException ex = new RoutingException(e);

        ex.setOutboundEndpointId(content.getOutgoingEndpoint().getId());

        ex.setRouterType(RoutingException.CONTENT_BASED_ROUTER);
                                throw ex;
                        }

                        // Transforma a volta
                        if (content.getOutboundTransformerId() != null) {
                                try {
                                        logger.warn("Executando transformer de saida: "
                                                        + content.getOutboundTransformerId());
                                        Transformer trans = getTransformer(content
                                                        .getOutboundTransformerId());
                                        if (trans == null) {
                                                throw new Exception("Erro ao instanciar
transformer "
                                                                +
content.getOutboundTransformerId());
                                        }
```

```
                                        Object resp = trans.doTransform(returnMessage);
                                        returnMessage =
MessageConverter.objectToMessage(resp);
                                } catch (Exception e) {
                                        logger
                                                        .warn("Exceção ocorrida ao executar
transformer de saida: "
                                                                        +
content.getInboundTransformerId());
                                        TransformException ex = new TransformException(e);
                                        ex.setTransformerId(content.getInboundTransformerId());
                                        throw ex;
                                }
                        }

                        return returnMessage;
                }
                return onixMessage;
        }

        /**
         * Roteia a mensagem nos roateadores Broadcasting
         *
         * @param routers
         * @param onixMessage
         * @throws Exception
         */
        private void broadcastMessage(CtRouters routers, Message onixMessage)
                        throws Exception {
                for (CtBroadcastingRouter broad : routers.getBroadcastingRouter()) {
                        if (broad.getInboundTransformerId() != null) {
                                try {
                                        logger.warn("Executando transformer de entrada: "
                                                        + broad.getInboundTransformerId());
                                        CtTransformerImplementation impl =
transformers.get(broad
                                                        .getInboundTransformerId());
                                        Transformer trans = (Transformer) ClassUtils
                                                        .instanciateClass(impl.getClazz());
                                        Object resp = trans.doTransform(onixMessage);

                                        if (resp instanceof OnixMessage) {
                                                onixMessage = (Message) resp;
                                        } else {
                                                onixMessage =
MessageConverter.objectToMessage(resp);
                                        }
                                } catch (Exception e) {
                                        logger
                                                        .warn("Exceção ocorrida ao executar
transformer de entrada: "
                                                                        +
```

```
broad.getInboundTransformerId());
                                        TransformException ex = new TransformException(e);
                                        ex.setTransformerId(broad.getInboundTransformerId());
                                        throw ex;
                                }
                        }

                        for (CtEndpointDecl endpoint : broad.getOutgoingEndpoint()) {
                                try {
                                        this.dispatchMessage(endpoint, onixMessage);
                                } catch (Throwable e) {

                                        logger
                                                        .warn("Exceção ocorrida ao fazer
roteamento para saida no endpoint: "
                                                                        + endpoint.getId());
                                        RoutingException ex = new RoutingException(e);
                                        ex.setOutboundEndpointId(endpoint.getId());

        ex.setRouterType(RoutingException.BROADCASTING_ROUTER);
                                        throw ex;

                                }
                        }

                }
        }

        /**
         * Metodo realiza o envio da mensagem para um endpoint de acordo com o
         * router
         *
         * @param contentBasedRouter
         * @param onixMessage
         * @return
         */
        private Message sendMessage(CtEndpointDecl decl, Message onixMessage)
                        throws Exception {

                String address = getEndpointBuilder().getEndpointAddress(
                                outboundEndpoints.get(decl.getId()));

                Object resp = getMessageDispatcher().sendMessageToEsb(address,
                                onixMessage.getBody());
                return MessageConverter.objectToMessage(resp);
        }

        /**
         * Metodo despacha a mensagem para um endpoint de acordo com o router
         *
         * @param contentBasedRouter
         * @param onixMessage
```

```
	 * @return
	 */
	private void dispatchMessage(CtEndpointDecl decl, Message onixMessage)
			throws Exception {
		String address = getEndpointBuilder().getEndpointAddress(
				outboundEndpoints.get(decl.getId()));

		getMessageDispatcher().dispatchMessageToEsb(address,
				onixMessage.getBody());
	}

	/**
	 * Metodo Executa os enriquecimentos que devem ser feitos na mensagem
	 * validada
	 *
	 * @param enrichers
	 */
	private Message executeEnrichers(CtEnrichers enrichers, Message message)
			throws EnricherException {
		if (enrichers == null) {
			logger
					.info("Nenhum enricher para executar, abortando
enriquecimento..");
			return message;
		}

		Object payLoad = null;
		for (CtEnricherDecl decl : enrichers.getEnricher()) {
			try {
				Enricher enricher = getEnricher(decl.getId());
				if (enricher == null) {
					throw new Exception("Erro ao instanciar enricher "
							+ decl.getId());
				}
				logger.debug("Executando enriquecimento: " + decl.getId());
				payLoad = enricher.doEnrich(message);

				if (payLoad instanceof OnixMessage) {
					message = (Message) payLoad;
				} else {
					message =
MessageConverter.objectToMessage(payLoad);
				}
			} catch (Exception e) {
				logger.warn("Exceção ocorrida ao executar enricher: "
						+ decl.getId());
				EnricherException ex = new EnricherException(e);
				ex.setEnricherId(decl.getId());
				throw ex;
			}
		}
		return message;
```

```
	}

	/**
	 * Metodo executa as validações que devem ser feitas em cima da mensagem de
	 * entrada
	 *
	 * @param validators
	 */
	private void executeValidations(CtValidators validators, Message message)
			throws ValidationException {
		if (validators == null) {
			logger
					.info("Nenhum validator para executar, abortando
validação..");
			return;
		}

		for (CtValidatorDecl decl : validators.getValidator()) {
			try {

				Validator validator = getValidator(decl.getId());
				if (validator == null) {
					throw new Exception("Erro ao instanciar validator "
							+ decl.getId());
				}
				logger.debug("Executando validação: " + decl.getId());
				validator.doValidate(message);
			} catch (Exception e) {
				logger.warn("Exceção ocorrida ao executar validator: "
						+ decl.getId());
				ValidationException ex = new ValidationException(e);
				ex.setValidatorId(decl.getId());
				throw ex;
			}
		}
	}

	@Override
	public Validator getValidator(String id) {

		if (id == null || id.equals("")) {
			throw new IllegalArgumentException(
					"The value of validator id cannot be null nor empty.");
		}

		if (validators == null) {
			return null;
		}

		CtValidatorImplementation impl = this.validators.get(id);
		if (impl == null) {
```

```
                logger.warn("Validator " + id
                        + " not found at xml configuration file.");
                return null;
        }
        try {
                return (Validator) ClassUtils.instanciateClass(impl.getClazz());
        } catch (Exception e) {
                logger.fatal("Error while getting validator " + id
                        + ". Returning null.", e);
        }
        return null;
}

@Override
public Enricher getEnricher(String id) {
        if (id == null || id.equals("")) {
                throw new IllegalArgumentException(
                        "The value of enricher id cannot be null nor empty.");
        }

        if (enrichers == null) {
                return null;
        }

        CtEnricherImplementation enrichImpl = enrichers.get(id);
        if (enrichImpl == null) {
                logger.warn("Enricher " + id
                        + " not found at xml configuration file.");
                return null;
        }

        try {
                return (Enricher) ClassUtils
                        .instanciateClass(enrichImpl.getClazz());
        } catch (Exception e) {
                logger.fatal("Error while getting validator " + id
                        + ". Returning null.", e);
        }

        return null;
}

@Override
public Transformer getTransformer(String id) {
        if (id == null || id.equals("")) {
                throw new IllegalArgumentException(
                        "The value of tranformer id cannot be null nor empty.");
        }

        if (transformers == null) {
                return null;
        }
```

```
                CtTransformerImplementation transformer = transformers.get(id);
                if (transformer == null) {
                        logger.warn("Tranformer " + id
                                        + " not found at xml configuration file.");
                        return null;
                }

                try {
                        return (Transformer) ClassUtils.instanciateClass(transformer
                                        .getClazz());
                } catch (Exception e) {
                        logger.fatal("Error while getting transformer " + id
                                        + ". Returning null.", e);
                }

                return null;
        }

        public ServiceRegistry getRegistry() {
                return registry;
        }

        public void setRegistry(ServiceRegistry registry) {
                this.registry = registry;
        }

}
```

Class DefaultServiceRegistry

```
package br.inf.ufsc.onix.impl;

import java.util.HashMap;
import java.util.Map;

import org.apache.log4j.Logger;

import br.inf.ufsc.onix.api.ServiceRegistry;
import br.inf.ufsc.onix.config.types.CtIntengrationChain;

/**
 * Implementação padrão do ServiceRegistry do framework.
 *
 * @author Gabriel Nunes Menezes<bielmenezes@gmail.com>
 *
 * @since 15/01/2009
 */
public class DefaultServiceRegistry implements ServiceRegistry {

        private Map<String, CtIntengrationChain> endpointRegistry = new
HashMap<String, CtIntengrationChain>();
```

```
	private Map<String, CtIntengrationChain> nameRegistry = new HashMap<String,
CtIntengrationChain>();

	protected Logger logger = Logger.getLogger(this.getClass());

	@Override
	public boolean hasService(String endpoint) {
		return endpointRegistry.keySet().contains(endpoint) ||
nameRegistry.keySet().contains(endpoint) ;
	}

	@Override
	public void register(String endpoint, CtIntengrationChain processorChain) {
		logger.info("Registrando integration chain no endpoint " + endpoint);
		endpointRegistry.put(endpoint, processorChain);
		nameRegistry.put(processorChain.getName(), processorChain);
	}

	@Override
	public CtIntengrationChain getServiceByEndpoint(String endpoint) {
		CtIntengrationChain chain = endpointRegistry.get(endpoint);
		return chain;
	}

	@Override
	public CtIntengrationChain getServiceByName(String name) {
		CtIntengrationChain chain = endpointRegistry.get(name);
		return chain;
	}

}
```

Class OnixContext

```
package br.inf.ufsc.onix.impl;

import java.util.HashMap;
import java.util.Map;

/**
 * Contém atributos utilizados pelo contexto do framework
 * durante a execução de uma Thread.
 *
 * @author Gabriel Nunes Menezes<bielmenezes@gmail.com>
 *
 * @since 26/12/2008
 */
public class OnixContext {

	private static ThreadLocal<Map<String, Object>> context;
	static {
```

```
		context = new ThreadLocal<Map<String,Object>>();
	}

	public static void put(String key, Object value){
		Map<String, Object> attributes = context.get();
		if(attributes == null){
			context.set(new HashMap<String, Object>());
			attributes = context.get();
		}

		attributes.put(key, value);
	}

	public static Object get(String key){
		Map<String, Object> attributes = context.get();
		if(attributes == null){
			return null;
		}

		return attributes.get(key);
	}

	/**
	 * Invalida a thread local.
	 */
	public static void invalidate() {
		context.set(new HashMap<String, Object>());
	}

}
```

Class OnixHeader

```
package br.inf.ufsc.onix.impl;

import java.io.Serializable;
import java.util.Map;

import br.inf.ufsc.onix.api.Header;

/**
 *
 * @author Gabriel Nunes Menezes<bielmenezes@gmail.com>
 *
 * @since 29/11/2008
 */
public class OnixHeader implements Header {

	private static final long serialVersionUID = -7256476618059490076L;

	private String inboundEndpointUrl;
```

```
	private Map<Object, Object> properties;

	/* (non-Javadoc)
	 * @see br.inf.ufsc.onix.impl.Header#getInboundEndpointUrl()
	 */
	public String getInboundEndpointUrl() {
		return inboundEndpointUrl;
	}
	/* (non-Javadoc)
	 * @see br.inf.ufsc.onix.impl.Header#setInboundEndpointUrl(java.lang.String)
	 */
	public void setInboundEndpointUrl(String inboundEndpointUrl) {
		this.inboundEndpointUrl = inboundEndpointUrl;
	}
	/* (non-Javadoc)
	 * @see br.inf.ufsc.onix.impl.Header#getProperties()
	 */
	public Map<Object, Object> getProperties() {
		return properties;
	}
	/* (non-Javadoc)
	 * @see br.inf.ufsc.onix.impl.Header#setProperties(java.util.Map)
	 */
	public void setProperties(Map<Object, Object> properties) {
		this.properties = properties;
	}


}
```

Class OnixMessage

```
package br.inf.ufsc.onix.impl;

import java.io.Serializable;

import br.inf.ufsc.onix.api.Header;
import br.inf.ufsc.onix.api.Message;

/**
 *
 * @author Gabriel Nunes Menezes<bielmenezes@gmail.com>
 *
 * @since 29/11/2008
 */
public class OnixMessage implements Message {


	private static final long serialVersionUID = 3540519320260053649L;

	private Header header;
```

```
	private Object body;

	/* (non-Javadoc)
	 * @see br.inf.ufsc.onix.impl.Message#getHeader()
	 */
	public Header getHeader() {
		return header;
	}
	/* (non-Javadoc)
	 * @see br.inf.ufsc.onix.impl.Message#setHeader(br.inf.ufsc.onix.impl.Header)
	 */
	public void setHeader(Header header) {
		this.header = header;
	}
	/* (non-Javadoc)
	 * @see br.inf.ufsc.onix.impl.Message#getBody()
	 */
	public Object getBody() {
		return body;
	}
	/* (non-Javadoc)
	 * @see br.inf.ufsc.onix.impl.Message#setBody(java.lang.Object)
	 */
	public void setBody(Object body) {
		this.body = body;
	}


}
```

Class InitialOnixMessageBuilderUmo

```
package br.inf.ufsc.onix.esb.mule;

import org.apache.log4j.Logger;
import org.mule.api.MuleEventContext;
import org.mule.api.lifecycle.Callable;

import br.inf.ufsc.onix.api.Message;
import br.inf.ufsc.onix.util.MessageConverter;

/**
 * Componente que cria a mensagem inicial que entrará no IntegrationChain.
 *
 * @author Gabriel Nunes Menezes<bielmenezes@gmail.com>
 *
 * @since 15/01/2009
 */
public class InitialOnixMessageBuilderUmo implements Callable {

	public static final String COMPONENT_NAME = "InitialOnixMessageBuilder";
```

```
	protected Logger logger = Logger.getLogger(this.getClass());

	@Override
	public Object onCall(MuleEventContext eventContext) throws Exception {
		logger.info("Mensagem recebida no componente " + COMPONENT_NAME
				+ ", criando OnixMessage.");
		Message onixMessage =
MessageConverter.objectToMessage(eventContext.getMessage().getPayload());

		String endpoint = eventContext.getEndpointURI().toString();
		onixMessage.getHeader().setInboundEndpointUrl(endpoint);

		logger.info("Mensagem criada com sucesso, despachando para ser "
				+ "enviada ao integration chain respectivo.");

		return onixMessage;
	}
}
```

Class MuleEndpointBuilder

```
package br.inf.ufsc.onix.esb.mule;

import org.apache.log4j.Logger;
import org.mule.MuleServer;
import org.mule.api.MuleContext;
import org.mule.api.endpoint.EndpointException;
import org.mule.api.endpoint.InboundEndpoint;
import org.mule.api.endpoint.OutboundEndpoint;
import org.mule.api.lifecycle.InitialisationException;
import org.mule.api.registry.RegistrationException;
import org.mule.api.registry.Registry;
import org.mule.api.routing.InboundRouterCollection;
import org.mule.api.routing.OutboundRouterCollection;
import org.mule.api.service.Service;
import org.mule.endpoint.EndpointURIEndpointBuilder;
import org.mule.endpoint.URIBuilder;
import org.mule.model.seda.SedaService;
import org.mule.routing.outbound.OutboundPassThroughRouter;

import br.inf.ufsc.onix.api.EndpointBuilder;
import br.inf.ufsc.onix.config.types.CtEndpoint;
import br.inf.ufsc.onix.config.types.StProtocols;
import br.inf.ufsc.onix.exception.OnixEndpointException;
import br.inf.ufsc.onix.exception.OnixInitializationException;

/**
 * Classe com regras especificas do mule para a criacao de endpoints.
 *
 * @author Gabriel Nunes Menezes<bielmenezes@gmail.com>
 *
 * @since 15/01/2009
```

```
 */
public class MuleEndpointBuilder implements EndpointBuilder {

	protected Logger logger = Logger.getLogger(this.getClass());

	@Override
	public void buildEsbInboundEndpoint(CtEndpoint endpoint)
			throws OnixInitializationException, OnixEndpointException {
		try {
			logger.info("Construindo endpoint identificando por "
					+ endpoint.getId());

			// pega o contexto
			MuleContext context = MuleServer.getMuleContext();

			// pega o registry de servicos
			Registry registry = context.getRegistry();

			// procura o respectivo service
			Service service = (Service) registry
					.lookupObject(InitialOnixMessageBuilderUmo.COMPON
ENT_NAME);

			// construi o address do endpoint
			String address = getEndpointAddress(endpoint);

			// constroi o endpoint
			EndpointURIEndpointBuilder euebuilder = new
EndpointURIEndpointBuilder(
					new URIBuilder(address), context);
			InboundEndpoint inboundEndpoint =
euebuilder.buildInboundEndpoint();
			InboundRouterCollection collection = service.getInboundRouter();
			collection.addEndpoint(inboundEndpoint);

			// seta o inbound no service
			service.setInboundRouter(collection);
		} catch (InitialisationException e) {
			logger.error("Erro ao criar endpoint identificando por "
					+ endpoint.getId());
			OnixInitializationException exception = new
OnixInitializationException(
					e);
			throw exception;
		} catch (EndpointException e) {
			logger.error("Erro ao criar endpoint identificando por "
					+ endpoint.getId());
			OnixEndpointException exception = new OnixEndpointException(e);
			throw exception;
		}
	}
```

```
	@Override
	public void buildEsbOutboundEndpoint(CtEndpoint endpoint)
			throws OnixInitializationException, OnixEndpointException {
		try {
			logger.info("Construindo oubound endpoint identificando por "
					+ endpoint.getId());

			// pega o contexto
			MuleContext context = MuleServer.getMuleContext();

			// pega o registry de servicos
			Registry registry = context.getRegistry();

			// procura o respectivo service
			Service service = new SedaService();
			service.setName(endpoint.getId());

			Service serviceInbound = (Service) registry
					.lookupObject(InitialOnixMessageBuilderUmo.COMPON
ENT_NAME);

			service.setModel(serviceInbound.getModel());

			// construi o address do endpoint
			String address = getEndpointAddress(endpoint);

			// constroi o endpoint
			EndpointURIEndpointBuilder euebuilder = new
EndpointURIEndpointBuilder(
					new URIBuilder(address), context);
			OutboundEndpoint outboundEndpoint = euebuilder
					.buildOutboundEndpoint();

			// Faz o router
			OutboundPassThroughRouter router = new
OutboundPassThroughRouter();

			// Seta o endpoint no router
			router.addEndpoint(outboundEndpoint);
			OutboundRouterCollection collection = service.getOutboundRouter();
			collection.addRouter(router);

			// seta o router no service
			service.setOutboundRouter(collection);
			registry.registerObject(endpoint.getId(), service);
		} catch (InitialisationException e) {
			logger.error("Erro ao criar oubound endpoint identificando por "
					+ endpoint.getId(), e);
			OnixInitializationException exception = new
OnixInitializationException(
					e);
			throw exception;
```

```
			} catch (EndpointException e) {
				logger.error("Erro ao criar endpoint identificando por "
						+ endpoint.getId(), e);
				OnixEndpointException exception = new OnixEndpointException(e);
				throw exception;
			} catch (RegistrationException e) {
				logger.error(
						"Erro ao registrar service criado " + endpoint.getId(), e);
				OnixEndpointException exception = new OnixEndpointException(e);
				throw exception;
			}
		}

		/**
		 * TODO - Criar padrao strategy com as regras para criar endereco a partir
		 * do protocolo.
		 *
		 * @param endpoint
		 * @return
		 */
		public String getEndpointAddress(CtEndpoint endpoint) {
			if (endpoint.getAddress() != null && !endpoint.getAddress().equals("")) {
				String address = "";
				if (endpoint.getProtocol().equals(StProtocols.FILE)) {
					address = endpoint.getProtocol().toString().toLowerCase()
							+ "://" + endpoint.getAddress();
				} else if (endpoint.getProtocol().equals(StProtocols.HTTP)) {
					address = endpoint.getAddress();
				} else if (endpoint.getProtocol().equals(StProtocols.EMAIL)) {
					address = endpoint.getAddress();
				} else {
					address = endpoint.getProtocol().toString().toLowerCase() + ":"
							+ endpoint.getAddress();
				}
				return address;
			}

			return endpoint.getProtocol().toString().toLowerCase()
					+ "://"
					+ endpoint.getHost().getHost()
					+ (endpoint.getHost().getPort() != "" ? ":"
							+ endpoint.getHost().getPort() : "");
		}
}
```

Class MuleMessageConverter

```
package br.inf.ufsc.onix.esb.mule;

import org.mule.DefaultMuleMessage;
import org.mule.api.MuleMessage;
```

```
import br.inf.ufsc.onix.api.Header;
import br.inf.ufsc.onix.api.Message;
import br.inf.ufsc.onix.impl.OnixMessage;

/**
 * Classe utilitaria criada para converter objetos em mensagens do Onix.
 *
 * @author Gabriel Nunes Menezes<bielmenezes@gmail.com>
 *
 * @since 22/01/2009
 */
public class MuleMessageConverter {

	/**
	 * Converte uma MuleMessagem em uma OnixMessage.
	 *
	 * @param message
	 * @param header
	 * @return
	 */
	public Message convertMuleMessage(MuleMessage message, Header header) {
		Message onixMessage = new OnixMessage();
		onixMessage.setHeader(header);
		onixMessage.setBody(message.getPayload());

		return onixMessage;

	}

	/**
	 * Converte OnixMessage em MuleMessage
	 *
	 * @param onixMessage
	 * @return
	 */
	public MuleMessage convertOnixMessage(Message onixMessage) {
		DefaultMuleMessage message = new DefaultMuleMessage(onixMessage
				.getBody());
		return message;

	}

}
```

Class MuleMessageDispatcher

```
package br.inf.ufsc.onix.esb.mule;

import org.apache.log4j.Logger;
import org.mule.DefaultMuleMessage;
import org.mule.module.client.MuleClient;
```

```
import br.inf.ufsc.onix.api.OnixEngine;
import br.inf.ufsc.onix.exception.OnixException;
import br.inf.ufsc.onix.impl.AbstractMessageDispatcher;

/**
 * Implementação para o Mule ESB do Message dispatcher do Onix.
 *
 * @author Gabriel Nunes Menezes<bielmenezes@gmail.com>
 *
 * @since 22/01/2009
 */
public class MuleMessageDispatcher extends AbstractMessageDispatcher {

	protected static Logger logger = Logger
			.getLogger(MuleMessageDispatcher.class);

	public MuleMessageDispatcher(OnixEngine onixEngine) throws OnixException {
		super(onixEngine);
	}

	@Override
	public void dispatchMessageToEsb(String url, Object message)
			throws Exception {
		DefaultMuleMessage muleMessage = new DefaultMuleMessage(message);
		MuleClient client = new MuleClient();
		client.dispatch(url, muleMessage);

	}

	@Override
	public Object sendMessageToEsb(String url, Object message) throws Exception {
		DefaultMuleMessage muleMessage = new DefaultMuleMessage(message);
		MuleClient client = new MuleClient();
		Object ret = client.send(url, muleMessage);
		return ret;
	}

}
```

Class MuleOnixEngine

```
package br.inf.ufsc.onix.esb.mule;

import java.io.InputStream;

import org.apache.log4j.Logger;
import org.mule.api.MuleContext;
import org.mule.api.MuleException;
import org.mule.api.config.ConfigurationBuilder;
import org.mule.config.ConfigResource;
import org.mule.config.spring.SpringXmlConfigurationBuilder;
import org.mule.context.DefaultMuleContextFactory;
```

```
import br.inf.ufsc.onix.api.EndpointBuilder;
import br.inf.ufsc.onix.api.MessageDispatcher;
import br.inf.ufsc.onix.exception.OnixException;
import br.inf.ufsc.onix.impl.AbstractOnixEngine;
import br.inf.ufsc.onix.impl.DefaultServiceRegistry;

/**
 * Esta implementaçao da engine de integracao do framework se baseia em
 * componentes do Mule ESB.
 *
 * @author Gabriel Nunes Menezes<bielmenezes@gmail.com> *
 * @since 16/02/2009
 */
public class MuleOnixEngine extends AbstractOnixEngine {

	protected static Logger logger = Logger.getLogger(MuleOnixEngine.class);

	private MuleContext context;

	private EndpointBuilder endpointBuilder;

	private MessageDispatcher dispatcher;

	private static MuleOnixEngine instance;
	static {
		try {
			instance = new MuleOnixEngine();
			instance.setRegistry(new DefaultServiceRegistry());
		} catch (MuleException e) {
			logger.fatal("Erro ao criar instancia do OnixEngine, "
					+ "o framework nao inicializou");
			logger.fatal(e);

			throw new ExceptionInInitializerError(e);
		}
	}

	public static MuleOnixEngine getInstance() {
		return instance;
	}

	public MuleOnixEngine() throws MuleException {
		logger.info("Criando nova instancia do Onix Engine.");
		// String masterConfigFile = this.getClass().getResource(
		// "mule2-xml-master-config.xml").getPath();
		InputStream masterXmlConfigStream = this.getClass()
				.getResourceAsStream("mule2-xml-master-config.xml");
		ConfigResource configResource = new ConfigResource("xmlConfig",
				masterXmlConfigStream);
		ConfigurationBuilder configBuilder = new SpringXmlConfigurationBuilder(
				new ConfigResource[] { configResource });
```

```
		context = new DefaultMuleContextFactory()
				.createMuleContext(configBuilder);
		endpointBuilder = new MuleEndpointBuilder();
		try {
			dispatcher = new MuleMessageDispatcher(this);
		} catch (OnixException e) {
			// TODO - TRATAR
		}
		logger.info("Nova instancia do Onix Engine criada.");
	}

	@Override
	public void restart() throws OnixException {
		try {
			logger.info("Reiniciando engine do Onix.");
			context.stop();
			context.start();
			logger.info("Engine reiniciada com sucesso.");
		} catch (MuleException e) {
			logger.fatal("Erro ao reinicializar engine.");
			throw new OnixException(e);
		}
	}

	@Override
	public void start() throws OnixException {
		try {
			context.start();
		} catch (MuleException e) {
			logger.fatal("Erro ao inicializar engine.");
			throw new OnixException(e);
		}
	}

	@Override
	public void stop() throws OnixException {
		try {
			context.stop();
		} catch (MuleException e) {
			logger.fatal("Erro ao parar engine.");
			throw new OnixException(e);
		}
	}

	@Override
	public EndpointBuilder getEndpointBuilder() {
		return endpointBuilder;
	}

	@Override
	public MessageDispatcher getMessageDispatcher() {
		return dispatcher;
```

```
	}

}
```

Class MuleOnixInboundUmo

```
package br.inf.ufsc.onix.esb.mule;

import org.apache.log4j.Logger;
import org.mule.api.MuleEventContext;
import org.mule.api.lifecycle.Callable;
import org.mule.api.lifecycle.Initialisable;
import org.mule.api.lifecycle.InitialisationException;

import br.inf.ufsc.onix.api.Message;
import br.inf.ufsc.onix.api.OnixEngine;
import br.inf.ufsc.onix.config.types.CtIntengrationChain;
import br.inf.ufsc.onix.impl.OnixMessage;

/**
 * Componente responsável por receber e tratar as mensagens de integração.
 *
 * @author Gabriel Nunes Menezes<bielmenezes@gmail.com>
 *
 * @since 15/01/2009
 */
public class MuleOnixInboundUmo implements Initialisable, Callable {

	public static final String MULE_ADDRESS = "vm://onix-processor/chain";

	private static final String COMPONENT_NAME = "OnixChainUmo";

	protected Logger logger = Logger.getLogger(this.getClass());

	private OnixEngine engine;

	@Override
	public void initialise() throws InitialisationException {
		logger.info("Inicializando componente " + COMPONENT_NAME);
		engine = MuleOnixEngine.getInstance();
		if (engine == null) {
			// TODO - Alterar para Initialisation Exception
			logger.fatal("Erro ao inicalizar " + COMPONENT_NAME
					+ ", engine não pode ser null.");
			throw new RuntimeException(
					"Erro ao criar componente, engine não pode ser null.");
		}

		logger.info("Componente " + COMPONENT_NAME
				+ " inicializado com sucesso.");
	}
```

```
	@Override
	public Object onCall(MuleEventContext eventContext) throws Exception {
		logger.info("Mensagem recebida no componente " + COMPONENT_NAME
				+ ", iniciando tratamento..");
		Object muleMessage = eventContext.getMessage().getPayload();
		if (!(muleMessage instanceof OnixMessage)) {
			throw new IllegalArgumentException(
					"Mensagem recebida no fluxo não é instancia da Onix
Message.");
		}

		Message onixMessage = (Message) muleMessage;
		// envia a mensagem para o processamento
		Message onixResultMessage = engine.processMessage(onixMessage);

		// retorna o corpo da mensagem para o cliente.
		return onixResultMessage.getBody();
	}

}
```

Class CtBroadcastingRouter

```
package br.inf.ufsc.onix.config.types;

import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlType;


/**
 * <p>Java class for ct_broadcasting_router complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within this
class.
 *
 * <pre>
 * &lt;complexType name="ct_broadcasting_router">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;sequence>
 *         &lt;element name="outgoing-endpoint"
type="{http://inf.ufsc.br/onix}ct_endpoint_decl" maxOccurs="unbounded"/>
 *       &lt;/sequence>
 *       &lt;attribute name="inboundTransformerId"
type="{http://www.w3.org/2001/XMLSchema}string" />
 *     &lt;/restriction>
 *   &lt;/complexContent>
```

```
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "ct_broadcasting_router", propOrder = {
    "outgoingEndpoint"
})
public class CtBroadcastingRouter {

    @XmlElement(name = "outgoing-endpoint", required = true)
    protected List<CtEndpointDecl> outgoingEndpoint;
    @XmlAttribute
    protected String inboundTransformerId;

    /**
     * Gets the value of the outgoingEndpoint property.
     *
     * <p>
     * This accessor method returns a reference to the live list,
     * not a snapshot. Therefore any modification you make to the
     * returned list will be present inside the JAXB object.
     * This is why there is not a <CODE>set</CODE> method for the outgoingEndpoint
property.
     *
     * <p>
     * For example, to add a new item, do as follows:
     * <pre>
     *    getOutgoingEndpoint().add(newItem);
     * </pre>
     *
     *
     * <p>
     * Objects of the following type(s) are allowed in the list
     * {@link CtEndpointDecl }
     *
     *
     */
    public List<CtEndpointDecl> getOutgoingEndpoint() {
        if (outgoingEndpoint == null) {
            outgoingEndpoint = new ArrayList<CtEndpointDecl>();
        }
        return this.outgoingEndpoint;
    }

    /**
     * Gets the value of the inboundTransformerId property.
     *
     * @return
     *     possible object is
     *     {@link String }
```

```
     *
     */
    public String getInboundTransformerId() {
        return inboundTransformerId;
    }

    /**
     * Sets the value of the inboundTransformerId property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setInboundTransformerId(String value) {
        this.inboundTransformerId = value;
    }

}
```

Class CtContentbasedRouter

```
package br.inf.ufsc.onix.config.types;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlType;


/**
 * <p>Java class for ct_contentbased_router complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within this
class.
 *
 * <pre>
 * &lt;complexType name="ct_contentbased_router">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;sequence>
 *         &lt;element name="outgoing-endpoint"
type="{http://inf.ufsc.br/onix}ct_endpoint_decl"/>
 *         &lt;element name="properties" type="{http://inf.ufsc.br/onix}ct_properties"
minOccurs="0"/>
 *       &lt;/sequence>
 *       &lt;attribute name="class" use="required"
type="{http://www.w3.org/2001/XMLSchema}string" />
 *       &lt;attribute name="inboundTransformerId"
type="{http://www.w3.org/2001/XMLSchema}string" />
```

```
 *       &lt;attribute name="outboundTransformerId"
type="{http://www.w3.org/2001/XMLSchema}string" />
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "ct_contentbased_router", propOrder = {
    "outgoingEndpoint",
    "properties"
})
public class CtContentbasedRouter {

    @XmlElement(name = "outgoing-endpoint", required = true)
    protected CtEndpointDecl outgoingEndpoint;
    protected CtProperties properties;
    @XmlAttribute(name = "class", required = true)
    protected String clazz;
    @XmlAttribute
    protected String inboundTransformerId;
    @XmlAttribute
    protected String outboundTransformerId;

    /**
     * Gets the value of the outgoingEndpoint property.
     *
     * @return
     *     possible object is
     *     {@link CtEndpointDecl }
     *
     */
    public CtEndpointDecl getOutgoingEndpoint() {
        return outgoingEndpoint;
    }

    /**
     * Sets the value of the outgoingEndpoint property.
     *
     * @param value
     *     allowed object is
     *     {@link CtEndpointDecl }
     *
     */
    public void setOutgoingEndpoint(CtEndpointDecl value) {
        this.outgoingEndpoint = value;
    }

    /**
     * Gets the value of the properties property.
```

```
     *
     * @return
     *     possible object is
     *     {@link CtProperties }
     *
     */
    public CtProperties getProperties() {
        return properties;
    }

    /**
     * Sets the value of the properties property.
     *
     * @param value
     *     allowed object is
     *     {@link CtProperties }
     *
     */
    public void setProperties(CtProperties value) {
        this.properties = value;
    }

    /**
     * Gets the value of the clazz property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getClazz() {
        return clazz;
    }

    /**
     * Sets the value of the clazz property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setClazz(String value) {
        this.clazz = value;
    }

    /**
     * Gets the value of the inboundTransformerId property.
     *
     * @return
     *     possible object is
     *     {@link String }
```

```
     *
     */
    public String getInboundTransformerId() {
        return inboundTransformerId;
    }

    /**
     * Sets the value of the inboundTransformerId property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setInboundTransformerId(String value) {
        this.inboundTransformerId = value;
    }

    /**
     * Gets the value of the outboundTransformerId property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getOutboundTransformerId() {
        return outboundTransformerId;
    }

    /**
     * Sets the value of the outboundTransformerId property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setOutboundTransformerId(String value) {
        this.outboundTransformerId = value;
    }

}
```

Class CtEndpoint

```
package br.inf.ufsc.onix.config.types;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlElement;
```

```
import javax.xml.bind.annotation.XmlType;


/**
 * <p>Java class for ct_endpoint complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within this
class.
 *
 * <pre>
 * &lt;complexType name="ct_endpoint">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;sequence>
 *         &lt;element name="protocol" type="{http://inf.ufsc.br/onix}st_protocols"/>
 *         &lt;choice>
 *           &lt;element name="host" type="{http://inf.ufsc.br/onix}ct_host"/>
 *           &lt;element name="address"
type="{http://www.w3.org/2001/XMLSchema}string"/>
 *         &lt;/choice>
 *         &lt;element name="properties" type="{http://inf.ufsc.br/onix}ct_properties"
minOccurs="0"/>
 *       &lt;/sequence>
 *       &lt;attribute name="id" use="required"
type="{http://www.w3.org/2001/XMLSchema}string" />
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "ct_endpoint", propOrder = {
    "protocol",
    "host",
    "address",
    "properties"
})
public class CtEndpoint {

    @XmlElement(required = true)
    protected StProtocols protocol;
    protected CtHost host;
    protected String address;
    protected CtProperties properties;
    @XmlAttribute(required = true)
    protected String id;

    /**
     * Gets the value of the protocol property.
     *
```

```
     * @return
     *     possible object is
     *     {@link StProtocols }
     *
     */
    public StProtocols getProtocol() {
        return protocol;
    }

    /**
     * Sets the value of the protocol property.
     *
     * @param value
     *     allowed object is
     *     {@link StProtocols }
     *
     */
    public void setProtocol(StProtocols value) {
        this.protocol = value;
    }

    /**
     * Gets the value of the host property.
     *
     * @return
     *     possible object is
     *     {@link CtHost }
     *
     */
    public CtHost getHost() {
        return host;
    }

    /**
     * Sets the value of the host property.
     *
     * @param value
     *     allowed object is
     *     {@link CtHost }
     *
     */
    public void setHost(CtHost value) {
        this.host = value;
    }

    /**
     * Gets the value of the address property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
```

```
     */
    public String getAddress() {
        return address;
    }

    /**
     * Sets the value of the address property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setAddress(String value) {
        this.address = value;
    }

    /**
     * Gets the value of the properties property.
     *
     * @return
     *     possible object is
     *     {@link CtProperties }
     *
     */
    public CtProperties getProperties() {
        return properties;
    }

    /**
     * Sets the value of the properties property.
     *
     * @param value
     *     allowed object is
     *     {@link CtProperties }
     *
     */
    public void setProperties(CtProperties value) {
        this.properties = value;
    }

    /**
     * Gets the value of the id property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getId() {
        return id;
    }
```

```
    /**
     * Sets the value of the id property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setId(String value) {
        this.id = value;
    }

}
```

Class CtEndpointDecl

```
package br.inf.ufsc.onix.config.types;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlType;


/**
 * <p>Java class for ct_endpoint_decl complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within this
class.
 *
 * <pre>
 * &lt;complexType name="ct_endpoint_decl">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;attribute name="id" use="required"
type="{http://www.w3.org/2001/XMLSchema}string" />
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "ct_endpoint_decl")
public class CtEndpointDecl {

    @XmlAttribute(required = true)
    protected String id;

    /**
```

```
     * Gets the value of the id property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getId() {
        return id;
    }

    /**
     * Sets the value of the id property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setId(String value) {
        this.id = value;
    }

}
```

Class CtEndpoints

```
package br.inf.ufsc.onix.config.types;

import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlType;


/**
 * <p>Java class for ct_endpoints complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within this class.
 *
 * <pre>
 * &lt;complexType name="ct_endpoints">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;sequence>
 *         &lt;element name="endpoint" type="{http://inf.ufsc.br/onix}ct_endpoint_decl" maxOccurs="unbounded"/>
```

```
 *       &lt;/sequence>
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "ct_endpoints", propOrder = {
    "endpoint"
})
public class CtEndpoints {

    @XmlElement(required = true)
    protected List<CtEndpointDecl> endpoint;

    /**
     * Gets the value of the endpoint property.
     *
     * <p>
     * This accessor method returns a reference to the live list,
     * not a snapshot. Therefore any modification you make to the
     * returned list will be present inside the JAXB object.
     * This is why there is not a <CODE>set</CODE> method for the endpoint property.
     *
     * <p>
     * For example, to add a new item, do as follows:
     * <pre>
     *    getEndpoint().add(newItem);
     * </pre>
     *
     *
     * <p>
     * Objects of the following type(s) are allowed in the list
     * {@link CtEndpointDecl }
     *
     *
     */
    public List<CtEndpointDecl> getEndpoint() {
        if (endpoint == null) {
            endpoint = new ArrayList<CtEndpointDecl>();
        }
        return this.endpoint;
    }

}
```

Class CtEnricherDecl

```
package br.inf.ufsc.onix.config.types;
```

```
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlType;


/**
 * <p>Java class for ct_enricher_decl complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within this
class.
 *
 * <pre>
 * &lt;complexType name="ct_enricher_decl">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;attribute name="id" use="required"
type="{http://www.w3.org/2001/XMLSchema}string" />
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "ct_enricher_decl")
public class CtEnricherDecl {

    @XmlAttribute(required = true)
    protected String id;

    /**
     * Gets the value of the id property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getId() {
        return id;
    }

    /**
     * Sets the value of the id property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
```

```
    public void setId(String value) {
        this.id = value;
    }

}
```

Class CtEnricherImplementation

```
package br.inf.ufsc.onix.config.types;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlType;


/**
 * <p>Java class for ct_enricher_implementation complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within this
class.
 *
 * <pre>
 * &lt;complexType name="ct_enricher_implementation">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;attribute name="id" use="required"
type="{http://www.w3.org/2001/XMLSchema}string" />
 *       &lt;attribute name="class" use="required"
type="{http://www.w3.org/2001/XMLSchema}string" />
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "ct_enricher_implementation")
public class CtEnricherImplementation {

    @XmlAttribute(required = true)
    protected String id;
    @XmlAttribute(name = "class", required = true)
    protected String clazz;

    /**
     * Gets the value of the id property.
     *
     * @return
     *     possible object is
     *     {@link String }
```

```
     *
     */
    public String getId() {
        return id;
    }

    /**
     * Sets the value of the id property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setId(String value) {
        this.id = value;
    }

    /**
     * Gets the value of the clazz property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getClazz() {
        return clazz;
    }

    /**
     * Sets the value of the clazz property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setClazz(String value) {
        this.clazz = value;
    }

}
```

Class CtEnrichers

```
package br.inf.ufsc.onix.config.types;

import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
```

```
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlType;


/**
 * <p>Java class for ct_enrichers complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within this
class.
 *
 * <pre>
 * &lt;complexType name="ct_enrichers">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;sequence>
 *         &lt;element name="enricher" type="{http://inf.ufsc.br/onix}ct_enricher_decl"
maxOccurs="unbounded"/>
 *       &lt;/sequence>
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "ct_enrichers", propOrder = {
    "enricher"
})
public class CtEnrichers {

    @XmlElement(required = true)
    protected List<CtEnricherDecl> enricher;

    /**
     * Gets the value of the enricher property.
     *
     * <p>
     * This accessor method returns a reference to the live list,
     * not a snapshot. Therefore any modification you make to the
     * returned list will be present inside the JAXB object.
     * This is why there is not a <CODE>set</CODE> method for the enricher property.
     *
     * <p>
     * For example, to add a new item, do as follows:
     * <pre>
     *    getEnricher().add(newItem);
     * </pre>
     *
     *
     * <p>
     * Objects of the following type(s) are allowed in the list
```

```
     * {@link CtEnricherDecl }
     *
     *
     */
    public List<CtEnricherDecl> getEnricher() {
        if (enricher == null) {
            enricher = new ArrayList<CtEnricherDecl>();
        }
        return this.enricher;
    }

}
```

Class CtExceptionStrategyDecl

```
package br.inf.ufsc.onix.config.types;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlType;


/**
 * <p>Java class for ct_exception_strategy_decl complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within this
class.
 *
 * <pre>
 * &lt;complexType name="ct_exception_strategy_decl">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;attribute name="id" use="required"
type="{http://www.w3.org/2001/XMLSchema}string" />
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "ct_exception_strategy_decl")
public class CtExceptionStrategyDecl {

    @XmlAttribute(required = true)
    protected String id;

    /**
     * Gets the value of the id property.
     *
```

```
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getId() {
        return id;
    }

    /**
     * Sets the value of the id property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setId(String value) {
        this.id = value;
    }

}
```

Class CtExceptionStrategyImpl

```
package br.inf.ufsc.onix.config.types;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlType;


/**
 * <p>Java class for ct_exception_strategy_impl complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within this
class.
 *
 * <pre>
 * &lt;complexType name="ct_exception_strategy_impl">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;attribute name="id" use="required"
type="{http://www.w3.org/2001/XMLSchema}string" />
 *       &lt;attribute name="class" use="required"
type="{http://www.w3.org/2001/XMLSchema}string" />
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
```

```
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "ct_exception_strategy_impl")
public class CtExceptionStrategyImpl {

    @XmlAttribute(required = true)
    protected String id;
    @XmlAttribute(name = "class", required = true)
    protected String clazz;

    /**
     * Gets the value of the id property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getId() {
        return id;
    }

    /**
     * Sets the value of the id property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setId(String value) {
        this.id = value;
    }

    /**
     * Gets the value of the clazz property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getClazz() {
        return clazz;
    }

    /**
     * Sets the value of the clazz property.
     *
     * @param value
     *     allowed object is
```

```
     *     {@link String }
     *
     */
    public void setClazz(String value) {
        this.clazz = value;
    }

}
```

Class CtExceptionStrategyImpl

```
package br.inf.ufsc.onix.config.types;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlType;


/**
 * <p>Java class for ct_exception_strategy_impl complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within this
class.
 *
 * <pre>
 * &lt;complexType name="ct_exception_strategy_impl">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;attribute name="id" use="required"
type="{http://www.w3.org/2001/XMLSchema}string" />
 *       &lt;attribute name="class" use="required"
type="{http://www.w3.org/2001/XMLSchema}string" />
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "ct_exception_strategy_impl")
public class CtExceptionStrategyImpl {

    @XmlAttribute(required = true)
    protected String id;
    @XmlAttribute(name = "class", required = true)
    protected String clazz;

    /**
     * Gets the value of the id property.
     *
```

```
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getId() {
        return id;
    }

    /**
     * Sets the value of the id property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setId(String value) {
        this.id = value;
    }

    /**
     * Gets the value of the clazz property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getClazz() {
        return clazz;
    }

    /**
     * Sets the value of the clazz property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setClazz(String value) {
        this.clazz = value;
    }

}
```

Class CtIntengrationChain

```
package br.inf.ufsc.onix.config.types;

import javax.xml.bind.annotation.XmlAccessType;
```

```
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlType;


/**
 * <p>Java class for ct_intengration_chain complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within this
class.
 *
 * <pre>
 * &lt;complexType name="ct_intengration_chain">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;sequence>
 *         &lt;element name="send" type="{http://inf.ufsc.br/onix}ct_send"/>
 *         &lt;element name="receive" type="{http://inf.ufsc.br/onix}ct_receive"
minOccurs="0"/>
 *         &lt;element name="exception-strategy"
type="{http://inf.ufsc.br/onix}ct_exception_strategy_decl"/>
 *       &lt;/sequence>
 *       &lt;attribute name="name" use="required"
type="{http://www.w3.org/2001/XMLSchema}string" />
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "ct_intengration_chain", propOrder = {
    "send",
    "receive",
    "exceptionStrategy"
})
public class CtIntengrationChain {

    @XmlElement(required = true)
    protected CtSend send;
    protected CtReceive receive;
    @XmlElement(name = "exception-strategy", required = true)
    protected CtExceptionStrategyDecl exceptionStrategy;
    @XmlAttribute(required = true)
    protected String name;

    /**
     * Gets the value of the send property.
     *
     * @return
```

```
     *     possible object is
     *     {@link CtSend }
     *
     */
    public CtSend getSend() {
        return send;
    }

    /**
     * Sets the value of the send property.
     *
     * @param value
     *     allowed object is
     *     {@link CtSend }
     *
     */
    public void setSend(CtSend value) {
        this.send = value;
    }

    /**
     * Gets the value of the receive property.
     *
     * @return
     *     possible object is
     *     {@link CtReceive }
     *
     */
    public CtReceive getReceive() {
        return receive;
    }

    /**
     * Sets the value of the receive property.
     *
     * @param value
     *     allowed object is
     *     {@link CtReceive }
     *
     */
    public void setReceive(CtReceive value) {
        this.receive = value;
    }

    /**
     * Gets the value of the exceptionStrategy property.
     *
     * @return
     *     possible object is
     *     {@link CtExceptionStrategyDecl }
     *
     */
```

```
    public CtExceptionStrategyDecl getExceptionStrategy() {
        return exceptionStrategy;
    }

    /**
     * Sets the value of the exceptionStrategy property.
     *
     * @param value
     *     allowed object is
     *     {@link CtExceptionStrategyDecl }
     *
     */
    public void setExceptionStrategy(CtExceptionStrategyDecl value) {
        this.exceptionStrategy = value;
    }

    /**
     * Gets the value of the name property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getName() {
        return name;
    }

    /**
     * Sets the value of the name property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setName(String value) {
        this.name = value;
    }

}
```

Class CtProperties

```
package br.inf.ufsc.onix.config.types;

import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlType;
```

```
/**
 * <p>Java class for ct_properties complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within this
class.
 *
 * <pre>
 * &lt;complexType name="ct_properties">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;sequence>
 *         &lt;element name="property" type="{http://inf.ufsc.br/onix}ct_property"
maxOccurs="unbounded" minOccurs="0"/>
 *       &lt;/sequence>
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "ct_properties", propOrder = {
    "property"
})
public class CtProperties {

    protected List<CtProperty> property;

    /**
     * Gets the value of the property property.
     *
     * <p>
     * This accessor method returns a reference to the live list,
     * not a snapshot. Therefore any modification you make to the
     * returned list will be present inside the JAXB object.
     * This is why there is not a <CODE>set</CODE> method for the property property.
     *
     * <p>
     * For example, to add a new item, do as follows:
     * <pre>
     *    getProperty().add(newItem);
     * </pre>
     *
     *
     * <p>
     * Objects of the following type(s) are allowed in the list
     * {@link CtProperty }
     *
     *
     */
```

```
    public List<CtProperty> getProperty() {
        if (property == null) {
            property = new ArrayList<CtProperty>();
        }
        return this.property;
    }

}
```

Class CtProperty

```
package br.inf.ufsc.onix.config.types;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlType;
import javax.xml.bind.annotation.XmlValue;


/**
 * <p>Java class for ct_property complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within this
class.
 *
 * <pre>
 * &lt;complexType name="ct_property">
 *   &lt;simpleContent>
 *     &lt;extension base="&lt;http://www.w3.org/2001/XMLSchema>string">
 *       &lt;attribute name="name" use="required"
type="{http://www.w3.org/2001/XMLSchema}string" />
 *     &lt;/extension>
 *   &lt;/simpleContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "ct_property", propOrder = {
    "value"
})
public class CtProperty {

    @XmlValue
    protected String value;
    @XmlAttribute(required = true)
    protected String name;

    /**
     * Gets the value of the value property.
     *
```

```
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getValue() {
        return value;
    }

    /**
     * Sets the value of the value property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setValue(String value) {
        this.value = value;
    }

    /**
     * Gets the value of the name property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getName() {
        return name;
    }

    /**
     * Sets the value of the name property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setName(String value) {
        this.name = value;
    }

}
```

Class CtReceive

```
package br.inf.ufsc.onix.config.types;

import javax.xml.bind.annotation.XmlAccessType;
```

```
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlType;


/**
 * <p>Java class for ct_receive complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within this
class.
 *
 * <pre>
 * &lt;complexType name="ct_receive">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;sequence>
 *         &lt;element name="enrichers" type="{http://inf.ufsc.br/onix}ct_enrichers"
minOccurs="0"/>
 *         &lt;element name="validators" type="{http://inf.ufsc.br/onix}ct_validators"
minOccurs="0"/>
 *       &lt;/sequence>
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "ct_receive", propOrder = {
    "enrichers",
    "validators"
})
public class CtReceive {

    protected CtEnrichers enrichers;
    protected CtValidators validators;

    /**
     * Gets the value of the enrichers property.
     *
     * @return
     *     possible object is
     *     {@link CtEnrichers }
     *
     */
    public CtEnrichers getEnrichers() {
        return enrichers;
    }

    /**
     * Sets the value of the enrichers property.
     *
```

```
     * @param value
     *     allowed object is
     *     {@link CtEnrichers }
     *
     */
    public void setEnrichers(CtEnrichers value) {
        this.enrichers = value;
    }

    /**
     * Gets the value of the validators property.
     *
     * @return
     *     possible object is
     *     {@link CtValidators }
     *
     */
    public CtValidators getValidators() {
        return validators;
    }

    /**
     * Sets the value of the validators property.
     *
     * @param value
     *     allowed object is
     *     {@link CtValidators }
     *
     */
    public void setValidators(CtValidators value) {
        this.validators = value;
    }

}
```

Class CtRouters

```
package br.inf.ufsc.onix.config.types;

import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlType;


/**
 * <p>Java class for ct_routers complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within this
class.
```

```
 *
 * <pre>
 * &lt;complexType name="ct_routers">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;sequence>
 *         &lt;element name="content-based-router"
type="{http://inf.ufsc.br/onix}ct_contentbased_router" maxOccurs="unbounded"
minOccurs="0"/>
 *         &lt;element name="broadcasting-router"
type="{http://inf.ufsc.br/onix}ct_broadcasting_router" maxOccurs="unbounded"
minOccurs="0"/>
 *       &lt;/sequence>
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "ct_routers", propOrder = {
    "contentBasedRouter",
    "broadcastingRouter"
})
public class CtRouters {

    @XmlElement(name = "content-based-router")
    protected List<CtContentbasedRouter> contentBasedRouter;
    @XmlElement(name = "broadcasting-router")
    protected List<CtBroadcastingRouter> broadcastingRouter;

    /**
     * Gets the value of the contentBasedRouter property.
     *
     * <p>
     * This accessor method returns a reference to the live list,
     * not a snapshot. Therefore any modification you make to the
     * returned list will be present inside the JAXB object.
     * This is why there is not a <CODE>set</CODE> method for the contentBasedRouter
property.
     *
     * <p>
     * For example, to add a new item, do as follows:
     * <pre>
     *    getContentBasedRouter().add(newItem);
     * </pre>
     *
     *
     * <p>
     * Objects of the following type(s) are allowed in the list
     * {@link CtContentbasedRouter }
```

```
     *
     *
     */
    public List<CtContentbasedRouter> getContentBasedRouter() {
        if (contentBasedRouter == null) {
            contentBasedRouter = new ArrayList<CtContentbasedRouter>();
        }
        return this.contentBasedRouter;
    }

    /**
     * Gets the value of the broadcastingRouter property.
     *
     * <p>
     * This accessor method returns a reference to the live list,
     * not a snapshot. Therefore any modification you make to the
     * returned list will be present inside the JAXB object.
     * This is why there is not a <CODE>set</CODE> method for the broadcastingRouter
property.
     *
     * <p>
     * For example, to add a new item, do as follows:
     * <pre>
     *    getBroadcastingRouter().add(newItem);
     * </pre>
     *
     *
     * <p>
     * Objects of the following type(s) are allowed in the list
     * {@link CtBroadcastingRouter }
     *
     *
     */
    public List<CtBroadcastingRouter> getBroadcastingRouter() {
        if (broadcastingRouter == null) {
            broadcastingRouter = new ArrayList<CtBroadcastingRouter>();
        }
        return this.broadcastingRouter;
    }

}
```

Class CtSend

```
package br.inf.ufsc.onix.config.types;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlType;
```

```
/**
 * <p>Java class for ct_send complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within this
class.
 *
 * <pre>
 * &lt;complexType name="ct_send">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;sequence>
 *         &lt;element name="incoming" type="{http://inf.ufsc.br/onix}ct_endpoints"/>
 *         &lt;element name="validators" type="{http://inf.ufsc.br/onix}ct_validators"
minOccurs="0"/>
 *         &lt;element name="enrichers" type="{http://inf.ufsc.br/onix}ct_enrichers"
minOccurs="0"/>
 *         &lt;element name="routers" type="{http://inf.ufsc.br/onix}ct_routers"/>
 *       &lt;/sequence>
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "ct_send", propOrder = {
    "incoming",
    "validators",
    "enrichers",
    "routers"
})
public class CtSend {

    @XmlElement(required = true)
    protected CtEndpoints incoming;
    protected CtValidators validators;
    protected CtEnrichers enrichers;
    @XmlElement(required = true)
    protected CtRouters routers;

    /**
     * Gets the value of the incoming property.
     *
     * @return
     *     possible object is
     *     {@link CtEndpoints }
     *
     */
    public CtEndpoints getIncoming() {
        return incoming;
    }
```

```
/**
 * Sets the value of the incoming property.
 *
 * @param value
 *     allowed object is
 *     {@link CtEndpoints }
 *
 */
public void setIncoming(CtEndpoints value) {
    this.incoming = value;
}

/**
 * Gets the value of the validators property.
 *
 * @return
 *     possible object is
 *     {@link CtValidators }
 *
 */
public CtValidators getValidators() {
    return validators;
}

/**
 * Sets the value of the validators property.
 *
 * @param value
 *     allowed object is
 *     {@link CtValidators }
 *
 */
public void setValidators(CtValidators value) {
    this.validators = value;
}

/**
 * Gets the value of the enrichers property.
 *
 * @return
 *     possible object is
 *     {@link CtEnrichers }
 *
 */
public CtEnrichers getEnrichers() {
    return enrichers;
}

/**
 * Sets the value of the enrichers property.
 *
```

```
     * @param value
     *     allowed object is
     *     {@link CtEnrichers }
     *
     */
    public void setEnrichers(CtEnrichers value) {
        this.enrichers = value;
    }

    /**
     * Gets the value of the routers property.
     *
     * @return
     *     possible object is
     *     {@link CtRouters }
     *
     */
    public CtRouters getRouters() {
        return routers;
    }

    /**
     * Sets the value of the routers property.
     *
     * @param value
     *     allowed object is
     *     {@link CtRouters }
     *
     */
    public void setRouters(CtRouters value) {
        this.routers = value;
    }

}
```

Class CtTransformerImplementation

```
package br.inf.ufsc.onix.config.types;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlType;


/**
 * <p>Java class for ct_transformer_implementation complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within this
class.
 *
 * <pre>
```

```
 * &lt;complexType name="ct_transformer_implementation">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;attribute name="id" use="required"
type="{http://www.w3.org/2001/XMLSchema}string" />
 *       &lt;attribute name="class" use="required"
type="{http://www.w3.org/2001/XMLSchema}string" />
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "ct_transformer_implementation")
public class CtTransformerImplementation {

    @XmlAttribute(required = true)
    protected String id;
    @XmlAttribute(name = "class", required = true)
    protected String clazz;

    /**
     * Gets the value of the id property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getId() {
        return id;
    }

    /**
     * Sets the value of the id property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setId(String value) {
        this.id = value;
    }

    /**
     * Gets the value of the clazz property.
     *
     * @return
     *     possible object is
```

```
     *     {@link String }
     *
     */
    public String getClazz() {
        return clazz;
    }

    /**
     * Sets the value of the clazz property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setClazz(String value) {
        this.clazz = value;
    }

}
```

Class CtValidatorDecl

```
package br.inf.ufsc.onix.config.types;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlType;


/**
 * <p>Java class for ct_validator_decl complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within this
class.
 *
 * <pre>
 * &lt;complexType name="ct_validator_decl">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;attribute name="id" use="required"
type="{http://www.w3.org/2001/XMLSchema}string" />
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "ct_validator_decl")
```

```
public class CtValidatorDecl {

    @XmlAttribute(required = true)
    protected String id;

    /**
     * Gets the value of the id property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getId() {
        return id;
    }

    /**
     * Sets the value of the id property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setId(String value) {
        this.id = value;
    }

}
```

Class CtValidatorImplementation

```
package br.inf.ufsc.onix.config.types;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlType;


/**
 * <p>Java class for ct_validator_implementation complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within this
class.
 *
 * <pre>
 * &lt;complexType name="ct_validator_implementation">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;attribute name="id" use="required"
```

```
type="{http://www.w3.org/2001/XMLSchema}string" />
 *       &lt;attribute name="class" use="required"
type="{http://www.w3.org/2001/XMLSchema}string" />
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "ct_validator_implementation")
public class CtValidatorImplementation {

    @XmlAttribute(required = true)
    protected String id;
    @XmlAttribute(name = "class", required = true)
    protected String clazz;

    /**
     * Gets the value of the id property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getId() {
        return id;
    }

    /**
     * Sets the value of the id property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setId(String value) {
        this.id = value;
    }

    /**
     * Gets the value of the clazz property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getClazz() {
```

```
        return clazz;
    }

    /**
     * Sets the value of the clazz property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setClazz(String value) {
        this.clazz = value;
    }

}
```

Class CtValidators

```
package br.inf.ufsc.onix.config.types;

import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlType;


/**
 * <p>Java class for ct_validators complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within this
class.
 *
 * <pre>
 * &lt;complexType name="ct_validators">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;sequence>
 *         &lt;element name="validator" type="{http://inf.ufsc.br/onix}ct_validator_decl"
maxOccurs="unbounded"/>
 *       &lt;/sequence>
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "ct_validators", propOrder = {
```

```
    "validator"
})
public class CtValidators {

    @XmlElement(required = true)
    protected List<CtValidatorDecl> validator;

    /**
     * Gets the value of the validator property.
     *
     * <p>
     * This accessor method returns a reference to the live list,
     * not a snapshot. Therefore any modification you make to the
     * returned list will be present inside the JAXB object.
     * This is why there is not a <CODE>set</CODE> method for the validator property.
     *
     * <p>
     * For example, to add a new item, do as follows:
     * <pre>
     *    getValidator().add(newItem);
     * </pre>
     *
     *
     * <p>
     * Objects of the following type(s) are allowed in the list
     * {@link CtValidatorDecl }
     *
     *
     */
    public List<CtValidatorDecl> getValidator() {
        if (validator == null) {
            validator = new ArrayList<CtValidatorDecl>();
        }
        return this.validator;
    }

}
```

Class ObjectFactory

```
package br.inf.ufsc.onix.config.types;

import javax.xml.bind.JAXBElement;
import javax.xml.bind.annotation.XmlElementDecl;
import javax.xml.bind.annotation.XmlRegistry;
import javax.xml.namespace.QName;


/**
 * This object contains factory methods for each
 * Java content interface and Java element interface
 * generated in the br.inf.ufsc.onix.config.types package.
```

```
 * <p>An ObjectFactory allows you to programatically
 * construct new instances of the Java representation
 * for XML content. The Java representation of XML
 * content can consist of schema derived interfaces
 * and classes representing the binding of schema
 * type definitions, element declarations and model
 * groups.  Factory methods for each of these are
 * provided in this class.
 *
 */
@XmlRegistry
public class ObjectFactory {

    private final static QName _OnixConfig_QNAME = new QName("http://inf.ufsc.br/onix",
"onix-config");

    /**
     * Create a new ObjectFactory that can be used to create new instances of schema
derived classes for package: br.inf.ufsc.onix.config.types
     *
     */
    public ObjectFactory() {
    }

    /**
     * Create an instance of {@link CtBroadcastingRouter }
     *
     */
    public CtBroadcastingRouter createCtBroadcastingRouter() {
        return new CtBroadcastingRouter();
    }

    /**
     * Create an instance of {@link CtExceptionStrategyDecl }
     *
     */
    public CtExceptionStrategyDecl createCtExceptionStrategyDecl() {
        return new CtExceptionStrategyDecl();
    }

    /**
     * Create an instance of {@link CtEnricherDecl }
     *
     */
    public CtEnricherDecl createCtEnricherDecl() {
        return new CtEnricherDecl();
    }

    /**
     * Create an instance of {@link CtValidatorImplementation }
     *
     */
```

```
public CtValidatorImplementation createCtValidatorImplementation() {
    return new CtValidatorImplementation();
}

/**
 * Create an instance of {@link CtReceive }
 *
 */
public CtReceive createCtReceive() {
    return new CtReceive();
}

/**
 * Create an instance of {@link CtEndpoint }
 *
 */
public CtEndpoint createCtEndpoint() {
    return new CtEndpoint();
}

/**
 * Create an instance of {@link CtValidatorDecl }
 *
 */
public CtValidatorDecl createCtValidatorDecl() {
    return new CtValidatorDecl();
}

/**
 * Create an instance of {@link CtProperties }
 *
 */
public CtProperties createCtProperties() {
    return new CtProperties();
}

/**
 * Create an instance of {@link CtEndpointDecl }
 *
 */
public CtEndpointDecl createCtEndpointDecl() {
    return new CtEndpointDecl();
}

/**
 * Create an instance of {@link CtIntengrationChain }
 *
 */
public CtIntengrationChain createCtIntengrationChain() {
    return new CtIntengrationChain();
}
```

```
/**
 * Create an instance of {@link CtEndpoints }
 *
 */
public CtEndpoints createCtEndpoints() {
    return new CtEndpoints();
}

/**
 * Create an instance of {@link CtSend }
 *
 */
public CtSend createCtSend() {
    return new CtSend();
}

/**
 * Create an instance of {@link OnixConfig }
 *
 */
public OnixConfig createOnixConfig() {
    return new OnixConfig();
}

/**
 * Create an instance of {@link CtExceptionStrategyImpl }
 *
 */
public CtExceptionStrategyImpl createCtExceptionStrategyImpl() {
    return new CtExceptionStrategyImpl();
}

/**
 * Create an instance of {@link CtTransformerImplementation }
 *
 */
public CtTransformerImplementation createCtTransformerImplementation() {
    return new CtTransformerImplementation();
}

/**
 * Create an instance of {@link CtEnrichers }
 *
 */
public CtEnrichers createCtEnrichers() {
    return new CtEnrichers();
}

/**
 * Create an instance of {@link CtRouters }
 *
 */
```

```
    public CtRouters createCtRouters() {
        return new CtRouters();
    }

    /**
     * Create an instance of {@link CtHost }
     *
     */
    public CtHost createCtHost() {
        return new CtHost();
    }

    /**
     * Create an instance of {@link CtEnricherImplementation }
     *
     */
    public CtEnricherImplementation createCtEnricherImplementation() {
        return new CtEnricherImplementation();
    }

    /**
     * Create an instance of {@link CtContentbasedRouter }
     *
     */
    public CtContentbasedRouter createCtContentbasedRouter() {
        return new CtContentbasedRouter();
    }

    /**
     * Create an instance of {@link CtValidators }
     *
     */
    public CtValidators createCtValidators() {
        return new CtValidators();
    }

    /**
     * Create an instance of {@link CtProperty }
     *
     */
    public CtProperty createCtProperty() {
        return new CtProperty();
    }

    /**
     * Create an instance of {@link JAXBElement }{@code <}{@link OnixConfig }{@code
>}}
     *
     */
    @XmlElementDecl(namespace = "http://inf.ufsc.br/onix", name = "onix-config")
    public JAXBElement<OnixConfig> createOnixConfig(OnixConfig value) {
        return new JAXBElement<OnixConfig>(_OnixConfig_QNAME, OnixConfig.class, null,
```

```
value);
    }

}
```

Class OnixConfig

```
package br.inf.ufsc.onix.config.types;

import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlType;


/**
 * <p>Java class for onix-config complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within this
class.
 *
 * <pre>
 * &lt;complexType name="onix-config">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;sequence>
 *         &lt;element name="integration-chain"
type="{http://inf.ufsc.br/onix}ct_intengration_chain" maxOccurs="unbounded"
minOccurs="0"/>
 *         &lt;element name="endpoint" type="{http://inf.ufsc.br/onix}ct_endpoint"
maxOccurs="unbounded" minOccurs="0"/>
 *         &lt;element name="validator"
type="{http://inf.ufsc.br/onix}ct_validator_implementation" maxOccurs="unbounded"
minOccurs="0"/>
 *         &lt;element name="enricher"
type="{http://inf.ufsc.br/onix}ct_enricher_implementation" maxOccurs="unbounded"
minOccurs="0"/>
 *         &lt;element name="tranformer"
type="{http://inf.ufsc.br/onix}ct_transformer_implementation" maxOccurs="unbounded"
minOccurs="0"/>
 *         &lt;element name="exception-strategy"
type="{http://inf.ufsc.br/onix}ct_exception_strategy_impl" maxOccurs="unbounded"
minOccurs="0"/>
 *       &lt;/sequence>
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
```

```
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "onix-config", propOrder = {
    "integrationChain",
    "endpoint",
    "validator",
    "enricher",
    "tranformer",
    "exceptionStrategy"
})
public class OnixConfig {

    @XmlElement(name = "integration-chain")
    protected List<CtIntengrationChain> integrationChain;
    protected List<CtEndpoint> endpoint;
    protected List<CtValidatorImplementation> validator;
    protected List<CtEnricherImplementation> enricher;
    protected List<CtTransformerImplementation> tranformer;
    @XmlElement(name = "exception-strategy")
    protected List<CtExceptionStrategyImpl> exceptionStrategy;

    /**
     * Gets the value of the integrationChain property.
     *
     * <p>
     * This accessor method returns a reference to the live list,
     * not a snapshot. Therefore any modification you make to the
     * returned list will be present inside the JAXB object.
     * This is why there is not a <CODE>set</CODE> method for the integrationChain
property.
     *
     * <p>
     * For example, to add a new item, do as follows:
     * <pre>
     *    getIntegrationChain().add(newItem);
     * </pre>
     *
     *
     * <p>
     * Objects of the following type(s) are allowed in the list
     * {@link CtIntengrationChain }
     *
     *
     */
    public List<CtIntengrationChain> getIntegrationChain() {
        if (integrationChain == null) {
            integrationChain = new ArrayList<CtIntengrationChain>();
        }
        return this.integrationChain;
    }

    /**
```

```
     * Gets the value of the endpoint property.
     *
     * <p>
     * This accessor method returns a reference to the live list,
     * not a snapshot. Therefore any modification you make to the
     * returned list will be present inside the JAXB object.
     * This is why there is not a <CODE>set</CODE> method for the endpoint property.
     *
     * <p>
     * For example, to add a new item, do as follows:
     * <pre>
     *    getEndpoint().add(newItem);
     * </pre>
     *
     *
     * <p>
     * Objects of the following type(s) are allowed in the list
     * {@link CtEndpoint }
     *
     *
     */
    public List<CtEndpoint> getEndpoint() {
        if (endpoint == null) {
            endpoint = new ArrayList<CtEndpoint>();
        }
        return this.endpoint;
    }

    /**
     * Gets the value of the validator property.
     *
     * <p>
     * This accessor method returns a reference to the live list,
     * not a snapshot. Therefore any modification you make to the
     * returned list will be present inside the JAXB object.
     * This is why there is not a <CODE>set</CODE> method for the validator property.
     *
     * <p>
     * For example, to add a new item, do as follows:
     * <pre>
     *    getValidator().add(newItem);
     * </pre>
     *
     *
     * <p>
     * Objects of the following type(s) are allowed in the list
     * {@link CtValidatorImplementation }
     *
     *
     */
    public List<CtValidatorImplementation> getValidator() {
        if (validator == null) {
```

```
        validator = new ArrayList<CtValidatorImplementation>();
    }
    return this.validator;
}

/**
 * Gets the value of the enricher property.
 *
 * <p>
 * This accessor method returns a reference to the live list,
 * not a snapshot. Therefore any modification you make to the
 * returned list will be present inside the JAXB object.
 * This is why there is not a <CODE>set</CODE> method for the enricher property.
 *
 * <p>
 * For example, to add a new item, do as follows:
 * <pre>
 *    getEnricher().add(newItem);
 * </pre>
 *
 *
 * <p>
 * Objects of the following type(s) are allowed in the list
 * {@link CtEnricherImplementation }
 *
 *
 */
public List<CtEnricherImplementation> getEnricher() {
    if (enricher == null) {
        enricher = new ArrayList<CtEnricherImplementation>();
    }
    return this.enricher;
}

/**
 * Gets the value of the tranformer property.
 *
 * <p>
 * This accessor method returns a reference to the live list,
 * not a snapshot. Therefore any modification you make to the
 * returned list will be present inside the JAXB object.
 * This is why there is not a <CODE>set</CODE> method for the tranformer property.
 *
 * <p>
 * For example, to add a new item, do as follows:
 * <pre>
 *    getTranformer().add(newItem);
 * </pre>
 *
 *
 * <p>
 * Objects of the following type(s) are allowed in the list
```

```
     * {@link CtTransformerImplementation }
     *
     *
     */
    public List<CtTransformerImplementation> getTranformer() {
        if (tranformer == null) {
            tranformer = new ArrayList<CtTransformerImplementation>();
        }
        return this.tranformer;
    }

    /**
     * Gets the value of the exceptionStrategy property.
     *
     * <p>
     * This accessor method returns a reference to the live list,
     * not a snapshot. Therefore any modification you make to the
     * returned list will be present inside the JAXB object.
     * This is why there is not a <CODE>set</CODE> method for the exceptionStrategy
property.
     *
     * <p>
     * For example, to add a new item, do as follows:
     * <pre>
     *    getExceptionStrategy().add(newItem);
     * </pre>
     *
     *
     * <p>
     * Objects of the following type(s) are allowed in the list
     * {@link CtExceptionStrategyImpl }
     *
     *
     */
    public List<CtExceptionStrategyImpl> getExceptionStrategy() {
        if (exceptionStrategy == null) {
            exceptionStrategy = new ArrayList<CtExceptionStrategyImpl>();
        }
        return this.exceptionStrategy;
    }

}
```

Class StProtocols

```
package br.inf.ufsc.onix.config.types;

import javax.xml.bind.annotation.XmlEnum;
import javax.xml.bind.annotation.XmlType;


/**
```

```
 * <p>Java class for st_protocols.
 *
 * <p>The following schema fragment specifies the expected content contained within this
class.
 * <p>
 * <pre>
 * &lt;simpleType name="st_protocols">
 *   &lt;restriction base="{http://www.w3.org/2001/XMLSchema}string">
 *     &lt;enumeration value="TCP"/>
 *     &lt;enumeration value="HTTP"/>
 *     &lt;enumeration value="FILE"/>
 *     &lt;enumeration value="EMAIL"/>
 *     &lt;enumeration value="WS_AXIS"/>
 *     &lt;enumeration value="WS_CXF"/>
 *   &lt;/restriction>
 * &lt;/simpleType>
 * </pre>
 *
 */
@XmlType(name = "st_protocols")
@XmlEnum
public enum StProtocols {

    TCP,
    HTTP,
    FILE,
    EMAIL,
    WS_AXIS,
    WS_CXF;

    public String value() {
        return name();
    }

    public static StProtocols fromValue(String v) {
        return valueOf(v);
    }

}
```

XML Schema OnixConfig

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://inf.ufsc.br/onix"
        elementFormDefault="qualified" xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:onx="http://inf.ufsc.br/onix">
        <annotation>
                <documentation>
                        Schema diz como devem ser feitas as especificações de
                        configuração para utlização do framework Onix Desenvolvido
                        por Gabriel Nunes Menezes e Pedro Castello Branco Galoppini
                </documentation>
```

```
        </annotation>

        <complexType name="ct_endpoint">
                <sequence>
                        <element name="protocol" type="onx:st_protocols" maxOccurs="1"
                                minOccurs="1">
                        </element>



         <choice>
                                <element name="host" type="onx:ct_host" maxOccurs="1"
                                        minOccurs="1">
                                </element>
            <element name="address" type="string" maxOccurs="1"
minOccurs="1"></element>
                        </choice>

          <element name="properties" type="onx:ct_properties" maxOccurs="1"
minOccurs="0">
                        </element>
       </sequence>
                <attribute name="id" type="string" use="required"></attribute>
        </complexType>


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

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

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

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

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

        <complexType name="ct_property">

                <simpleContent>
```

```
                    <extension base="string">
                        <attribute name="name" type="string"
use="required"></attribute>

                    </extension>
                </simpleContent></complexType>


        <complexType name="ct_contentbased_router">
                <sequence>

                    <element name="outgoing-endpoint"
                        type="onx:ct_endpoint_decl" maxOccurs="1" minOccurs="1">
                    </element>
                    <element name="properties" type="onx:ct_properties"
                        maxOccurs="1" minOccurs="0">
                    </element>
                </sequence>
                <attribute name="class" type="string" use="required"></attribute>
                <attribute name="inboundTransformerId" type="string"></attribute>
                <attribute name="outboundTransformerId" type="string"></attribute>
        </complexType>

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

        <complexType name="ct_intengration_chain">
                <sequence>
                    <element name="send" type="onx:ct_send" maxOccurs="1"
                        minOccurs="1">
                    </element>
                    <element name="receive" type="onx:ct_receive" maxOccurs="1"
                        minOccurs="0">
                    </element>
                    <element name="exception-strategy"
                        type="onx:ct_exception_strategy_decl" maxOccurs="1"
minOccurs="1">
                    </element>
                </sequence>
                <attribute name="name" type="string" use="required"></attribute>
        </complexType>

        <complexType name="ct_endpoints">
                <sequence>
                    <element name="endpoint" type="onx:ct_endpoint_decl"
                        maxOccurs="unbounded" minOccurs="1">
                    </element>
                </sequence>
        </complexType>

        <complexType name="ct_validators">
```

```
            <sequence>
                <element name="validator" type="onx:ct_validator_decl"
                    maxOccurs="unbounded" minOccurs="1">
                </element>
            </sequence>
        </complexType>

        <complexType name="ct_enrichers">
            <sequence>
                <element name="enricher" type="onx:ct_enricher_decl"
                    maxOccurs="unbounded" minOccurs="1">
                </element>
            </sequence>
        </complexType>

        <complexType name="ct_routers">
            <sequence>
                <element name="content-based-router"
                    type="onx:ct_contentbased_router" maxOccurs="unbounded"
                    minOccurs="0">
                </element>
                <element name="broadcasting-router"
type="onx:ct_broadcasting_router" maxOccurs="unbounded" minOccurs="0"></element>
            </sequence>


        </complexType>


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

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




        <complexType name="onix-config">
            <sequence>
                <element name="integration-chain"
                    type="onx:ct_intengration_chain" maxOccurs="unbounded"
                    minOccurs="0">
                </element>
                <element name="endpoint" type="onx:ct_endpoint"
                    maxOccurs="unbounded" minOccurs="0">
                </element>
                <element name="validator"
                    type="onx:ct_validator_implementation"
```

```
maxOccurs="unbounded"
                        minOccurs="0">
                    </element>
                    <element name="enricher"
                        type="onx:ct_enricher_implementation"
maxOccurs="unbounded"
                        minOccurs="0">
                    </element>
                    <element name="tranformer"
                        type="onx:ct_transformer_implementation"
maxOccurs="unbounded"
                        minOccurs="0">
                    </element>
                    <element name="exception-strategy"
                        type="onx:ct_exception_strategy_impl"
maxOccurs="unbounded"
                        minOccurs="0">
                    </element>
                </sequence>
        </complexType>

        <element name="onix-config" type="onx:onix-config"></element>

        <complexType name="ct_send">
                <sequence>
                    <element name="incoming" type="onx:ct_endpoints"
                        maxOccurs="1" minOccurs="1">
                    </element>
                    <element name="validators" type="onx:ct_validators"
                        maxOccurs="1" minOccurs="0">
                    </element>
                    <element name="enrichers" type="onx:ct_enrichers"
                        maxOccurs="1" minOccurs="0">
                    </element>

                    <element name="routers" type="onx:ct_routers" maxOccurs="1"
                        minOccurs="1">
                    </element>
                </sequence>
        </complexType>

        <complexType name="ct_receive">
                <sequence>
                    <element name="enrichers" type="onx:ct_enrichers"
                        maxOccurs="1" minOccurs="0">
                    </element>
                    <element name="validators" type="onx:ct_validators"
                        maxOccurs="1" minOccurs="0">
                    </element>
                </sequence>

        </complexType>
```

```
	<complexType name="ct_properties">
		<sequence>
			<element name="property" type="onx:ct_property"
maxOccurs="unbounded" minOccurs="0"></element>
		</sequence>
	</complexType>


	<complexType name="ct_host">
		<sequence>
			<element name="host" type="string" maxOccurs="1"
minOccurs="1"></element>
			<element name="port" type="string" maxOccurs="1"
minOccurs="1"></element>
		</sequence>
	</complexType>

	<complexType name="ct_broadcasting_router">
		<sequence>

			<element name="outgoing-endpoint"
				type="onx:ct_endpoint_decl" maxOccurs="unbounded"
minOccurs="1">
			</element>

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

	</complexType>

	<simpleType name="st_protocols">
		<restriction base="string">
			<enumeration value="TCP"></enumeration>
			<enumeration value="HTTP"></enumeration>
			<enumeration value="FILE"></enumeration>
			<enumeration value="EMAIL"></enumeration>
			<enumeration value="WS_AXIS"></enumeration>
			<enumeration value="WS_CXF"></enumeration>
		</restriction>
	</simpleType>
</schema>
```