UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA DE ELETRICIDADE
CURSO DE PÓS-GRADUAÇÃO EM ENGENHARIA DE ELETRICIDADE

RÔMULO MARTINS FRANÇA

UM INTERCEPTADOR BASEADO EM AOP PARA TRATAR INTERESSES TRANSVERSAIS EM SERVIÇOS WEB

São Luís
2008

RÔMULO MARTINS FRANÇA

UM INTERCEPTADOR BASEADO EM AOP PARA TRATAR INTERESSES TRANSVERSAIS EM SERVIÇOS WEB

Dissertação de Mestrado em Engenharia de Eletricidade, na área de Ciência da Computação, apresentada à Coordenação do Programa de Pós-Graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão.

Orientador: Prof. Dr. Sofiane Labidi

São Luís
2008

França, Rômulo Martins

Um interceptador baseado em AOP para tratar interesses transversais em serviços Web. Rômulo Martins França. São Luís, 2008.
108 f.

Impresso por computador (fotocópia).
Orientador: Sofiane Labidi.
Dissertação (Mestrado) - Universidade Federal do Maranhão, Programa de Pós-Graduação em Engenharia de Eletricidade, 2008.

1. Serviços Web. 2. Arquitetura Orientada a Serviços. 3. Programação Orientada a Aspectos. I. Título

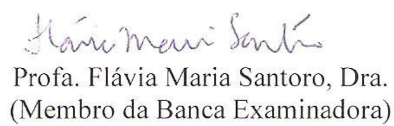
CDU 004.738.52

UM INTERCEPTADOR BASEADO EM AOP PARA TRATAR INTERESSES TRANSVERSAIS EM SERVIÇOS WEB

Rômulo Martins França

Dissertação aprovada em 12 de março de 2008.

Prof. Sofiane Labidi, Dr.
(Orientador)



Profa. Flávia Maria Santoro, Dra.
(Membro da Banca Examinadora)

Prof. Zair Abdelouahab, Ph.D.
(Membro da Banca Examinadora)

AGRADECIMENTOS

Primeiramente, agradeço a Deus por me dar a vida, saúde e força para chegar aos meus objetivos. E também de me tornar uma pessoa cada vez melhor no sentido de fazer o bem ao próximo.

A meu querido pai, França, minha querida mãe, Lúcia e ao meu irmão, Lucas, que acreditaram e investiram em mim, realizando muitos esforços, para que eu chegasse a esse momento, que nunca irei esquecer e muito menos esquecê-los.

A minha filha, Maria Clara, que está prestes a nascer. Que venha com muita saúde, paz e amor.

A Rhaíssa, minha namorada, pela compreensão, atenção, carinho e força.

Aos meus avôs e avós. Que Deus ilumine suas vidas e que a gente possa viver mais momentos juntos.

Aos meus padrinhos Raimundo e Roselene por sempre nos terem dado força em nossas vidas.

A tia Helma que me ajudou em vários momentos durante a minha graduação.

Ao professor Dr. Sofiane Labidi que profissionalmente confiou em mim em todos os momentos acadêmicos e profissionais.

Ao Ricardo Ferraz, Thiago Drummond e Valeska Trinta pela parceria, paciência e pelos os grandes ensinamentos adquiridos.

Aos meus amigos do laboratório de sistemas inteligentes (LSI).

Aos professores, funcionários e alunos do CPGEE.

Aos meus amigos da FAPEMA.

E a todos aqueles que contribuíram diretamente ou indiretamente para a realização desse trabalho.

Dedico este trabalho a

Deus, minha família e amigos pelo apoio e incentivo.

“O único lugar onde o sucesso vem antes do trabalho é no dicionário.”

Albert Einsten

RESUMO

Esta dissertação apresenta a descrição de um projeto atualmente em desenvolvimento no laboratório de sistemas inteligentes (LSI), da Universidade Federal do Maranhão (UFMA). Os *middlewares* ESB (*Enterprise Services Bus*) atuais como o BEA Web Logic, BizTalk, Mule ESB e similares, não possuem uma semântica para o tratamento dos interesses transversais antes, durante e depois de um grupo de operações serem executadas. Muitos interesses importantes estão espalhados por vários módulos, serviços e se misturam com outras propriedades de sistema de maneira intrusiva, dificultando a reutilização e manutenção de seus componentes. Este trabalho apresenta uma abordagem chamada de InterceptadorAOP, para o tratamento de interesses transversais em Serviços Web em *middlewares* ESB. Os InterceptadoresAop são elementos responsáveis pelo tratamento dos interesses transversais como o log de dados, tratamento de exceções, *debug* e medidor de tempo já pré-definidos. São baseados na semântica da linguagem AspectJ, oriunda da Programação Orientada a Aspectos que visa fornecer uma melhor separação dos interesses funcionais dos não-funcionais de uma aplicação, promovendo serviços mais fáceis de serem mantidos, legíveis e modularizados. Já a Arquitetura Orientada a Serviço (SOA) estimula e oferece mecanismos para a composição de aplicações distribuídas de forma flexível e com custo reduzido. O trabalho descreve o estado da arte, detalhes técnicos dos InterceptadoresAOP e a sua aplicação em dois cenários para a validação do modelo.

Palavras-Chave: Serviços Web, Arquitetura Orientada a Servicos, Programação Orientada a Aspectos.

ABSTRACT

This research currently presents the description of a project in development in the laboratory of intelligent systems (LSI), of the Federal University of the Maranhão (UFMA). The current Middlewares ESB such as the BEA Web Logic, BizTalk, Mule ESB and similars, does not possess a semantics for the treatment of the crosscutting concerns neither before, during and after a group of operations being executed. Many important concerns are spread by some modules, services and if they mix other properties of system in an inner way, making it difficult the reuse and maintenance of its components. This work presents a boarding called InterceptadorAOP, for the treatment of crosscutting concerns in Web Services in the middlewares ESB. The InterceptadoresAop is responsible elements for the treatment of the crosscutting concerns as log of data, treatment of exceptions, debug and daily pay-defined measurer of time already. They are based on the semantics of the AspectJ language, deriving of the Aspect-oriented Programming that it aims at to supply one better separation of the functional interests of the non-functional of an application, promoting services more easy to be kept, legible and modularized. Already the Architecture-oriented Services stimulates and offers mechanisms for the composition of distributed applications of flexible form and with reduced cost. The research describes the state of the art, details technician of the InterceptadoresAOP and its application in two scenes for the model validation.

Keywords: Web Services, Service-oriented Architecture, Aspect-Oriented Programming.

SUMÁRIO

LISTA DE FIGURAS 11

LISTA DE TABELAS 12

LISTA DE ABREVIATURAS E SIGLAS 13

1. INTRODUÇÃO 14
   1.1 CONTEXTO 14
   1.2 DESCRIÇÃO DO PROBLEMA 15
   1.3 OBJETIVOS 17
   1.4 ORGANIZAÇÃO 18

2. SOA: ARQUITETURA ORIENTADA A SERVIÇOS 19
   2.1 FUNDAMENTOS E PRÍNCIPIOS 19
   2.2 BENEFÍCIOS DA SOA 20
   2.3 ESB (ENTERPRISE SERVICE BUS) 21
   2.4 MIDDLEWARES ESB 22
      2.4.1 Plataforma BEA WebLogic Entreprise 23
      2.4.2 Microsoft BizTalk Server 2004 25
      2.4.3 Mule ESB Open Source 26

3. SERVIÇOS WEB 30
   3.1 CONCEITOS 30
   3.2 ANTECESSORES 31
      3.2.1 CORBA (Common Object Request Broker Architecture) 32
      3.2.2 COM e DCOM 34
   3.3 TECNOLOGIAS ASSOCIADAS AOS SERVIÇOS WEB 35
      3.3.1 XML (eXtensible Markup Language) 37
      3.3.2 WSDL (Web Services Description Language) 38
      3.3.3 SOAP (Simple Object Acess Protocol) 39
      3.3.4 HTTP (HyperText Transfer Protocol) 40
      3.3.5 UDDI (Universal Discovery Description Integration) 41

4. SOC – SEPARAÇÃO DE INTERESSES 43
   4.1 ORIGENS E FUNDAMENTOS 43
   4.2 PROGRAMAÇÃO PROCEDURAL 44
   4.3 PROGRAMAÇÃO MODULAR 44
   4.4 PROGRAMAÇÃO ORIENTADA A OBJETOS (OOP) 46
      4.4.1 Exemplo de uma Classe Orientada a Objetos 49
   4.5 PROGRAMAÇÃO ORIENTADA A ASPECTOS (AOP) 50
      4.5.1 Fundamentos 50
      4.5.2 AspectJ 53
      4.5.2.1 Aspecto (Aspect) 54
      4.5.2.2 Pontos de Junção (Join Points) 55
      4.5.2.3 Pontos de Atuação (Pointcuts) 55
      4.5.2.4 Advices 57
      4.5.2.5 Inserção (Introduction) 58
   4.6 EXEMPLO ORIENTADO A ASPECTOS 59

5. INTERCEPTADORAOP 62
   5.1 INTRODUÇÃO 62
   5.2 TRABALHOS RELACIONADOS 63
   5.3 MODELO PROPOSTO 66

5.4 DETALHAMENTO DO MODELO 68
5.5 FERRAMENTAS 75
5.6 CONFIGURAÇÃO DO AMBIENTE 76
5.7 AVALIAÇÃO DO MODELO 78
5.7.1 Cenários 79
5.7.2 Cenário 1 79
5.7.2.1 Arquitetura 81
5.7.2.2 Implementação e Execução 83
5.7.3 Cenário 2 85
5.7.3.1 Arquitetura 86
5.7.3.2 Implementação e Execução 88

6. CONCLUSÃO E TRABALHOS FUTUROS 91

6.1 CONTRIBUIÇÕES DO TRABALHO 91
6.2 CONSIDERAÇÕES FINAIS 92
6.3 LIMITAÇÕES 93
6.4 TRABALHOS FUTUROS 94

REFERÊNCIAS 95

APENDICE A – CÓDIGO FONTE DO CENÁRIO 1 100

APENDICE B – CÓDIGO FONTE DO CENÁRIO 2 103

APENDICE C – CÓDIGO DOS INTERCEPTADORES AOP 105

LISTA DE FIGURAS

Figura 1: Utilização de um ESB para integração de Aplicativos SOA..22
Figura 2: Plataforma BEA WebLogic. 24
Figura 3: Mule Manager...27
Figura 4: Diagrama com os componentes da arquitetura do Mule..27
Figura 5: Transformador de entrada. 29
Figura 6: Transformador de saída..29
Figura 7. Interações em uma Arquitetura Orientada a Serviços Web..31
Figura 8: Object Management Architecture...32
Figura 9: Estrutura da Interface CORBA. 33
Figura 10: Arquitetura DCOM...34
Figura 11. Camadas conceituais dos Serviços Web. 36
Figura 12. Elementos do WSDL. 38
Figura 13. Estrutura de uma mensagem SOAP...39
Figura 14. Estrutura do UDDI. 42
Figura 15. Reuso de módulo por colaboração. 45
Figura 16. Troca de mensagem. 47
Figura 17. Exemplo de uma classe em JAVA. 49
Figura 18. Separação de Interesses na AOP..51
Figura 19. Paradigmas de programação mais antigos com códigos de interesses transversais espalhados nas regras de negócios. 52
Figura 20. Exemplo de um Ponto de Atuação..55
Figura 21. Exemplo de *Advice*..57
Figura 22. Classe Conta - código sem interesses transversais misturados. 60
Figura 23. AspectoLog.aj - código referente ao Aspecto que influi na classe Conta. 61
Figura 24. Resultado da Implementação entre a Classe Conta e o Aspecto AspectoLog. 61
Figura 25. Processamento de Mensagens do Mule ESB sem os InterceptadoresAOP. 65
Figura 26. Processamento de Mensagens do Mule ESB com os Interceptadores AOP. 67
Figura 27. Processamento de Mensagens utilizando o InterceptadorLogAop. 69
Figura 28. Saída padrão da execução do InterceptadorLogAop. 70
Figura 29. Processamento de Mensagens utilizando o InterceptDebugAop. 71
Figura 30. Saída padrão da execução do InterceptDebugAop. 72
Figura 31. Processamento de Mensagens utilizando o InterceptTempoAop. 72
Figura 32. Saída padrão da execução do InterceptTempoAop em tempo total de 0.032 segundos. 73
Figura 33. Processamento de Mensagens utilizando o InterceptExcecaoAop...74
Figura 34. Saída padrão da execução do InterceptExcecaoAop...75
Figura 35: Criação de um projeto com suporte a AspectJ. 77
Figura 36: Variáveis de ambiente do MULE no Eclipse. 78
Figura 37. Arquitetura do cenário 1...82
Figura 38. Exemplo do Serviço Web ComprovanteResidencia sem InterceptadoresAOP.....................83
Figura 39. Execução do Serviço Web ComprovanteResidencia com InterceptadorTempoAOP............84
Figura 40. Execução do Serviço Web ComprovanteResidencia com o InterceptadorExcecaoAop. 85
Figura 41. Arquitetura do cenário 2...87
Figura 42. Parte do código do cliente Mule. 88
Figura 43. Execução do serviço web SituacaoPessoa com os InterceptadoresAOP. 89

LISTA DE TABELAS

Tabela 1: Componentes da Plataforma BEA WebLogic 24
Tabela 2. Tipo de caracteres especiais (curingas). 56
Tabela 3. Tipo de designadores do ponto de atuação. 56
Tabela 4. Tipo de designadores. 58
Tabela 5. Pontos de atuação na forma de enumerações. 62

LISTA DE ABREVIATURAS E SIGLAS

AOP	Programação Orientada a Aspectos
BPEL	Business Process Execution Language
BPM	Business Process Management
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
DCOM	Distributed Component Object Model
ESB	Enterprise Service Bus
HTTP	HyperText Transfer Protocol
IDL	Interface Definition Language
JBDC	Java Data Base Connectivity
JMS	Java Message Service
MTS	Microsoft Transaction Server
OLE	Object Linked and Embedding
OMA	Object Management Architecture
OMG	Object Management Group
OOP	Programação Orientada a Objetos
ORB	Object Request Broker
RPC	Chamada de Procedimento Remoto
SMTP	Simple Mail Transfer Protocol
SOA	Arquitetura Orientada a Serviços
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
UDDI	Universal Description Discovery and Integration
UMO	Objeto de Mensagem Universal
W3C	World Wide Web Consortium
WSDL	Web Services Description Language
XML	Extensible Markup Language
XSD	Validator Selected Reading

1. INTRODUÇÃO

1.1 Contexto

As constantes e crescentes transformações no mundo dos negócios estão tornando o mercado mais competitivo e conduzindo as organizações a repensarem suas estratégias de atuação, dentro de seus respectivos nichos. As empresas são forçadas a se adaptar rapidamente às mudanças do ambiente no qual estão inseridas.

Vernadat (1996) ressalta a importância e a necessidade das organizações apresentarem flexibilidade tanto no nível gerencial, o que representa a reorganização dos processos de negócios e estruturas organizacionais, quanto no nível técnico, ou seja, a capacidade de modificar rapidamente as tecnologias, requisitos, serviços, sistemas de informação e as características de projeto que compõem o produto oferecido em função das tendências do mercado e das customizações requeridas pelos clientes.

Na comunidade de Engenharia de Software, é bem conhecido e aceito que aplicações são mais frequentemente modificadas e evoluídas que escritas (THOMAS, 1996).

Uma das principais limitações das tecnologias atuais de processos de desenvolvimento de software é a falta de flexibilidade no tratamento de mudança à medida que os negócios evoluem (ARBAOUI et al., 2002; FUGGETTA, 2000). Outra importante limitação é a falta de interoperabilidade com outras ferramentas e tecnologias (PRADO et al., 2005).

Diversos pesquisadores da comunidade da engenharia de software (FILMAN et al, 2005) (KICZALES et al, 1997) (TARR, et al 1999) têm identificado que as abstrações e os mecanismos de composição do paradigma Orientado a Objeto possuem limitações para separar e compor alguns interesses relevantes em sistemas de software. Assim, muitos interesses importantes, espalham-se por vários módulos e se misturam com outras propriedades de um sistema de maneira

intrusiva, dificultando a reutilização e manutenção de seus componentes. Alguns exemplos clássicos destas propriedades são log de dados, tratamento de exceções, controle de concorrência, desempenho, segurança e persistência.

A Programação Orientada a Aspectos é um paradigma de programação que visa fornecer uma melhor separação dos interesses funcionais dos não-funcionais de uma aplicação (KICZALES et al., 1997). Ela se propõe a resolver algumas das limitações dos paradigmas de desenvolvimento de softwares atuais, como por exemplo, o entrelaçamento e a dispersão de código referente aos interesses transversais (do inglês, *crosscutting concerns*) da aplicação.

Harry Sneed mencionou, durante um painel da IEEE especialmente organizado para discutir a manutenção e evolução dos Serviços Web, que os Serviços Web introduziram um conceito totalmente novo de organização, no qual é chamado de Centros de Serviços. Estas organizações estão focadas na criação de novos Serviços Web e na publicação dos mesmos em diretórios de serviços.

Ainda, de acordo com Harry Sneed, a evolução e manutenção dos Serviços Web podem ser muito custosa para muitos Centros de Serviços. Pois eles têm que possuir engenheiros altamente experientes, comprar ferramentas, softwares necessários e fazer reorganizações necessárias dentro de suas empresas" (KAJKO-MATTSSON, 2004).

1.2 Descrição do Problema

O ESB (do inglês, *Entreprises Services Bus*) é uma camada no qual serviços são expostos e consumidos. Viabiliza para que os serviços, independemente de sua implementação, possam se comunicar entre si. Facilitam o compartilhamento de serviços dentro das organizações, fornece transparência na localização do serviço e a habilidade de separar serviços em termos de habilidades de negócio de sua implementação real.

Os *middlewares* ESB atuais como o BEA Web Logic, BizTalk, Mule ESB, ServicesMix e similares, não possuem uma semântica para o tratamento dos

interesses não-funcionais, também conhecidos como interesses transversais, antes, durante e depois de um grupo de operações.

Segundo Hilsdale et al. (2001), muitos interesses podem ser separados do código da aplicação. Entretanto, não são necessariamente tranversais. Por exemplo, o método *sqrt*() pode ser invocado em muitas classes em diferentes locais, a fim de calcular a raiz quadrada de um número. Entretanto não significa que *sqrt*() é um interesse transversal.

A possibilidade de enumerar locais onde um comportamento pode ser introduzido determina se ele se trata de um comportamento transversal ou não. Por exemplo, pode se dizer "dentro do pacote de todas as classes do Apache Tomcat, registre um log toda vez que um método update é executado." (HILSDALE et al., 2001). Tais declarações normalmente não fazem sentido quando aplicados a função *sqrt*().

Com o BizTalk da empresa Microsoft, por exemplo, podem-se compor interesses transversais em tempo de projeto em cada método do serviço, mas não aplicá-los para uma enumeração. Semelhante ao que se faz com *triggers* em Banco de Dados, utilizando uma *trigger* pode-se dizer para banco "*emita uma mensagem sempre que um update acontecer na tabela x*" (JEFFREY GRAY, 2002).

Soluções intrusivas, semelhantes ao descrito no artigo de Bonér e Vasseur (2004), podem ser obtidas com ferramentas AOP como JBossAOP, SpringAOP, Dynaop e o AspektWerkz. Mas serviços, *a* priori, não têm o conhecimento da estrutura interna, são componentes *black box*, então não tem como declarar pontos de junção de uma maneira útil.

Como uma alternativa para resolver estes problemas, apresenta-se uma abordagem para *middlewares* ESB, chamada de InterceptadorAOP, para resolver problemas relacionados as possibilidades de enumeração, permitindo uma semântica no tratamento dos interesses transversais nos Serviços Web, antes, durante e depois de um grupo de operações. São baseados na semântica da linguagem AspectJ, oriunda da Programação Orientada a Aspectos que visa fornecer uma melhor separação dos interesses funcionais dos não-funcionais de

uma aplicação, promovendo serviços mais fáceis de serem mantidos, legíveis e modularizados (KICZALES et al., 1997). E a Arquitetura Orientada a Serviço (SOA) oferece a infra-estrutura tecnológica necessária para que uma aplicação possa ser definida por meio da composição de Serviços Web. Desta forma, SOA estimula e oferece mecanismos nos InterceptadoresAOP para a composição de aplicações distribuídas de forma flexível e com custo reduzido, permitindo integrar novas aplicações com aplicações já existentes. (PAPAZOGLOU; GEORGAKOPOULOS, 2003).

1.3 Objetivos

O objetivo deste trabalho é apresentar uma solução para um dos *middlewares* ESB, o Mule ESB, permitindo assim uma semântica AOP no tratamento dos interesses transversais, antes, durante e depois de um grupo de operações nos Serviços Web.

Especificamente, pretende-se:

1. Permitir uma melhor compreensão dos conceitos, padrões e ferramentas decorrentes da Arquitetura Orientada a Serviços, Serviços Web e da Programação Orientada a Aspectos;

2. Propor a semântica AOP como uma alternativa para tratamento dos interesses transversais nos Serviços Web em *middlewares* ESB;

3. Amenizar os problemas de manutenção, legibilidade, modularidade e flexibilidade nos Serviços Web, no que concerne ao tratamento de mudança à medida que os negócios das organizações evoluem, em função dos códigos de negócio e não funcionais estarem menos espalhados e emaranhados.

4. Experimentar e verificar a eficácia dos conceitos, padrões e soluções apresentadas quando aplicados a casos reais.

5. Utilizar o Mule ESB *Open Source*, como *middleware* ESB de modelo para a demonstração do modelo;

1.4 Organização

Esta dissertação está organizada em seis capítulos. O capítulo 1 é constituído por esta introdução, onde é contextualizado o trabalho, descrição da problemática e seus objetivos.

Nos Capítulos 2, 3, 4, tenta-se alcançar o primeiro objetivo do trabalho: fazer a revisão bibliográfica e apresentar o referencial teórico necessário para o entendimento da solução proposta. Esta parte é dividida em três capítulos: O Capítulo 2 apresenta uma visão da Arquitetura Orientada a Serviços – conceitos, benefícios e componentes; Capítulo 3 compila-se os conceitos necessários ao entendimento da Tecnologia dos Serviços Web, discorrendo sobre os principais conceitos, antecessores e tecnologias associadas; Capítulo 4 descreve os fundamentos da Separacão de Interesses abordando alguns Paradigmas de Linguagens de Programação com os conceitos e alguns exemplos; Apresentando também uma visão detalhada da Programação Orientada a Aspectos (AOP) – definição, linguagem de aspectos e um exemplo orientado a aspecto;

No Capítulo 5 espera-se alcançar o segundo e o terceiro objetivo do trabalho: Apresentar os trabalhos relacionados a proposta, explicar e detalhar seu modelo, apresentar as ferramentas utilizadas para o desenvolvimento da solução, definir cenários e demonstrar uma avalição do modelo, implementação e execução da solução.

Por fim, o Capítulo 6 apresenta as conclusões deste trabalho de pesquisa, contribuições, considerações finais, limitações e sugestões para trabalhos futuros.

2. SOA: ARQUITETURA ORIENTADA A SERVIÇOS

Este capítulo apresenta os principais conceitos e características relacionadas à Arquitetura Orientada a Serviços, bem como a contribuição e adequação do uso desta para este trabalho.

2.1 Fundamentos e Príncipios

SOA é uma abordagem para o desenvolvimento de sistemas distribuídos que oferece funcionalidades de aplicação como serviços, para a construção de aplicações para o usuário final ou outros serviços.

Pode ser baseada em Serviços Web ou utilizada com outras tecnologias. Ao utilizar SOA para projetar aplicativos distribuídos, pode-se expandir o uso de Serviços Web de um simples modelo cliente servidor para sistemas de complexidade arbitrária (COLAN, 2004).

O seu objetivo principal, segundo a IBM e OMG (apud. LINTHICUM, 2006), é de prover serviços de software de uma maneira mais dinâmica que a usual, com redução de custos e simplificação nas integrações da grande quantidade de sistemas existentes atualmente, tais como sistemas de relacionamento com o cliente (CRM - *Customer Relationship Management*), administração corporativa (ERP - *Enterprise Resource Planning*), gerenciamento, entre outros (IBM, 2007).

Ainda segundo a IBM (2007), SOA como um paradigma, transfere a preocupação dos detalhes de implementação (em se tratando de projeto de sistemas de informação) para uma preocupação maior com os grandes componentes do sistema.

Um serviço em SOA deve ser fracamente acoplado, independentes de localização física, trabalhar em conjunto com outros serviços, permitir composição e possuir um registro de serviços, no qual todos eles são catalogados (IBM, 2007).

Segundo Papazoglou e Georgakopoulos (2003), SOA utiliza Serviços Web como elementos fundamentais para o desenvolvimento de aplicações distribuídas. Em SOA, a comunicação entre os serviços acontece via troca de mensagens.

2.2 Benefícios da SOA

Abaixo, são citados os seguintes benefícios da SOA para as organizações como, por exemplo:

- Permitir a reutilização de software, minimizando custos e esforços de *deployment*.
- Independência da infra-estrutura das aplicações e da plataforma tecnológica.

Outros benefícios podem ser agrupados conforme abaixo:

Para os negócios:

- Eficiência: Transforma os processos de negócios em serviços compartilhados com um menor custo de manutenção.
- Capacidade de Resposta: Rápida adaptação e *deployment* de serviços, o que é fundamental para responder as demandas dos clientes e parceiros de negócios.
- Adaptabilidade: Facilita a adoção das mudanças adicionando flexibilidade e reduzindo o esforço.

Para a Tecnologia da Informação:

- Reduz a complexidade graças à interoperabilidade baseada em padrões de integração ponto a ponto.
- Reutilizam os serviços compartilhados que podem ter sido desenvolvidos anteriormente.

- Integra aplicações herdadas limitando os custos de manutenção e integração.

2.3 ESB (Enterprise Service Bus)

Camada no qual serviços são expostos e consumidos. Em geral, um ESB suporta uma série de transportes. Frequentemente são utilizados de modo que os sistemas existentes possam ser expostos como serviços. Além disso, resolve aspectos de qualidade de serviço, como a validação dos dados ou a segurança que resultam da exposição de serviços a uma variedade de clientes.

Segundo Botto (2004), ESB é uma camada lógica de mensagens compartilhadas para interoperabilidade de aplicações e serviços entre organizações. Faz a transformação e roteamento de mensagens. Segundo esse mesmo autor, ESB tem as seguintes características tradicionais e algumas novas:

- Apóia-se na Arquitetura Orientada a Serviços;
- Possui funções de transformação, roteamento e gerência;
- Suporte a diversos padrões de interfaces e conectores;
- Baseia-se em padrões abertos e de segurança;
- Operação e gerência distribuída – e não em um ponto integrador centralizado;
- Possui serviços de repositório;
- Se alavanca em legados de JAVA e .NET;

Nott (2004) define que um ESB fornece um conjunto de capacidades implementadas pela tecnologia de middleware, que permite a integração de serviços dentro de uma arquitetura orientada a serviço, conforme mostra a Figura 1 abaixo.

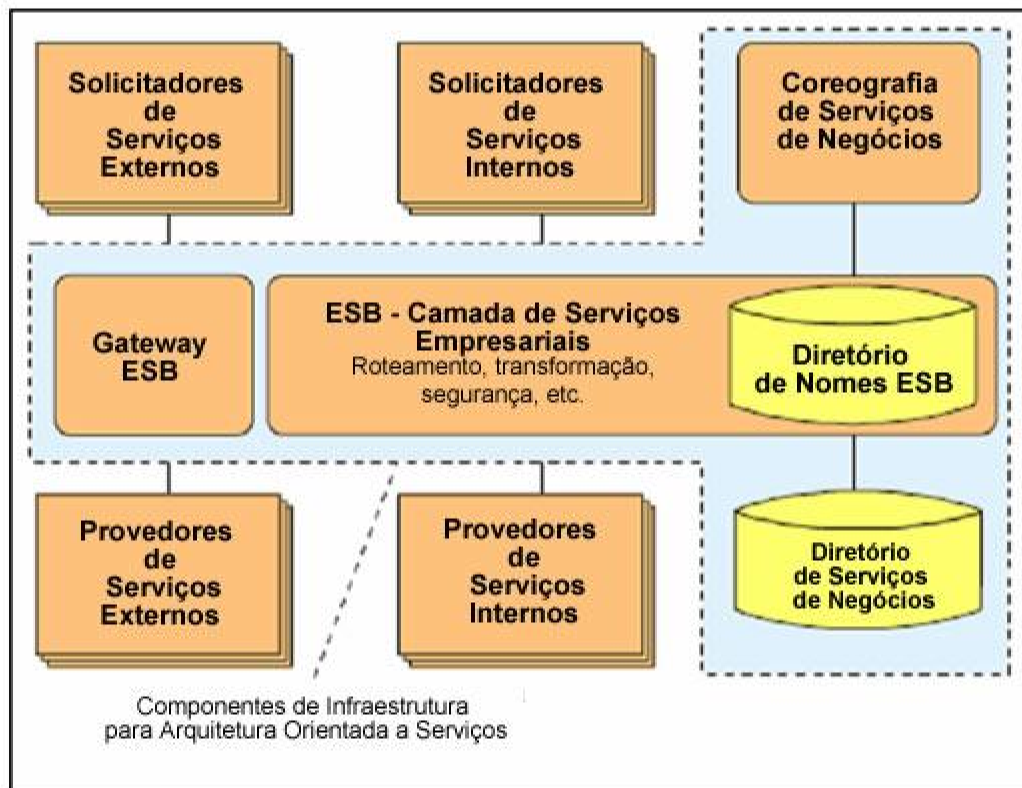


Figura 1: Utilização de um ESB para integração de Aplicativos SOA
Fonte: Nott , 2004.

Considerando a coreografia dos serviços do negócio como um componente, podem-se estabelecer os seguintes papéis para os demais componentes:

- Diretório *Namespace* ESB: fornece informação para permitir o roteamento das interações de serviço.
- Diretório de Serviços de Negócio: fornece uma taxonomia e detalhes de serviços disponíveis para sistemas participantes de uma Arquitetura Orientada a Serviço.
- *Gateway* de Serviços ESB: é usado para fornecer um ponto controlado de acessos externos aos serviços.

2.4 Middlewares ESB

A disseminação no uso de SOA e Serviços Web nos últimos anos, incentivou o mercado a oferecer uma grande variedade de ferramentas e aplicações para prover suporte a estas tecnologias.

ESB é uma das peças fundamentais dentro da arquitetura SOA, pois é ele que viabiliza que os serviços, independemente de sua implementação, possam se comunicar entre si. Facilita o compartilhamento de serviços dentro das organizações, fornece transparência na localização do serviço e a habilidade de separar serviços em termos de habilidades de negócio da sua implementação real.

Existem diversas funcionalidades que podem ser encontradas em um ESB, tais como: roteamento, segurança, gerenciamento de transações, orquestração de serviços, coreografia de processos, processamento de mensagens, mapeamento de serviços, transformação de protocolos, enriquecimento e transformação de mensagens.

Atualmente, as principais plataformas são as dos fornecedores: Sun Microsystems, IBM, BEA, Apache, Mule Open Source, Systinet e Microsoft.

A seguir, são descritos os conceitos de alguns *middlewares* ESB com o objetivo de citar as tecnologias disponíveis durante a realização deste trabalho de pesquisa.

2.4.1 Plataforma BEA WebLogic Entreprise

Desenvolvida pela empresa Bea Systems especializada em soluções de infra-estrutura de software para conexão com ambiente web. A família de produtos da BEA chama-se *WebLogic*.

BEA Web Logic Enterprise é uma plataforma voltada para a integração e o desenvolvimento de aplicações. Ela oferece um desenvolvimento e framework em tempo real que unifica todos os componentes de integração de negócios - gerenciamento de processos negócios, transformação de dados, parceiros comerciais de integração, conectividade, corretor de mensagem, monitoramento de aplicações e interação de usuários em um único ambiente flexível (BEA SYSTEMS, 2003).

A plataforma BEA WebLogic Enterprise como mostrado na Figura 2, fornece um conjunto integrado de componentes: um servidor de aplicação compatível J2EE

com o desenvolvimento, integração e um framework de portais (BEA SYSTEMS, 2003).

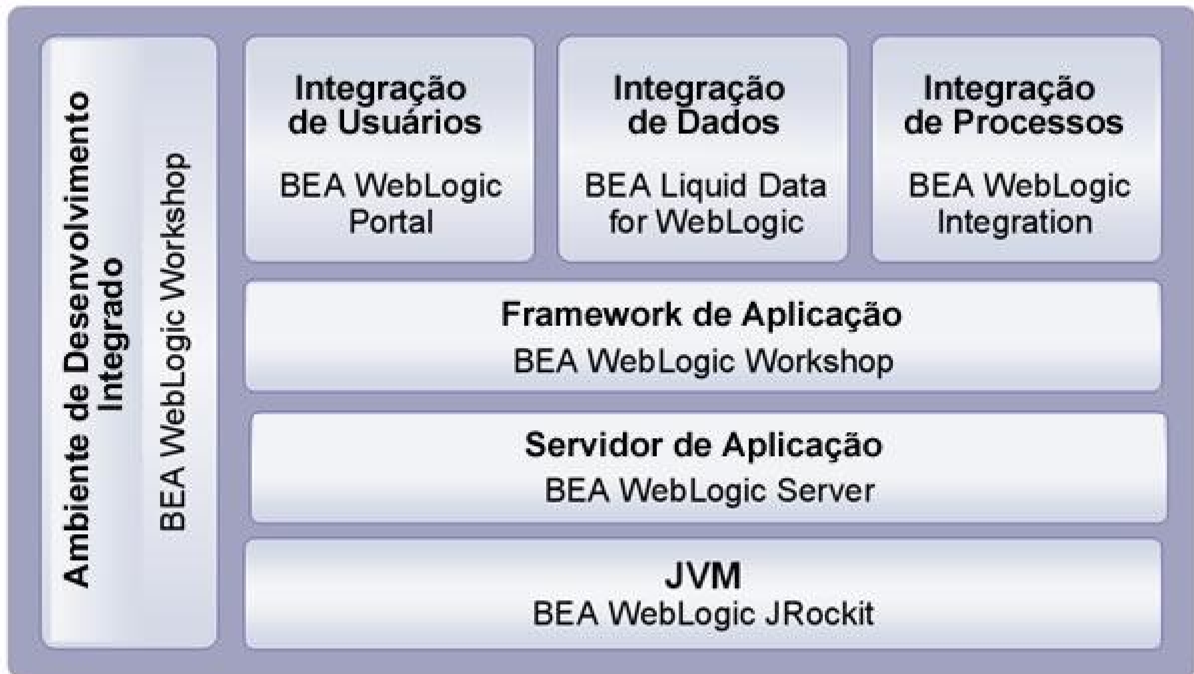


Figura 2: Plataforma BEA WebLogic.
Fonte: BEA SYSTEMS, 2003.

Dentre os seus objetivos estão reduzir os custos de gerenciamento e operações com condições altamente confiáveis, estáveis, escaláveis e com a missão crítica de integração de soluções.

Tabela 1: Componentes da Plataforma BEA WebLogic
Fonte: BEA Systems, 2003.

Componente	Funções
WebLogic Workshop	Desenvolvimento integrado, *deployment* e ambiente em tempo-real para construção de aplicações na Plataforma WebLogic.
WebLogic Portal	Framework para desenvolvimento e gerenciamento de portais corporativos.
WebLogic Integration	Framework para desenvolvimento, integração de aplicações e processos de negócio de dentro e fora das empresas.
WebLogic Server	Base sólida para o rápido desenvolvimento, *deployment* e gerenciamento de aplicações *e-business*.
WebLogic JRockit	JVM (Java Virtual Machine) otimizado para servidores laterais e escaláveis.

Na Tabela 1 estes componentes têm suporte para processos BPM (*Business Process Management*) e BPEL (*Business Process Execution Language*) como padrão para o desenvolvimento dos Serviços Web. A suíte da BEA busca deste modo simplificar a construção de produtos para Arquiteturas Orientadas a Serviço.

2.4.2 Microsoft BizTalk Server 2004

É uma plataforma para aplicações empresariais e gerenciamento de processos de negócio. Ela fornece ferramentas visuais para desenho e desenvolvimento de integrações entre origem de dados, criação e manutenção de processos de negócio que são impactados na aplicação.

O BizTalk Server possui a capacidade de separar estas duas funções, permitindo que os usuários de negócio possam focar em regras e definições de processos, enquanto os desenvolvedores podem focar nos detalhes técnicos de conexão de origem de dados que guiam à aplicação.

Microsoft BizTalk proporciona capacidades para a integração de aplicações e funcionalidades (PULIDO, 2004):

- Ambiente de desenvolvimento: completamente baseado no Visual Studio .NET.
- Ferramenta orientada a funções dos usuários: motor de regras de negócio, BAM (Monitoramento de Atividades de Negócio), modelo de projeto de processos na ferramenta Visio da Microsoft, BAS (Serviços de Atividades de Negócio) e gestão de entidades externas;
- Ambiente Administrativo: console de administração, HAT (Rastreador de Atividades), depuração e monitoração de processos de negócio em tempo real;
- Motor de regras de negócio: criação, administração e instalação de regras de negócio que podem ser utilizadas pelos desenvolvedores;

- BAM (Monitoramente de Atividades de Negócio): possibilidade de obter informações em tempo real e modificar os processos, consulta de dados agregados, uso de dados de documentos e processos, complemento das soluções inteligentes de negócio em SQL;
- Suporte a padrões: WSDL, XML, XSD, BPEL;
- Melhor escalabilidade horizontal na criação de servidores sem estado: execução das orquestrações com rapidez e suporte para mensagens de grande tamanho;

2.4.3 Mule ESB Open Source

Para que seja possível entender a abordagem proposta, faz-se necessário o entendimento da arquitetura e dos componentes do Mule ESB.

O Mule ESB é uma plataforma de transferência de mensagens em arquiteturas ESB. O seu núcleo do está baseado no container de serviço SEDA, que gerencia serviços de objetos, conhecidos como Objetos de Mensagem Universal (UMOs, do inglês *Universal Message Objects*), que são simples objetos Java.

Todas as comunicações entre UMOs e outras aplicações são realizadas através de mensagens *endpoints*. Estes *endpoints* proporcionam uma simples e consistente interface para tecnologias como JMS (*Java Message Service*), SMTP (*Simple Mail Transfer Protocol*), JBDC (*Java Data Base Connectivity*), TCP (*Transmission Control Protocol*), HTTP (*HyperText Transfer Protocol*), arquivos, etc.

O *Mule Manager* é parte central de uma instância do servidor Mule (Figura 3). Seu primeiro papel é gerenciar os objetos tais como conectores, *endpoints* e transformadores para uma instância Mule. Estes objetos são empregados para controlar fluxos de mensagens e de serviços/componentes.

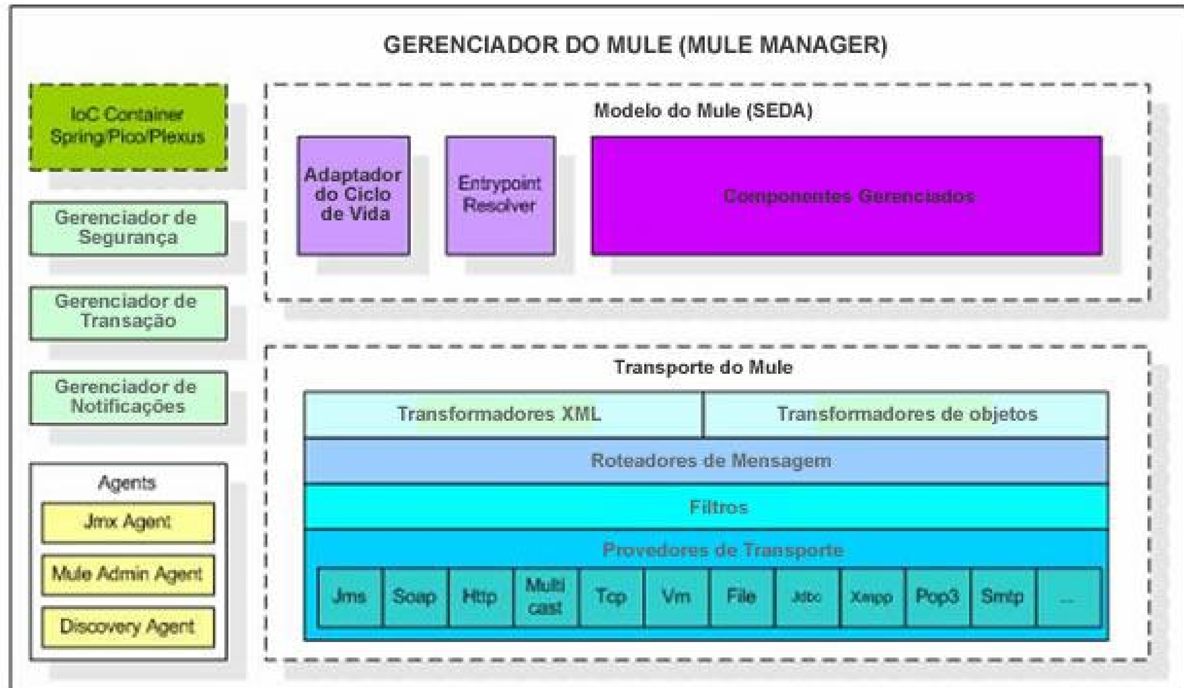


Figura 3: Mule Manager.
Fonte: Mule Esb, 2006.

A Figura 4 apresenta um diagrama com os componentes da arquitetura do Mule. O *Mule Manager* gerencia todos os objetos e componentes contidos na instância.

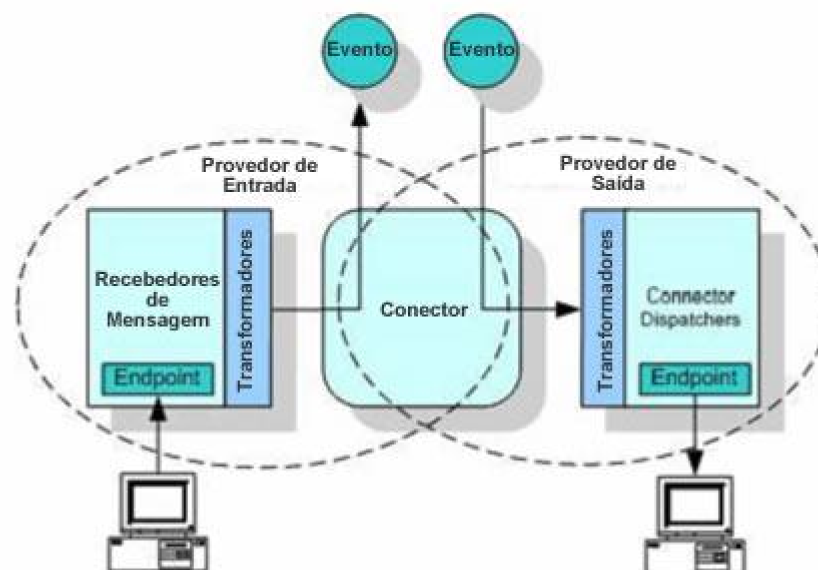


Figura 4: Diagrama com os componentes da arquitetura do Mule.
Fonte: Mule ESB, 2006.

O *Mule Model* ou *Container* é o local onde os objetos são gerenciados e executados.

O Componente UMO são os objetos de negócio no padrão *Java-Beans*. Endpoints são os que definem o canal de comunicação a ser utilizado entre um ou mais componentes. Dessa forma, os componentes UMO podem se comunicar com outras instâncias do Mule ou com aplicações externas, tais como servidores de aplicação ou sistemas legados.

O Mule também oferece uma série de objetos para configuração que correspondem aos tipos: Transporte, Conector, Roteador, Filtros e Transformadores. Abaixo segue a descrição de cada um deles.

Transporte: são os objetos que permitem a conectividade necessária pelo ESB. O Mule disponibiliza uma série de objetos para facilitar o desenvolvimento, tais como: *EjbProvider*, *EmailProvider*, *FileProvider*, *HttpProvider*, *JmsProvider* e *SoapProvider*. Outros podem ser facilmente desenvolvidos caso seja necessário. É através dos objetos de Transporte que um serviço web pode chamar um serviço que é disponibilizado através de um JMS, por exemplo.

Conector: é o mecanismo utilizado para se conectar a sistemas externos e protocolos de modo a enviar e receber dados. Uma série de conectores é disponibilizada pelo Mule, tais como: *Pop3Connector*, *SmtpConnector*, *EjbConnector* e *FileConnector*.

Objetos do tipo Roteador: controlam como eventos são enviados e recebidos pelos componentes do sistema. Mule define os roteadores de entrada para os eventos que são recebidos e os roteadores de saída para os eventos que são invocados quando um está sendo enviado.

Objetos do tipo Filtro: fornecem a lógica para que permite invocar um roteador específico. Dessa forma, você pode filtrar os tipos de mensagens de acordo com parte da informação enviada e indicar se o evento deve ou não ser invocado.

Transformadores: são objetos que realizam transformações nos dados para o tipo requerido para processamento pelo o objeto UMO. Eles recebem o dado e quando aplicados sobre o roteador de saída, garantem que o endpoint irá receber o tipo de dados requerido pela operação antes de ser enviado.

Na Figura 05, pode-se ver um exemplo de transformador de entrada:

Figura 5: Transformador de entrada.
Fonte: Mule ESB, 2006.

E um transformador de saída, na Figura 06 abaixo:

Figura 6: Transformador de saída.
Fonte: Mule ESB, 2006.

3. SERVIÇOS WEB

Neste capítulo, são abordadas as definições e características das tecnologias de Serviços Web, bem como o suporte dos Serviços Web para este trabalho.

3.1 Conceitos

A tecnologia de Serviços Web está focada em satisfazer requisitos técnicos e de negócios baseados em modelos produtos/consumidor. Eles são atividades executadas em resposta a uma solicitação (ou um evento) para entregar algum resultado.

O W3C (*World Wide Web Consortium*) e o OASIS são as instituições responsáveis pela padronização dos Serviços Web. Empresas como IBM e Microsoft, duas das maiores do setor de tecnologia, apóiam o desenvolvimento deste padrão.

Segundo a definição do W3C, um Serviço Web é uma aplicação de software identificada por um URI (Identificador de Recurso Único), cujas interfaces e ligações são capazes de ser definidas, descritas e descobertas como artefatos XML. Um Serviço Web suporta interações diretas com outros componentes de software usando mensagens baseadas em XML, trocadas via protocolos baseados na Internet (CHAMPION et al., 2002).

Assim, como componentes de software, Serviços Web representam uma funcionalidade *black box* que pode ser reutilizada sem a preocupação com a linguagem e o ambiente utilizado em seu desenvolvimento (GRAHAM, 2002; HANSEN, 2003).

Na arquitetura de Serviços Web estão definidas três entidades: o Provedor de Serviço, o Solicitador de Serviço e o Corretor de Serviços (CHAMPION et al., 2002).

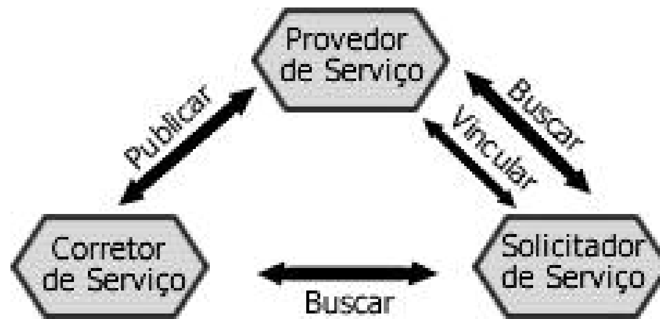


Figura 7. Interações em uma Arquitetura Orientada a Serviços Web.
Fonte: Champion et al., 2002.

Na Figura 7, o Provedor de Serviço fornece a realização de algum Serviço Web. Para tanto, deve anunciar suas funcionalidades. O Solicitador de Serviços é uma entidade que irá utilizar-se de um Serviço Web oferecido por um Provedor de Serviço, logo, ele deve descobrir o Serviço Web que melhor atenda às suas necessidades. Normalmente o Solicitador de Serviços vai buscá-lo em um diretório (UDDI, no caso específico dos Serviços Web). O Corretor de Serviços quando solicitado, desempenha o papel de oferecer os Serviços Web que satisfaçam os requisitos dos Solicitadores de Serviços (clientes).

Para obter uma melhor noção da tecnologia de Serviços Web, na próxima seção serão abordados três de seus antecessores, CORBA, COM e DCOM.

3.2 Antecessores

Anteriormente muitas soluções foram propostas, mas mostraram-se com algumas desvantagens como de altos custos, proprietárias e complexas, características que eliminavam os grandes potenciais das empresas que desejassem utilizar a Internet como meio de transação comercial (SOFTWARE AG, 2002).

Os primeiros passos para a utilização da tecnologia de Serviços Web foram o CORBA (CORBA, 2006), o COM e DCOM (DCOM, 2006). Embora muitos tenham sido desenvolvidos, problemas como alta complexidade e alto custo associados às tecnologias EAI (Integração de Aplicações Empresariais) inviabilizaram sua utilização em maiores proporções (ALONSO et al., 2003).

3.2.1 CORBA (Common Object Request Broker Architecture)

Especificação de uma arquitetura desenvolvida para ambientes distribuídos e heterogêneos, que utiliza a tecnologia de objetos distribuídos. Essa especificação foi criada pela OMG (*Object Management Group*).

A OMG é uma organização internacional da indústria de software fundada em 1989 suportada por centenas de membros. A carta de princípios da organização inclui o estabelecimento de diretrizes na indústria e especificações de gerenciamento de objetos para prover uma estrutura comum para o desenvolvimento de aplicações. O objetivo primário é alcançar sistemas baseados em objetos, em ambientes distribuídos e heterogêneos com características de reusabilidade, portabilidade e interoperabilidade.

Para compreender CORBA é necessário entender o seu papel na Arquitetura de Gerenciamento de Objetos (OMA do inglês, *Object Management Architecture*) mostrada na Figura 8. Ela é uma especificação, na verdade, um conjunto de especificações relacionadas que definem um vasto leque de serviços para a construção de aplicações distribuídas (OMG, 1997).

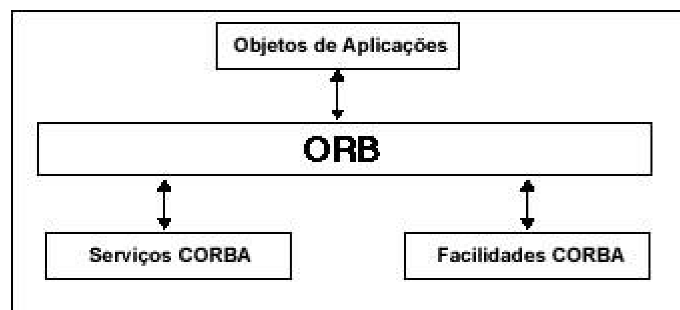


Figura 8: Object Management Architecture.
Fonte: CORBA, 1997.

A OMG também é responsável pela definição do componente ORB (*Object Request Broker*) na Figura 08, o qual tem por finalidade fornecer basicamente mecanismos pelos quais objetos fazem requisições e recebem as respectivas respostas de modo transparente em ambientes distribuídos e heterogêneos.

Na Figura 09 é descrito o mais básico dos componentes e interfaces definidos por CORBA. Esta figura é uma expansão do componente ORB da OMA descrito anteriormente na Figura 08.

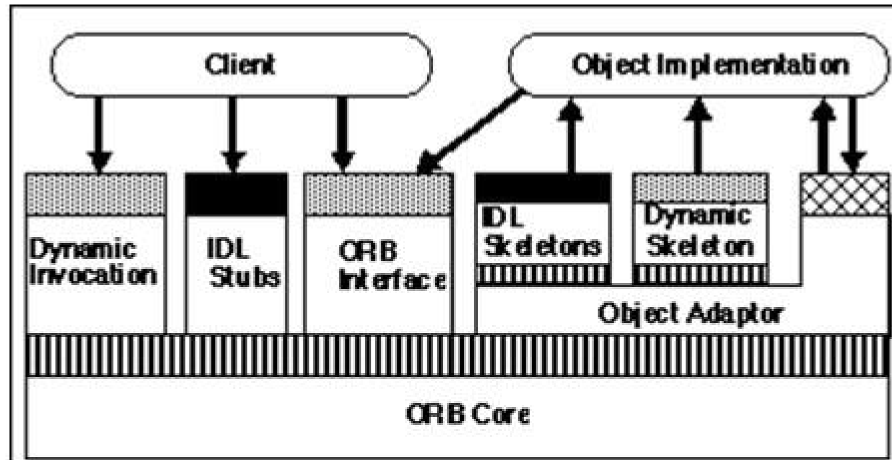


Figura 9: Estrutura da Interface CORBA.
Fonte: CORBA, 2007.

Outro elemento que é crucial para o entendimento de CORBA é a Linguagem de Definição de Interface (IDL do inglês, *Interface Definition Language*). Todos os objetos são definidos em CORBA (atualmente, na OMA) utilizando IDL.

A IDL permite que interfaces para os objetos sejam definidas independentemente da implementação dos objetos. Depois de definir uma interface na IDL, a definição da interface é utilizada como entrada para um compilador IDL que produz uma saída que pode ser compilada e linkada com uma implementação de um objeto e seus clientes..

Mapeamentos de linguagens são definidos de IDL para C, C++, Java, Ada95 e Smalltalk80.

Um ponto importante para ser enfatizado, é que CORBA especifica implementações de objetos e clientes. Podendo ser escritos em diferentes linguagens de programação, executados em diferentes arquiteturas de hardwares de computadores, diferentes sistemas operacionais e serem feitos de modo transparente. (CORBA, 1997).

3.2.2 COM e DCOM

DCOM (*Distributed Component Object Model*) é uma tecnologia proprietária da Microsoft para o desenvolvimento de componentes de software distribuídos em computadores interligados em rede. O DCOM é uma extensão do COM (*Component Object Model*) que permite a comunicação de componentes baseados em rede.

Enquanto os processos no COM podem ser executados na mesma máquina, mas em locais diferentes de endereço, a extensão DCOM permite a distribuição de processos em uma rede (DCOM, 1997). Atualmente é uma solução de plataforma única e os serviços de transações da Microsoft são construídos sobre ele.

O DCOM foca direto no meio dos componentes da aplicação. A Figura 10 mostra como tudo se encaixa em conjunto.

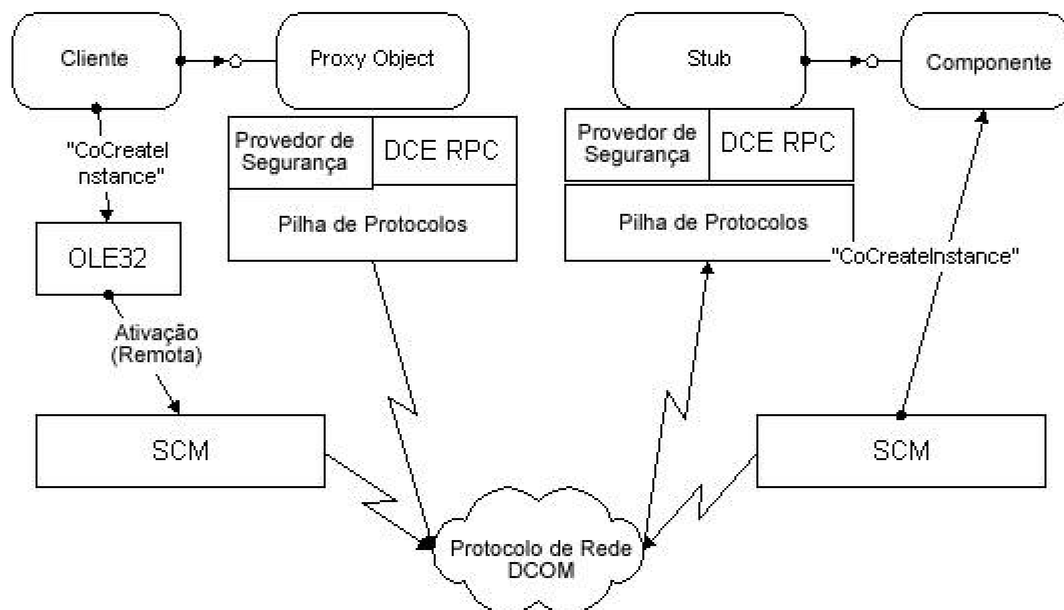


Figura 10: Arquitetura DCOM.
Fonte: Horstmann; Kirtland, 1997.

OLE (*Object Linking and Embedding*), ActiveX e MTS (*Microsoft Transaction Server*) representam serviços de aplicações de alto-nível que são desenvolvidos sobre o COM e DCOM, enquanto o COM e o DCOM representam tecnologias de baixo-nível que permite a interação de componentes (HARMON, 1999).

A tecnologia OLE é baseada no COM para prover serviços, assim como objetos conectados e embutidos são utilizados na criação de documentos compostos, documentos gerados a partir de múltiplas fontes de ferramentas.

ActiveX estende as capacidades básicas do COM para permitir que componentes sejam embutidos nos Sites Web.

MTS expande capacidades do COM através de serviços de negócio assim como transações e segurança, permitindo que Sistemas de Informação Empresariais sejam construídos utilizando componentes COM.

A tecnologia DCOM foi substituída na plataforma de desenvolvimento .NET, pela API .NET Remoting.

3.3 Tecnologias Associadas aos Serviços Web

Existem algumas tecnologias padrão para a construção de Serviços Web: HTTP *(HiperText Transfer Protocol)*, XML (*eXtensible Markup Language)*, WSDL (*Web Services Description Language*), SOAP (Simple Object Access Protocol) e UDDI (*Universal Description, Discovery and Integration*).

O mais comum é encontrar serviços implementados utilizando XML como a linguagem responsável por representar tipos de dados e compor as mensagens, SOAP como o protocolo de troca de mensagens XML, HTTP como o protocolo responsável pelo envio das mensagens, WSDL para descrever o serviço e UDDI para listar os serviços na rede.

A execução das interações entre as entidades acontece via rede, suportada pelo o conjunto das tecnologias mencionadas anteriormente. A Figura 11 demonstra este cenário em um conjunto de camadas conceituais.

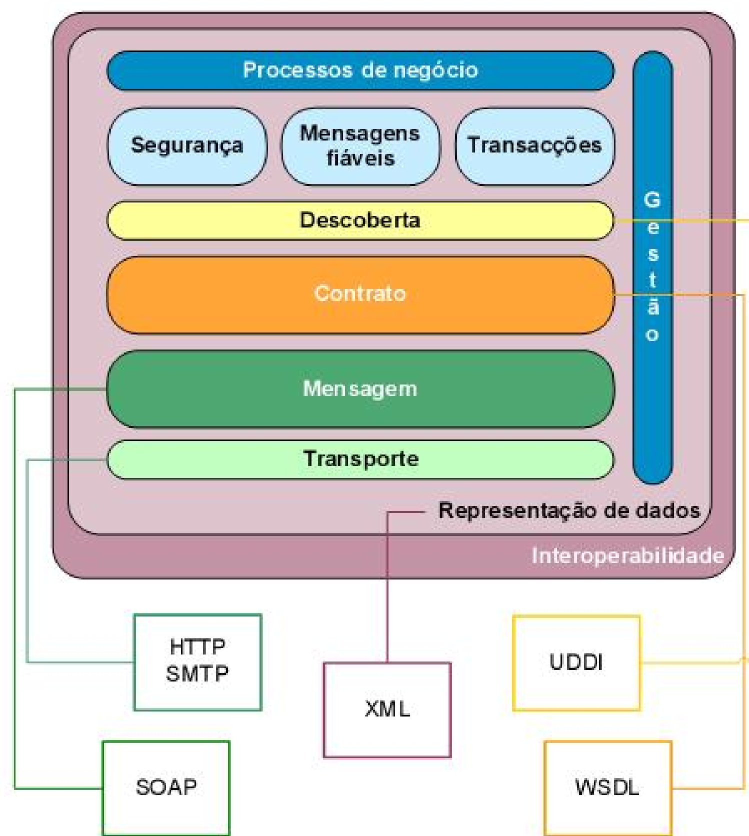


Figura 11. Camadas conceituais dos Serviços Web.
Fonte: adaptado de Hansen (2002, p. 3).

A rede é a camada base. Ela envolve os protocolos de transporte (HTTP, FTP, SMTP) e pode ser utilizada para implementação de necessidades das aplicações, tais como: disponibilidade, desempenho, segurança e confiabilidade. A mensagem baseada em XML é o alicerce da comunicação. Ela utiliza o protocolo SOAP para executar a troca de mensagens entre provedor, corretor e solicitante. A descrição de serviços é feita com uma linguagem específica: a WSDL. As camadas de publicação e descoberta de serviços utilizam UDDI para tornar Serviços Web disponíveis (HANSEN, 2002; FERRIS; FARRELL, 2003).

Para uma maior compreensão da tecnologia dos Serviços Web, nas próximas subseções serão abordadas com mais detalhes as suas tecnologias associadas.

3.3.1 XML (*eXtensible Markup Language*)

Linguagem baseada em texto que é totalmente independente da plataforma. Deste modo, esta linguagem é ideal para a troca de mensagens entre as plataformas, que é algo muito comum de ocorrer entre serviços, que, provavelmente estão em plataformas distintas.

De acordo com as especificações do XML definidas pelo W3C (XML W3C, 2006), o projeto do XML tem alguns objetivos:

- Ser diretamente utilizável na Internet;
- Oferecer suporte a uma ampla variedade de aplicações;
- Ser compatível com SGML (*Standard Generalized Markup Language*);
- Facilidade na escrita de programas que processem documentos XML;
- Número de recursos opcionais deve ser mantido ao mínimo absoluto, idealmente nulo;
- Documentos XML devem ser legíveis por seres humanos (*human-legible*) e razoavelmente claros;
- O projeto em XML deve ser formal, conciso e desenvolvido rapidamente;

XML é muito utilizada atualmente em diversas aplicações, pois, devido ao fato de ser possível criar marcadores de acordo com a necessidade atual, permite a representação de muitos tipos de dados.

A facilidade disponibilizada pela extensabilidade dos marcadores, torna-se possível criar marcadores que realmente trazem um valor semântico à mensagem e, assim, facilitam imensamente o tratamento dos dados.

Com XML, a informação é organizada de uma forma hierárquica, parecida com uma árvore, onde os marcadores mais internos são imediatamente dependentes do marcador que está no nível superior a estes.

Este tipo de estrutura, aliada aos marcadores com valor semântico, permite que os algoritmos responsáveis por analisar a mensagem sejam simples e eficientes.

3.3.2 WSDL (*Web Services Description Language*)

Linguagem para a descrição de Serviços Web que descreve a interface do serviço de forma estruturada e padronizada em XML (CHRISTENSEN et al., 2006).

WSDL descreve um serviço como uma coleção de operações que podem ser acessadas através de mensagens. Utilizando seus métodos, é possível descrever um serviço de forma transparente e independente de implementação. As duas partes envolvidas em uma interação de Serviços Web precisam ter acesso à mesma descrição WSDL para conseguirem entender uma à outra (HANSEN, 2002; NEWCOMER, 2002).

A Figura 12 demonstra que um WSDL contém os seguintes elementos:

Figura 12. Elementos do WSDL.

- Tipo (*type*) – define os tipos de dados que estão contidos em uma mensagem.
- Mensagem (*message*) – define entrada e saída de dados referentes às operações. É semelhante ao parâmetro na chamada do método;
- Tipo de porta (*portType*) – informa os elementos de operações do Serviço Web;

- Ligação (*binding*) – descreve protocolos, formato de data, segurança e outros atributos para uma interface (Tipo de Porta) em particular;
- Serviço e Porta (*port*) – Incluem a localização da implementação do serviço na rede, ou seja, contém a informação para onde enviar a solicitação do serviço.

WSDL também define protocolos de ligação e detalhes de rede. Ela apresenta descrições adicionais como contexto, QoS (*Quality of Service*) e relacionamentos entre serviços.

3.3.3 SOAP (*Simple Object Acess Protocol*)

Protocolo baseado em XML, utilizado para definir um modo uniforme de transmitir dados. É utilizado para troca de mensagens de via única e que não armazena informações sobre interações anteriores. É um paradigma bastante simples, no entanto, as aplicações podem implementar interações mais complexas, tais como requisição/resposta e requisição/múltiplas respostas.

Uma mensagem SOAP (Figura 13) é composta basicamente dos seguintes elementos:

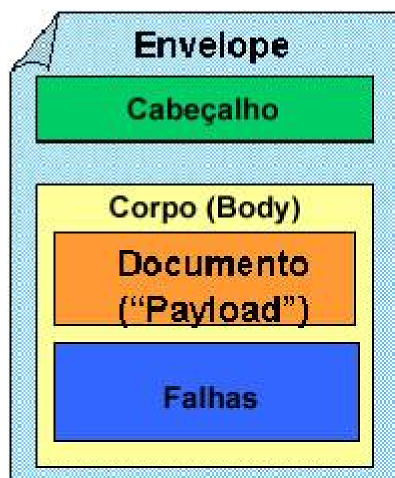


Figura 13. Estrutura de uma mensagem SOAP
Fonte: SOAP W3C, 2007.

- Envelope: Toda mensagem SOAP deve contê-lo. É o elemento raiz do documento XML. Identifica o documento XML como uma mensagem SOAP e é responsável por definir o conteúdo da mensagem;

- *Header*: É um cabeçalho opcional. Ele carrega informações adicionais, como por exemplo, se a mensagem deve ser processada por um determinado nó intermediário (é importante lembrar que, ao trafegar pela rede, a mensagem normalmente passa por diversos pontos intermediários até alcançar o destino final). Quando utilizado, o *Header* deve ser o primeiro elemento do Envelope;

- *Body*: Este elemento é obrigatório e contém as informações de chamada e de resposta ao servidor. O elemento *Body* pode conter um elemento opcional *Fault*;

- *Fault*: Utilizado para carregar mensagens de status e erros retornadas pelos "nós" ao processarem a mensagem. Este elemento só aparece nas mensagens de resposta do servidor;

SOAP é projetado para invocar aplicações remotas através de RPC (Chamadas Remotas de Procedimento) ou trocas de mensagens, em um ambiente independente de plataforma e linguagem de programação. SOAP é, portanto, um padrão normalmente aceito para ser utilizado com Serviços Web. Desta forma, pretende-se garantir a interoperabilidade e intercomunicação entre diferentes sistemas, através da utilização da linguagem XML e mecanismo de transporte padrão (HTTP).

3.3.4 HTTP (*HyperText Transfer Protocol*)

Protocolo da camada de aplicação que se tornou o protocolo padrão de transferência de dados na Internet.

Originalmente, o protocolo foi desenvolvido para o intercâmbio de páginas HTML, mas por ser um protocolo do tipo requisição-resposta, tem sido utilizado em diversas outras situações.

Uma requisição utilizando HTTP consiste em um cliente iniciando uma conexão TCP (*Transfer Control Protocol*), protocolo da camada de transporte responsável por dividir as mensagens em pacotes e enviá-las através de uma rede, com um servidor em um ponto distinto da rede.

A requisição é enviada a uma porta específica onde o servidor já monitora a entrada de requisições e, após processar a requisição, o servidor envia uma mensagem de resposta que será a informação requisitada ou alguma mensagem de erro.

Existe uma variação comum do HTTP, que é o HTTPS (*HTTP over Secure Socket Layer*). Neste caso, a camada de segurança SSL (*Secure Sockets Layer*) criptografa a mensagem antes do envio.

3.3.5 UDDI (Universal Discovery Description Integration)

Elemento central no grupo dos padrões envolvidos na tecnologia de Serviços Web. É o mediador através do qual se conhecem os possíveis clientes com os provedores dos serviços. Define um método padrão para publicar e descobrir serviços no contexto SOA. A sua implementação é semelhante a uma lista telefônica formada pelas seguintes partes:

- **Páginas Amarelas:** Contém informações organizadas por categoria específica do produto ou por regiões geográficas;

- **Páginas Brancas:** Contém informações sobre os fornecedores de serviços incluindo o endereço, o contato e os identificadores conhecidos;

- **Páginas Verdes:** Contém informações técnicas sobre as funcionalidades dos Serviços Web que são expostos pelo negócio. Por exemplo, demonstra como fazer a comunicação com eles;

O UDDI é composto por quatro especificações principais (Figura 14):

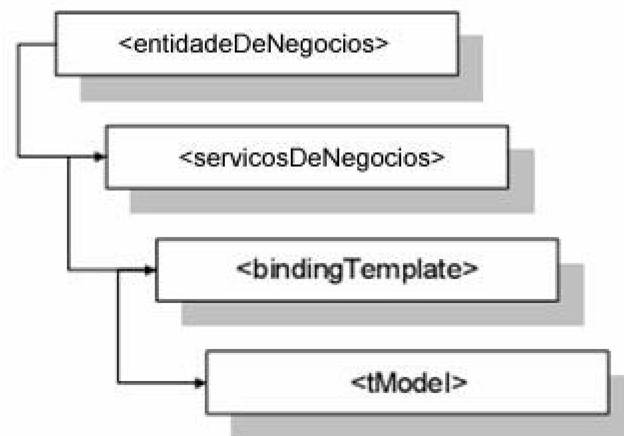


Figura 14. Estrutura do UDDI.

1. Entidade de Negócios (*Business Entities*): Contém informações sobre a empresa que publica um Serviço Web. Esta informação contém o nome da empresa e também pode incluir as informações das URLs dos serviços, os serviços oferecidos, além de informações de contato com a empresa;

2. Serviços de Negócios (*Business Services*): Contém informações sobre a descrição de cada serviço oferecido em termos de negócios;

3. *Binding Template*: Contém informações que descrevem como obter o acesso a um serviço, informando quais os pontos de acesso ao serviço chamado através de URLs;

4. *tModels*: Contém informações que descrevem uma especificação técnica do serviço. Por exemplo, os protocolos de rede ou as regras de sequência;

O propósito funcional desta estrutura UDDI é de conseguir definir a maneira de publicar e descobrir as informações sobre os Serviços Web, descrevendo as funcionalidades específicas dos serviços disponibilizados por uma empresa, através de uma conexão com a Internet para que outros softwares possam utilizá-lo.

4. SOC – SEPARAÇÃO DE INTERESSES

Neste são apresentados alguns fundamentos e conceitos da SoC (do inglês, *Separation Of Concerns*) e dos paradigmas de programação. Abordando um pouco da sua evolução, origens e fundamentos, Programação Procedural, Programação Orientada a Objetos e Programação Orientada a Aspectos.

4.1 Origens e Fundamentos

Nas primeiras formas de desenvolver software, os programadores inseriam os programas diretamente em código binário na memória principal do computador. Para tal, eram utilizados banco de chaves. A probabilidade de erros era alta, a manutenção de código praticamente impossível e o reuso, um sonho distante. Esta abordagem pode ser chamada de programação binária.

De acordo com Ossher e Tarr (1999) a Separação de Interesses é a capacidade de identificar, encapsular e manipular somente os interesses do software que são relevantes para um determinado conceito, objetivo ou efeito. Portanto, o interesse pode ser considerado a parte de um sistema de software pertinente a um determinado conceito, objetivo ou efeito.

Muitos paradigmas de programação ajudam desenvolvedores no processo da SoC. Por exemplo, na Programação Orientada a Objetos, bem como um desenho no padrão MVC (*Model View Controler*) pode separar o conteúdo da apresentação e o processamento de dados do conteúdo. Na Arquitetura Orientada a Serviços os interesses podem ser divididos em serviços, conforme visto nos capítulos 2 e 3. Linguagens de Programação Procedural, como C ou Pascal podem separar os interesses em procedimentos. Já na Programação Orientada a Aspectos podem separar os interesses em aspectos.

Nas próximas seções serão abordados os conceitos e exemplos de alguns destes paradigmas de programação de software, para um maior entendimento dos conceitos da SoC.

4.2 Programação Procedural

Paradigma de programação baseado no conceito de chamadas a procedimentos. Sendo que, procedimentos também são conhecidos como rotinas, subrotinas, métodos ou funções que simplesmente possuem um conjunto de passos computacionais a serem executados. Um dado procedimento pode ser chamado a qualquer hora durante a execução de um programa, inclusive por outros procedimentos ou por si mesmo.

René Descartes, com sua proposta para método científico tornou padrão a estratégia de dividir para conquistar, que é bastante comum para a solução de problemas complexos de qualquer natureza. E foi o surgimento de linguagens de programação capazes de representar o conceito de procedimentos que possibilitou esta estratégia. Mas a Programação Procedural apresenta algumas limitações:

- A sequência de chamadas podem se tornar demasiadamente emaranhada, o que dificulta a manutenção e a depuração do código;
- A capacidade de reuso de código é limitada pela interdependência entre procedimentos. Um procedimento, para ser utilizado, precisa estar em um contexto de execução. E para poder reutilizar é necessário recriar o contexto em questão.
- Não há possibilidade de se criar procedimentos mais específicos a partir de outros mais genéricos.
- Não há encapsulamento dos dados, todos os procedimentos são acessíveis, não existindo meios de possibilitar a restrição.
- Uma mudança na representação de dados implica na alteração em cada local onde estes são acessados.

4.3 Programação Modular

Tentativa de melhorar as deficiências provenientes da Programação Procedural. Esta programação divide os programas em vários módulos, combinando

procedimentos e dados. Estes módulos ocultam os detalhes internos do restante do programa, pois quando um programa precisa utilizar um, o faz utilizando a interface disponibilizada.

Uma das características desta programação foi a introdução do conceito de estado interno, que é a combinação dos dados (na forma de atributos) de um módulo.

Existem algumas limitações na Programação Modular:

- Capacidade de ampliar um módulo, impossibilitando alterações incrementais. Só é possível o reuso de código por meio de agregação de módulos ou colaboração.

Abaixo, pode ser observado um exemplo de reuso por meio de colaboração na Figura 15.

```
// módulo tipoPessoa
typedef struct {
        char nome[70];
        int idade;
} tipoPessoa

// módulo tipoCliente
typedef struct {
        tipoPessoa pessoa;
        int credito;
} tipoCliente
```

Figura 15. Reuso de módulo por colaboração.

- Falta de operadores de visibilidade. Todos os elementos de um módulo podem ser alterados diretamente, não existindo um meio para controlar o acesso aos recursos dos mesmos. Assim, um programador desinformado, ou mesmo, mal-intencionado, poderia fazer uso de elementos que compunham o módulo e comprometer sua utilidade.

4.4 Programação Orientada a Objetos (OOP)

Paradigma de programação de sistemas de software baseado na composição e interação entre diversas unidades de software chamadas de objetos.

Smalltalk, Perl, Python, Ruby, PHP, ColdFusion, C++, Object Pascal, Java, Javascript, ActionScript ,C# e VB.NET são exemplos de linguagens de programação com suporte a orientação a objetos, mas nem todas elas são puramente orientadas a objetos.

Dentre os seus conceitos fundamentais, segundo a Sun Microsystems, (2005) estão:

- **Classe**: representa um conjunto de objetos com características afins. Uma classe define o comportamento dos objetos, através de métodos e quais estados ele é capaz de manter através de atributos. Exemplo de classe: os *Clientes* de uma empresa.

- **Objeto**: é uma instância de uma classe. Ele é a chave para o entendimento da tecnologia orientada a objeto. Um objeto é capaz de armazenar estados, através de seus atributos e reagir as mensagens enviadas a ele, assim como se relacionar e enviar mensagens a outros objetos. Exemplo de objetos da classe *Clientes*: João, José, Maria.

- **Atributos:** são características de um objeto. Basicamente a estrutura de dados que vai representar a classe. Exemplos: *Cliente*: nome, endereço, telefone, CPF; *Carro*: nome, marca, ano, cor; *Livro*: autor, editora, ano. Por sua vez, os atributos possuem valores. Por exemplo, o atributo “ano” pode conter o valor 1984.

- **Métodos:** definem as habilidades dos objetos. *Palio* é uma instância da classe *Carro*, portanto tem habilidade para acelerar, implementada através do método *acelerar()*. Um método em uma classe é apenas uma definição. A ação só ocorre quando o método é invocado através do objeto, no caso *Palio*. Dentro do programa, a utilização de um método deve afetar apenas um objeto em particular; Todos os carros

podem acelerar, mas você quer que apenas *Palio* acelere. Normalmente, uma classe contém diversos métodos, que no caso da classe *Carro* poderiam ser *frear()*, *virar()* e *passarMarcha()*.

- **Mensagem**: é uma chamada a um objeto para invocar um de seus métodos, ativando um comportamento descrito por sua classe. Também pode ser direcionada diretamente a uma classe, através de uma invocação a um método estático, conforme pode ser observado na Figura 16 abaixo.

Figura 16. Troca de mensagem.

- **Sobrecarga:** é a utilização do mesmo nome para símbolos, métodos com operações ou funcionalidades distintas. Geralmente diferenciam-se os métodos pela sua assinatura.

- **Herança**: (ou generalização) é o mecanismo pelo qual uma classe (sub-classe) pode estender outra classe (super-classe), aproveitando seus comportamentos (métodos) e estados possíveis (atributos). Há Herança múltipla quando uma sub-classe possui mais de uma super-classe. Essa relação é normalmente chamada de relação "é um". Um exemplo de herança: *Mamífero* é super-classe de *Humano*. Ou seja, um Humano é um mamífero.

- **Associação:** é o mecanismo pelo qual um objeto utiliza os recursos de outro. Pode tratar-se de uma associação simples "usa um" ou de um acoplamento "parte de". Por exemplo: Um Humano usa um Telefone. A tecla "1" é parte de um telefone.

- **Encapsulamento:** consiste na separação de aspectos internos e externos de um objeto. Este mecanismo é utilizado amplamente para impedir o acesso direto ao estado de um objeto (seus atributos), disponibilizando externamente apenas os métodos que alteram estes estados. Exemplo: você não precisa conhecer os detalhes de um motor do carro para dirigi-lo. A lataria, volante, marcha, acelerador, freio encapsula esses detalhes, provendo a você uma interface mais amigável.

- **Abstração:** é a habilidade de concentrar nos aspectos essenciais de um contexto qualquer, ignorando características menos importantes ou acidentais. Em modelagem orientada a objetos, uma classe é uma abstração de entidades existentes no domínio do sistema de software.

- **Polimorfismo:** é o princípio pelos quais duas ou mais classes derivadas de uma mesma superclasse, podem invocar métodos que têm a mesma assinatura (lista de parâmetros e retorno), mas comportamentos distintos, especializados para cada classe derivada, usando para tanto uma referência a um objeto do tipo da superclasse. A decisão sobre qual o método que deve ser selecionado, de acordo com o tipo da classe derivada, é tomada em tempo de execução através do mecanismo de ligação tardia. No caso de polimorfismo, é necessário que os métodos tenham exatamente a mesma identificação, sendo utilizado o mecanismo de redefinição de métodos. Esse mecanismo de redefinição não deve ser confundido com o mecanismo de sobrecarga de métodos.

- **Interface:** é um contrato entre a classe e o mundo externo. Quando uma classe implementa uma interface, ela está comprometida a fornecer o comportamento publicado pela interface.

Entretanto, este paradigma tem demonstrado algumas limitações (OSSHER; TARR, 1999), como o entrelaçamento (*code tangling*) e o espalhamento de código. Estes problemas tornam a implementação mais complexa, dificultando entendimento e a manutenção das aplicações.

O espalhamento de código é a necessidade de atualização em vários trechos de código, em caso de manutenção de um sistema. Isto caracteriza um problema de redundância de código.

Já o entrelaçamento se refere aos trechos de código com diferentes funcionalidades em um mesmo módulo, o que pode complicar da mesma forma a manutenção de um sistema.

Na próxima sub-seção 4.5 sobre a Programação Orientada a Aspectos será apresentado um exemplo relacionado ao espalhamento e entrelaçamento de código.

4.4.1 Exemplo de uma Classe Orientada a Objetos

Na Figura 17 mostrada abaixo, pode-se visualizar um código simples escrito em JAVA de um elemento do mundo real, *Cliente*, para um melhor entendimento do que foi abordado até aqui sobre Orientação a Objetos.

```
Class Cliente {

private int idade = 0;
private String nome =  “” ;
private int credito = 0;

        public int getNome( ) {
                return nome;
        }

        public int setNome( String dNome ) {
                nome = dNome;
        }
        public int getCredito( ) {
                return credito;
        }

        public int setCredito( int dCredito) {
                credito = credito + dCredito;
        }
```

Figura 17. Exemplo de uma classe em JAVA.
Fonte: Elaboração própria, 2007.

Como visto na Figura 17, a classe Cliente contém três atributos que são: idade, nome, credito.

E a classe Cliente possui também quatro comportamentos (métodos) que são:

- getNome: retorna o nome do cliente;
- setNome: modifica o nome do cliente;
- getCredito: retorna o crédito do cliente;
- setCredito: modifica o crédito do cliente;

4.5 Programação Orientada a Aspectos (AOP)

A AOP (KICZALES et al., 1997) reduz os problemas apresentados na Programação Orientada a Objetos, aumentando a modularidade, através da separação do código que implementa funções diretamente ligadas ao sistema, que são os interesses tranversais ou não-funcionais. Com esta separação, o código gerado se torna mais compreensível e melhor para dar manutenção.

4.5.1 Fundamentos

A AOP é paradigma de programação que visa fornecer uma melhor separação dos interesses do sistema.

A AOP estende outras técnicas, como a POO ou programação estruturada, propondo não apenas uma decomposição funcional, mas também sistêmica do problema. Isso permite que a implementação de um sistema seja separada em requisitos funcionais e não-funcionais, disponibilizando a abstração de aspectos para a decomposição de interesses sistêmicos, além dos recursos já oferecidos pelas linguagens de componentes, como Java, por exemplo.

Segundo Kiczales et al. (1997), o seu principal objetivo é separar o código referente ao negócio do sistema dos interesses transversais, de uma forma bem definida e centralizada (Figura 18).

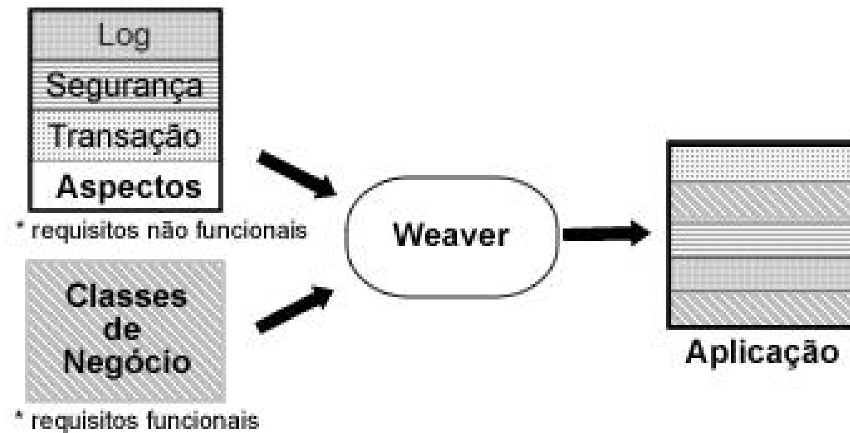


Figura 18. Separação de Interesses na AOP.
Fonte: Adaptada de Kiczales et al., 1997.

A função do *weaver* (combinador de aspectos) é identificar nos componentes, pontos de junção onde os aspectos se aplicam, produzindo o código final da aplicação, que implementa tanto as propriedades definidas pelos componentes como aquelas definidas pelos aspectos (CHAVES, 2004). Combinadores podem atuar em tempo de compilação ou de execução.

Conforme visto anteriormente, os paradigmas de programação mais antigos como a programação procedural e programação orientada a objeto, implementam os interesses sistêmicos ou transversais como a sincronização, log de dados, segurança, performance, transação, entre outros, juntamente com as regras de negócio do sistema, deixando-os espalhados pela a aplicação, ocasionando difícil manutenção e compreensão.

Na figura 19 é apresentada a implementação de uma classe Conta (referente à conta corrente em uma aplicação bancária), onde o código referente ao log de dados das operações realizadas sobre a conta corrente é misturado ao código referente a decomposição dominante da classe (código do negócio).

```
package exemplo.mestrado;

public class Conta {

        private String numero = "2929-8";
        protected double saldoAtual = 300.0;

        public void getSaldo(double novoSaldo) {
          System.out.print("Seu saldo atual é : " + novoSaldo + "\n");
        }

        public void sacar(double valor) {
         if(this.saldoAtual >= valor && valor > 0) {
            Log.registrar("Conta", numero, "sacar");
            double novoSaldo = this.saldoAtual - valor;
            Log.registrar("Conta", numero, "sacar");
            System.out.print("Você sacou " + valor + " reais de " + this.saldoAtual + "\n");
            this.getSaldo(novoSaldo);
         }
        }

        public void depositar(double valor) {
           if(valor > 0) {
              Log.registrar("Conta", numero, "depositar");
              double novoSaldo = this.saldoAtual + valor;
              Log.registrar("Conta", numero, "depositar");
              System.out.print("Você depositou " + valor + " reais \n");
              this.getSaldo(novoSaldo);
           }
        }

      public static void main(String args[]){
              new Conta().sacar(200.0);
              new Conta().depositar(1000.0);
      }
}
```

Figura 19. Paradigmas de programação mais antigos com códigos de interesses transversais espalhados nas regras de negócios.
Fonte: Elaboração própria, 2007.

Várias abordagens atualmente vêm sendo propostas para implementar aspectos, dentre elas pode-se destacar HyperJ, Programação Orientada a Assuntos (SOP), AspectC e AspectJ. Neste trabalho, o enfoque será para AspectJ, já que o objetivo é o estudo da modelagem orientada a aspectos que é baseada em AspectJ. A próxima seção apresenta os principais conceitos desta abordagem.

4.5.2 AspectJ

Uma linguagem bem aceita, por estender a linguagem de programação JAVA com construções eficientes para a implementação de interesses transversais em separado, em um sistema de software e por ser de uso geral, por meio dela, pode-se definir a implementação de aspectos em vários pontos de um programa.

Esta é uma proposta do *Palo Alto Research Center*, tradicional centro de pesquisas da Xerox.

Em AspectJ, JAVA é a linguagem de componentes utilizada. A linguagem de aspectos é genérica, possibilitando ao desenvolvedor especificar instruções nos seguintes pontos de combinação:

- Execução de métodos;
- Recebimento de chamadas a construtores;
- Execução de construtores;
- Acesso a campos;
- Execução de manipuladores de exceção.

O AspectJ oferece um conjunto de ferramentas para auxiliar na criação de aspectos, desde um compilador de aspectos, visualizadores de aspectos, plug-ins para integração com as mais populares IDE´s de desenvolvimento.

AspectJ utiliza extensões da linguagem de programação Java para construir novas regras para interesses dinâmicos e estáticos. O *AspectJ* utiliza construções próprias para os blocos de aspecto: a implementação dos aspectos é construída em blocos que formam os módulos que expressam os interesses tranversais.

Os elementos básicos em AspectJ são: pontos de junção, pontos de atuação, *advices*, inserções e aspectos.

Aspectos são os elementos principais. Podem alterar a estrutura estática ou dinâmica de um programa.

A estrutura estática é alterada adicionando, por meio das declarações intertipos, membros (atributos, métodos ou construtores) a uma classe, modificando assim a hierarquia do sistema. Já a alteração em uma estrutura dinâmica de um programa ocorre em tempo de execução por meio de pontos de junção, os quais são selecionados por pontos de atuação e através da adição de comportamentos (*advices*) antes, durante ou depois dos pontos de junção (KISELEV, 2002).

As seções seguintes conceituam cada um dos elementos da AspectJ aqui mencionados.

4.5.2.1 **Aspecto** (Aspect)

Tipicamente uma implementação AOP procura encapsular as chamadas aos interesses transversais da aplicação através de uma nova construção, chamada de Aspecto.

Um Aspecto é constituído por conceitos fundamentais em AOP, como os Pontos de Junção, Pontos de Atuação e *Advices*, conforme abordados anteriormente. Através destes, o comportamento do programa pode ser alterado.

Aspectos possuem semelhanças e diferenças com as classes (POO). Como semelhanças: podem ser estendidos, abstratos ou concretos e possuem um tipo, atributos e métodos. Ainda, como diferenças: não possuem um construtor, destrutor e podem conter pontos de atuação e *advices* como atributos.

O Aspecto é a unidade central da AspectJ da mesma forma que a classe é a unidade central de JAVA. Nele estão implementadas as regras de construção de interesses tranversais dinâmico e estático. Ponto de atuação, *advice*, inserção e declarações intertipos são combinados em um Aspecto. Como qualquer classe JAVA, os Aspectos podem conter atributos, métodos e classes internas.

Antes de começar a implementar o comportamento dos interesses tranversais, precisa ser feita uma análise para identificar os pontos de junção e quais argumentos de comportamentos se desejam modificar, para que em seguida seja definido o novo comportamento. Para iniciar uma implementação se

escreve um Aspecto que serve como um modelo que tem uma implementação geral. Então, dentro do Aspecto, escrevem-se os pontos de atuação para capturar os pontos de junção desejados. Finalmente, se escreve os *advices* para cada ponto de atuação e codifica-se a ação que se deseja executar quando o ponto de junção for chamado.

4.5.2.2 Pontos de Junção *(Join Points)*

Pontos na execução do programa. O conceito de Ponto de Junção é fundamental para o entendimento de AspectJ.

O Ponto de Junção é qualquer ponto de execução identificado dentro de um sistema. A AspectJ pode operar sobre os seguintes tipos de ponto de junção: a) chamada de métodos, b) execução de métodos, c) chamada de construtores, d) execução de inicialização, e) execução de construtores, f) execução de inicialização estática, g) pré-inicialização de objetos, h) inicialização de objetos, i) referência a campos, j) tratamento de exceções (GRADECKI; LESIECKI, 2003).

4.5.2.3 Pontos de Atuação *(Pointcuts)*

Conjunto de Pontos de Junção. Ele define onde o aspecto vai atuar.

```
pointcut set( )  : execution (* *.set(..) ) && thispoint(Point);
```

Figura 20. Exemplo de um Ponto de Atuação.

O Ponto de Atuação apresentado acima (Figura 20), especifica todos os Pontos de Junção de qualquer método que comece com "*set*" e o objeto focado seja do tipo *Point*.

Em AspectJ, um aspecto normalmente define *pontos de atuação*, que são formados pela composição de pontos de combinação, através de combinadores lógicos. Para definir um ponto de atuação utiliza-se construtores de AspectJ chamados designadores de pontos de atuação (*pointcuts designators*).

Um designador de ponto de atuação identifica o ponto de atuação pelo nome ou por uma expressão. Os termos ponto de atuação e designador de ponto de atuação são usados frequentemente como sinônimos. Pode ser declarado um ponto de atuação dentro de um aspecto, classe ou interface. Da mesma forma que atributos e métodos de classes, podem ser especificados um qualificador de acesso aos pontos de atuação (*public*, *private*, *protected* ou *default*) para restringir o acesso.

O corpo do ponto de atuação possui assinaturas e uma construção que pode definir um conjunto de tipos, denominados como padrão de tipos (*Type Pattern*). Estes são manipulados por caracteres especiais (conforme ilustrado na Tabela 2, abaixo), denominados curingas (*Wildcards*). Os curingas possuem a capacidade de incluir mais um ponto de junção no ponto de atuação.

Tabela 2. Tipo de caracteres especiais (curingas).

Caractere	Significado
*	Qualquer sequência de caracteres não contendo pontos.
(..)	Qualquer seqüência de caracteres, inclusive contendo pontos.
+	Qualquer subclasse de uma classe.

Ainda na assinatura de um ponto de atuação é preciso identificar o momento em que o método será capturado. Para isso, são chamados de designadores de ponto de atuação (SOARES; BORBA, 2005). Esses designadores podem ser identificados conforme é mostrado abaixo no Tabela 03.

Tabela 3. Tipo de designadores do ponto de atuação.

call (Assinatura)	Corresponde à chamada para um método ou para um construtor.
execution (Assinatura)	Corresponde à execução de um método ou de um construtor.
get (Assinatura)	Corresponde à referência para um atributo de uma classe.
set (Assinatura)	Corresponde à definição de um atributo de uma classe.
this (PadrãoTipo)	Retorna o objeto associado com o ponto de junção em particular ou limita o escopo de um ponto de junção utilizando um tipo de classe.

target (PadrãoTipo)	Retorna o objeto alvo de um ponto de junção ou limita o escopo do mesmo.
args (PadrãoTipo, ...)	Expõe os argumentos para o ponto de junção ou limita o escopo de um ponto de atuação.
within (PadrãoTipo)	Corresponde aos pontos de junção contidos em um tipo específico.

Em AspectJ, um ponto de atuação pode ter um nome ou ser anônimo. Pontos de Atuação anônimos, como classes anônimas, são definidas no lugar onde serão utilizadas, tais como parte de uma *advice* ou no momento da definição de outro ponto de atuação. Pontos de Atuação nomeados são elementos que podem ser referenciados de múltiplos lugares, aumentando a reusabilidade.

4.5.2.4 Advices

São métodos em que sua chamada depende dos Pontos de Junção e consequentemente em tempo de execução dos Pontos de Atuação. Um *Advice* é apenas invocado quando o Ponto de Atuação que está a ele ligado é atingido. Ele define como o aspecto irá fazer. Pode-se utilizar *Advices* de vários modos como: *after, after returning, after throwing, before e around*.

```
after( ) : set ( ) {
  Display.update ( );
}
```

Figura 21. Exemplo de *Advice*.

O código dentro deste *Advice* (Figura 23) é executado imediatamente depois do Ponto de Atuação *set*().

Inserem um novo comportamento em todos os pontos de junção, representados pelo ponto de atuação, dividindo-se em vários tipos de designadores que correspondem ao momento da ação do referente ponto de atuação, conforme a Tabela 04.

Tabela 4. Tipo de designadores.

before	Executa quando o ponto de junção é alcançado, mas imediatamente antes da sua computação
after returning	Executa após a computação com sucesso do ponto de junção
after throwing	Executa após a computação sem sucesso do ponto de junção
after	Executa após a computação do ponto de junção, em qualquer situação
around	Executa quando o ponto de junção é alcançado e tem total controle sobre a sua computação

O *advice* pode modificar a execução do código no ponto de junção, pode substituir ou passar por ele. Usando o *advice* pode-se “logar” nas mensagens antes de executar o código de determinados pontos de junção que estão espalhados em diferentes módulos. O corpo de um *advice* é semelhante ao de qualquer método, encapsulando a lógica a ser executada quando um ponto de junção é alcançado (GRADECKI; LESIECKI, 2003).

4.5.2.5 Inserção (*Introduction*)

Interesse transversal estático que introduz alterações nas classes, interfaces e aspectos do sistema. Alterações estáticas em módulos não tem efeito direto no comportamento. Por exemplo, pode ser adicionado um método ou um atributo na classe.

A *AspectJ* provê uma maneira de alterar a estrutura estática de uma aplicação, isto ocorre por meio das declarações intertipos que são descritas como interesses estáticos (*static crosscutting*). Estas declarações provêm uma construção chamada inserção.

As inserções modificam uma classe estruturalmente, acrescentando a ela novos membros, como construtores, métodos e campos por meio da cláusula *declare parents*. Analogicamente aos *advices*, que atuam em pontos específicos do programa denotados por pontos de atuação, *inserções* irão atuar sobre um conjunto de definições de classes (STEIN, 2002).

Para representar as inserções na programação orientada a aspectos, duas características devem ser analisadas (STEIN, 2002):

- Inserções são empregadas para representar a adição de novas características dentro dos elementos modelos;
- Aos elementos modelos, as Inserções podem atribuir novos relacionamentos.

4.6 Exemplo Orientado a Aspectos

Nesta seção é demonstrada a implementação da classe Conta (referente à conta corrente em uma aplicação bancária) na Figura 22 e de um Aspecto chamado de AspectoLog na Figura 23, no qual, diferentemente do exemplo visto na Figura 19 da seção 4.5.1, o código referente ao log de dados agora está armazenado em um Aspecto, que são combinados com a classe Conta. Separando assim o código de interesses tranversais dos referentes à decomposição dominante da classe (código do negócio).

```
package exemplo.mestrado;

public class Conta {

        private String numero = "2929-8";
        protected double saldoAtual = 300.0;

        public void getSaldo(double novoSaldo) {
          System.out.print("Seu saldo atual é : " + novoSaldo + "\n");
        }

        public void sacar(double valor) {
         if(this.saldoAtual >= valor && valor > 0) {
            double novoSaldo = this.saldoAtual - valor;
            System.out.print("Você sacou " + valor + " reais de " + this.saldoAtual + "\n");
            this.getSaldo(novoSaldo);
         }
        }

        public void depositar(double valor) {
            if(valor > 0) {
               double novoSaldo = this.saldoAtual + valor;
               System.out.print("Você depositou " + valor + " reais \n");
               this.getSaldo(novoSaldo);
            }
        }

      public static void main(String args[]){
               new Conta().sacar(200.0);
               new Conta().depositar(1000.0);
      }
}
```

Figura 22. Classe Conta - código sem interesses transversais misturados.
Fonte: Elaboração própria, 2007.

Como ilustrado na Figura 22, a classe Conta contém dois estados (atributos) que são: numero, saldoAtual e três comportamentos (métodos) que são:

- getSaldo(double novoSaldo): retorna o saldo do cliente;
- sacar(double valor): retira um valor da conta do cliente;
- depositar(double valor): deposita um valor na conta do cliente;

```
package exemplo.mestrado;

public aspect AspectoLog {

        pointcut logDados(): execution(* Conta.depositar(..)) || execution(* Conta.sacar(..));

        before(): logDados() {
                System.out.println("*** LOG (Antes) ");
        }

        after() returning(): logDados() {
                System.out.println("*** LOG (Depois)  \n");
        }
}
```

Figura 23. AspectoLog.aj - código referente ao Aspecto que influi na classe Conta.
Fonte: Elaboração própria, 2007.

Na Figura 23, o aspecto AspectoLog contém um ponto de atuação *LogDados()* com execução para os métodos *depositar()* e *sacar()* da classe Conta com um *advice* exibindo uma mensagem "***LOG" para antes da execução do método *before()* e outro *after() returning()* para depois do retorno do método.

O resultado deste exemplo que foi desenvolvido na IDE Eclipse versão 3.3 com o plugin do ADJT (*AspectJ Development Tools*) pode ser visto logo abaixo, na Figura 24.

```
*** LOG (Antes)
Você sacou 200.0 reais de 300.0
Seu saldo atual é : 100.0
*** LOG (Depois)

*** LOG (Antes)
Você depositou 1000.0 reais
Seu saldo atual é : 1300.0
*** LOG (Depois)
```

Figura 24. Resultado da Implementação entre a Classe Conta e o Aspecto AspectoLog.
Fonte: Elaboração própria, 2007.

Com o exemplo anterior, demonstra-se melhor o que a Programação Orientada a Aspectos complementa na Programação Orientada a Objetos, criando assim aplicações mais legíveis, mais fáceis de manutenção e modularizadas, já que os interesses transversais da aplicação estão separados dos interesses funcionais.

5. INTERCEPTADORAOP

Este capítulo apresenta o conceito dos InterceptadoresAOP, trabalhos relacionados, modelo proposto no Processamento de Mensagens do Mule, ferramentas utilizadas, configuração do ambiente, validação do modelo com dois cenários, suas arquiteturas, implementações e execuções.

5.1 Introdução

A natureza distribuída e fracamente acoplada da tecnologia de Serviços Web, indica uma série de preocupações que não estão diretamente relacionadas às regras de negócio das aplicações como o log de dados, cache, debug, autenticação, transação, desempenho, monitoramento, entre outras. Mas sim, devido às limitações dos ambientes de desenvolvimento para Serviços Web e SOA atualmente disponíveis, que costumam ser tratadas apenas de forma implícita, como parte dos requisitos funcionais das próprias aplicações.

Um exemplo são os middlewares ESB atuais, que tratam interesses transversais em cada método em tempo de projeto e não em tempo de execução. Não existindo uma forma de aplicá-los para uma enumeração, como se faz com triggers em banco de dados. Ex: “faça um log na tabela Y sempre que um insert acontecer na tabela Z” (JEFFREY GRAY, 2002). Sendo que este é o objetivo do modelo proposto neste trabalho, chamado de InterceptadorAOP.

Os InterceptadoresAOP são interceptadores que tratam os interesses transversais nos Serviços Web em middlewares ESB. Possuem suporte a AOP, baseando-se na semântica da AspectJ.

Com Aspectos, a forma mais básica de definir um ponto de atuação é enumerando os pontos de junção que devem compô-lo. Esta enumeração pode ser um a um ou através de alguma convenção de nomenclatura. Os pontos de atuação abaixo na Tabela 05, ilustram essa forma:

Tabela 5. Pontos de atuação na forma de enumerações.

Pointcut AtividadeBD ( ) :	pointcut AtividadeBD ( ) :

call (void Pessoa.updateNome (String)) || call (void Pessoa.updateIdade (int)) || call (void Pessoa.addPessoa (Pessoa)) || call (void Pessoa.deletePessoa (Pessoa));	call (void *.update* (..)) || call (void *.insert* (..)) || call (void *.delete* (..));

Na próxima seção são apresentados os trabalhos relacionados, com suas características e diferenças.

5.2 Trabalhos Relacionados

Existem alguns trabalhos relacionados sobre a inserção de interesses transversais nos middlewares ESB atuais, sendo desenvolvidos em alguns centros de pesquisa.

Dentre os trabalhos existentes, podem ser citados os trabalhos de Maffort e Valente (2006), Courbis e Finkelstein (2005), Verheecke, Vanderperren e Jonckers (2006). Um dos que mais se aproxima deste trabalho é o de *Bonér e Vasseur (2004)* em desenvolvimento na BEA Systems. O trabalho destes autores possui vários pontos em comum com este, podendo destacar particularmente dois pontos:

i) No uso de Aspectos para tratar os interesses transversais em aplicações distribuídas;

ii) O fato de empregar a noção da AOP em middlewares ESB;

Entretanto, alguns pontos são fundamentais na diferenciação deste em relação ao trabalho em desenvolvimento na BEA Systems:

i) Optou-se em utilizar um middleware ESB Open Source, o Mule, enquanto que no trabalho de Bonér e Vasseur, desenvolvido na empresa BEA Systems, optaram pelo middleware BEA WebLogic. A vantagem da escolha do middleware Mule para este trabalho, é pelo o fato dele ser Open Source, com baixo custo para os desenvolvedores, não ficando preso a fornecedores e mais leve para rodar em

computadores de baixo porte. Sendo que para rodar alguma aplicação no BEA WebLogic, por exemplo, um computador necessita ter a capacidade de quase 4 vezes mais memória. No Mule ESB qualquer máquina com 256 megabytes (Mb) de memória, roda aplicações satisfatoriamente. Foram realizados testes no BEA WebLogic em uma máquina com o processador Intel Pentium 4, 1 gigabyte (Gb) em uma aplicação, no qual rodou com dificuldades e em outra com 512 megabytes e o mesmo processador, que nem chegou a finalizar a compilação da mesma aplicação, originarando um erro.

ii) Neste trabalho é utilizada a linguagem AspectJ que é uma extensão orientada a aspectos da linguagem Java e que possui todas as abtrações da AOP. Enquanto que no trabalho de Bonér e Vasseur utilizaram o framework AspectWerkz que possui características semelhantes. A razão da opção neste trabalho pela AspectJ, é por ela estender todas as funções da AOP. Sendo que os frameworks atuais AOP, como por exemplo, o *AspectWerkz*, não possui suporte total a todas as funcionalidades da AOP.

iii) Outra vantagem de utilizar um middleware ESB de código aberto, é que esta abordagem afeta diretamente a fase de Processamento de Mensagens do Mule ESB, gerando uma nova funcionalidade dentro deste, ao contrário do BEA WebLogic, que por ser um software proprietário, não permite acesso ao código fonte.

iv) Por fim, os InterceptadoresAOP permanecem a disposição dos desenvolvedores de serem modificados, criados outros algoritmos de log de dados, tratamento de exceção ou medidor de tempo.

Outro importante trabalho relacionado é a abordagem que se encontra dentro do próprio Mule ESB, chamados de Interceptadores (*Interceptors).*

Sendo que estes não foram apresentados na seção do Mule, em outro capítulo, em razão de uma melhor diferenciação entre os Interceptadores (*Interceptors*) e InterceptadoresAOP no proposto neste trabalho.

Na Figura 25 abaixo, pode-se visualizar o Processamento de Mensagens do Mule com o mecanismo dos Interceptadores do Mule.

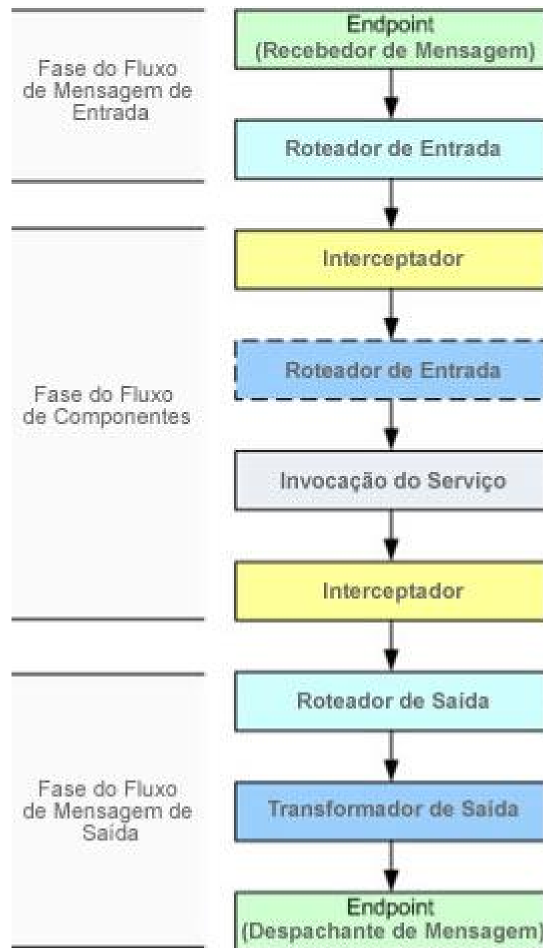


Figura 25. Processamento de Mensagens do Mule ESB sem os InterceptadoresAOP.
Fonte: Elaboração própria, 2007.

Como visto na Figura 25 acima, os Interceptadores do Mule ESB estão inseridos manualmente, sem nenhuma semântica ou meio de enumeração. E só suportam pré e pós invocações nos Componentes de Serviços. Não possuindo um modo de tratar interesses transversais durante a invocação.

Outro ponto é quanto à utilização destes em aplicações mais complexas, podendo o código se tornar um emaranhado de chamadas aos interesses transversais.

5.3 Modelo Proposto

Um ponto importante para ser enfatizado antes de prosseguir, é que o Mule ESB trata os Serviços Web como componentes UMO, que são objetos JAVA. Então os Serviços Web serão tratados nesta, como Componentes de Serviços.

Esta seção apresenta os InterceptadoresAOP no Processamento de Mensagens do Mule, mais especificamente na subfase de Fluxo dos Componentes.

Os InterceptadoresAOP são interceptadores que suportam AOP, baseando-se na semântica do AspecJ, responsáveis pelo tratamento dos interesses transversais como o log de dados, debug, tratamento de erros e medidor de tempo já pré-definidos no Mule ESB.

O objetivo principal dos InterceptadoresAop são de influir com interesses transversais sobre os Serviços Web de uma forma não intrusiva, reduzir o espalhamento de código e, consequentemente, permitir uma melhor manutenção e compreensão dos mesmos.

Está ilustrado na Figura 26, como os InterceptadoresAOP, que agora substituem os antigos *Interceptors* (sem noção da AOP), atuam nas subfases dos diferentes elementos envolvidos durante o Processamento de Mensagens do Mule ESB.

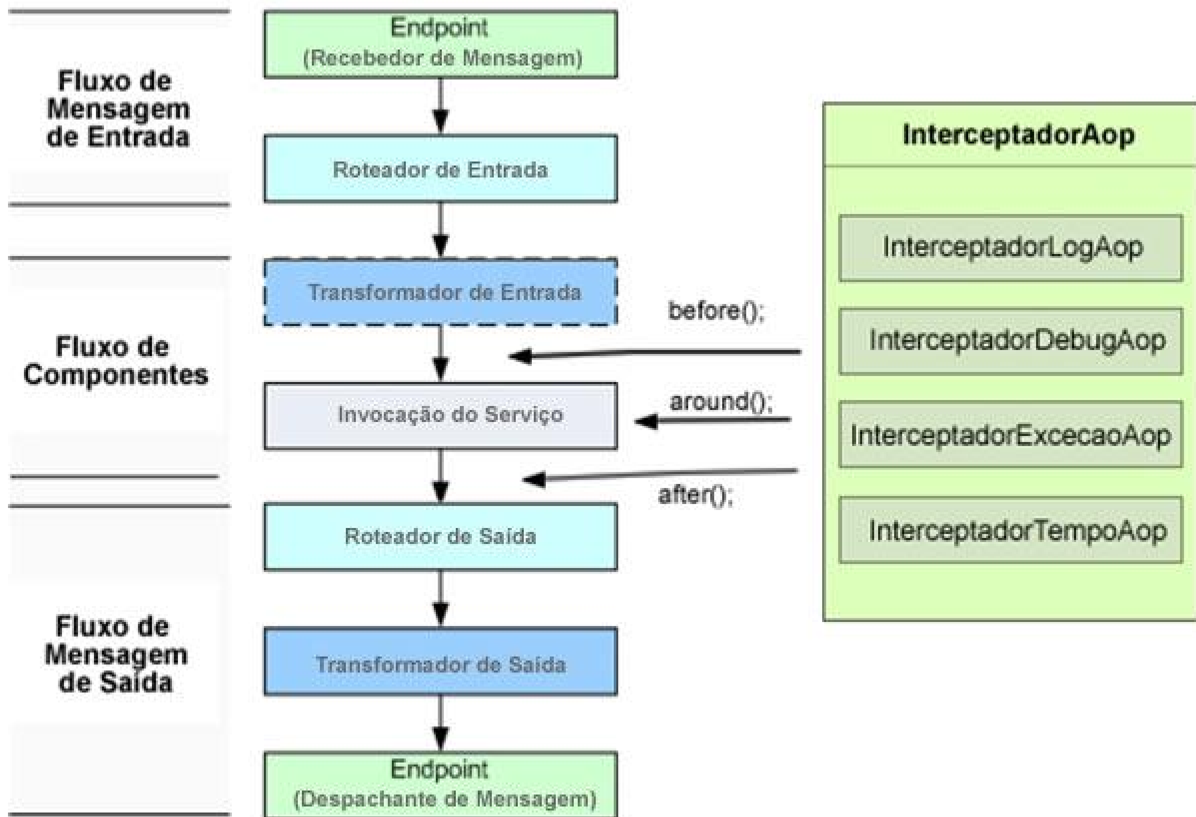


Figura 26. Processamento de Mensagens do Mule ESB com os Interceptadores AOP.
Fonte: Elaboração própria, 2007.

Mais especificamente, os InterceptadoresAOP no Processamento de Mensagens do Mule, tem como principal função, interceptar semanticamente os fluxos de mensagens dentro dos Componentes de Serviços. Estes podem ser utilizados influindo com comportamentos tranversais.

Para a realização do funcionamento dos InterceptadoresAOP foi utilizada a semântica da linguagem AspectJ, possuindo o *weaving* dos aspectos com os interesses de negócios. Sendo suportados três tipos de advices baseados na AspectJ, o *before*, *after* e *around* já apresentados anteriormente no capítulo 4, na seção da Programação Orientada a Aspectos. Com isto, não existe a necessidade de conhecer ou alterar a estrutura interna dos Componentes de Serviços manualmente.

Os InterceptadoresAOP são descritos como Aspectos e constituídos por conceitos fundamentais da AOP, com pontos de junção, pontos de atuação e

advices. Sendo que, através destes, o comportamento dos Componentes de Serviços podem ser alterados.

A descrição feita nesta seção oferece uma visão muito superficial do real funcionamento dos InterceptadoresAOP. E com o objetivo de detalhar melhor suas funcionalidades, é descrito na seção abaixo o detalhamento do modelo dos InterceptadoresAOP no Processamento de Mensagens do Mule ESB.

5.4 Detalhamento do Modelo

Inicialmente descreve-se a classe *Invocation* e método original do Mule para o funcionamento dos InterceptadoresAOP.

Invocation é uma classe também do pacote *org.mule.umo* que representa um link na cadeia dos InterceptadoresAOP. InterceptadoresAOP podem ser configurados ou modificados dentro do pacote *org.mule.interceptadorAop*. A classe *Invocation* possui o seguinte método construtor:

- *Invocation (UMOImmutableDescriptor descriptor, UMOMessage message, Invocation invocation)*, que inicializa uma invocação.
    - *descritor* é a descrição dos componentes de serviços;
    - *message* é a mensagem atual;
    - *invocation* é a próxima invocação na cadeia;

O protótipo dos InterceptadoresAOP é compostos por quatro tipos, o InterceptadorLogAop, InterceptadorDebugAop, InterceptadorTempoAop e InterceptadorExcecaoAop apresentados detalhadamente e individualmente nos parágrafos abaixo.

Sendo que para proporcionarmos um melhor entedimento, criou-se um Componente de Serviço chamado de FoneService, no qual o usuário informa o DDD e o serviço retorna o UF (Unidade Federativa) e o CEP do parâmetro correspondente.

A Figura 27 apresenta o Processamento de Mensagem no Mule ESB com o InterceptadorLogAop, responsável pela realização de logs dos dados antes e depois dos componentes de serviços.

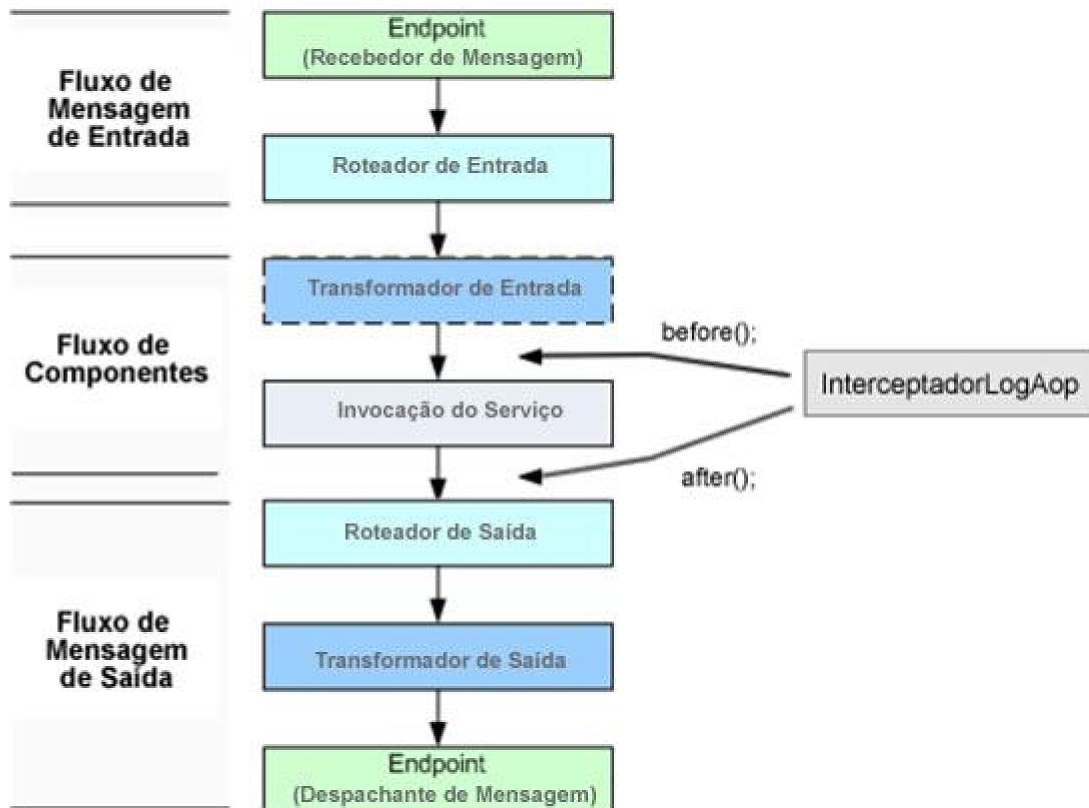


Figura 27. Processamento de Mensagens utilizando o InterceptadorLogAop.
Fonte: Elaboração própria, 2007.

O InterceptadorLogAop é composto pela classe *InterceptLog,* que realiza invocação ao Componente de Serviço através do método *Invocation*, apresentado anteriormente. E pelo Aspecto *InterceptLogAop* que possui os seguintes elementos:

- Um ponto de atuação chamado de *greeting* que executa o método *InterceptLog.intercept(..)*;
- Um *advice before( )* indicando o ponto de atuação *greeting*, responsável pela realização de log de dados antes do evento *InterceptLog.intercept(..)* ser processado;

- Um *advice after( )* indicando o ponto de atuação *greeting*, responsável pela realização de log de dados depois do evento *InterceptLog.intercept(..)* ser processado;

O InterceptLogAop realiza o log da data, horário e do Componente de Serviço, neste caso o FoneServiceUMO, conforme pode-se visualizar na Figura 28 abaixo.

```
Informe o DDD a ser consultado: 11

INFO  2007-11-05 17:27:49,296 [FoneServiceUMO.2]
org.mule.interceptadorAop.InterceptLogAop
: LOG - Antes - UMOMessage
org.mule.interceptadorAop.InterceptAop.intercept(Invocation)

INFO  2007-11-05 17:27:49,328 [FoneServiceUMO.2]
org.mule.interceptadorAop.InterceptLogAop
: LOG - Depois – UMOMessage
org.mule.interceptadorAop.InterceptAop.intercept(Invocation)

INFO  2007-11-05 17:27:49,375 [SystemStreamConnector.dispatcher.1]
org.mule.providers.stream.StreamMessageDispatcher: Connected:
StreamMessageDispatcher{this=1989b5, endpoint=stream://System.out}

UF: SP | CEP: 07500-000
```

Figura 28. Saída padrão da execução do InterceptadorLogAop.
Fonte: Elaboração própria, 2007.

A Figura 29 apresenta o Processamento de Mensagem no Mule ESB com o InterceptadorDebugAop, responsável pela realização da depuração durante da invocação dos componente de serviços.

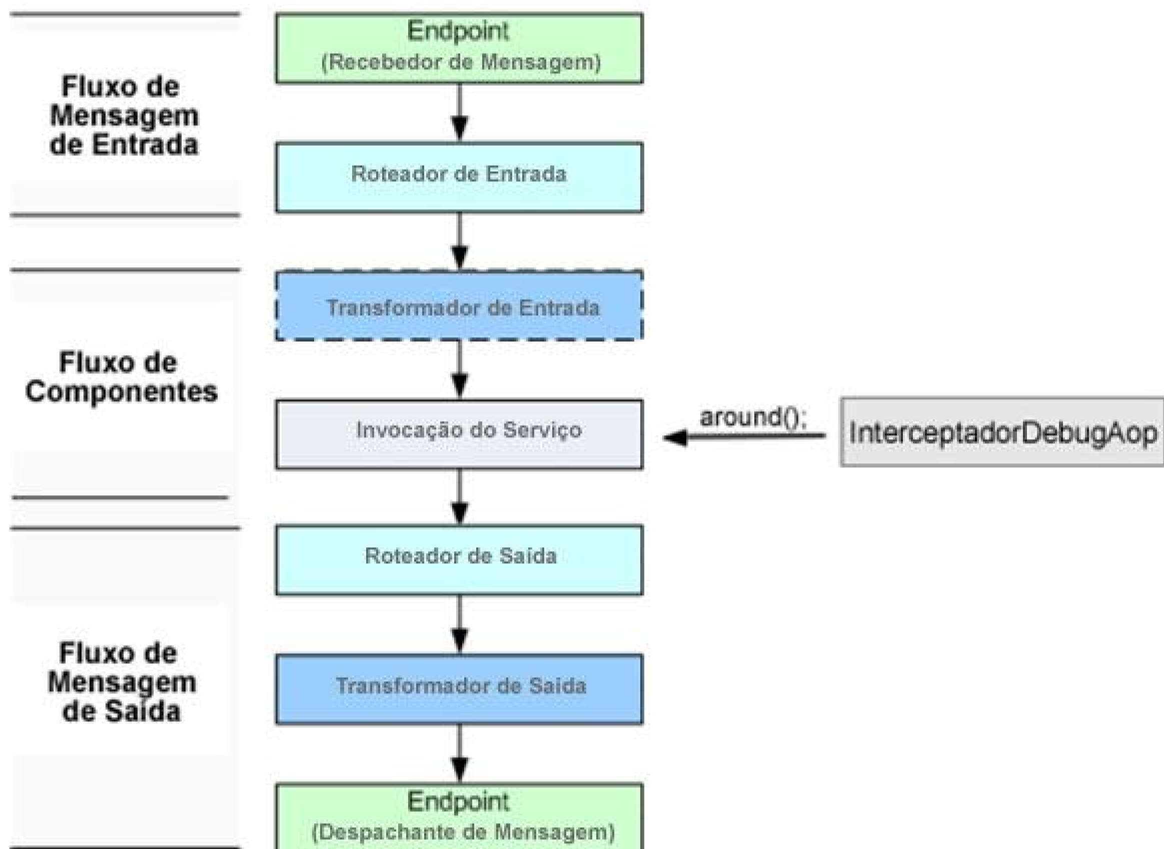


Figura 29. Processamento de Mensagens utilizando o InterceptDebugAop.
Fonte: Elaboração própria, 2007.

O InterceptadorDebugAop é composto pela classe *InterceptDebug,* que realiza invocação ao Componente de Serviço através do método *Invocation*. E pelo Aspecto *InterceptDebugAop* que possui os seguintes elementos:

- Um ponto de atuação chamado de pontoDebug que executa o método *InterceptDebug.intercept(..)* ;
- Um *advice around( )* indicando o ponto de atuação pontoDebug, sendo responsável pela realização do debug durante o processamento do evento *InterceptDebug.intercept(..)*;

O InterceptDebugAop descreve os métodos e classes acessadas, mensagens transmitidas, parâmetros passados e o nome da localização no código fonte da classe, conforme pode ser visualizado na Figura 30 abaixo.

```
Informe o DDD a ser consultado: 11

Entrando no método: execution(UMOMessage
org.mule.interceptadorAop.InterceptAop.intercept(Invocation))

Mensagem: intercept
da classe: org.mule.interceptadorAop.InterceptAop

Parametros:
  0. invocation : org.mule.umo.Invocation =
org.mule.interceptors.InterceptorStack$Invoc@1bc16f0

Localização no fonte: InterceptAop.java:31

UF: SP | CEP: 07500-000
```

Figura 30. Saída padrão da execução do InterceptDebugAop.
Fonte: Elaboração própria, 2007.

InterceptadorTempoAop é descrito no Processamento de Mensagem do Mule ESB, sendo responsável por medir o tempo total, antes e depois da invocação dos Componentes de Serviços, como pode ser visualizado na Figura 31 abaixo.

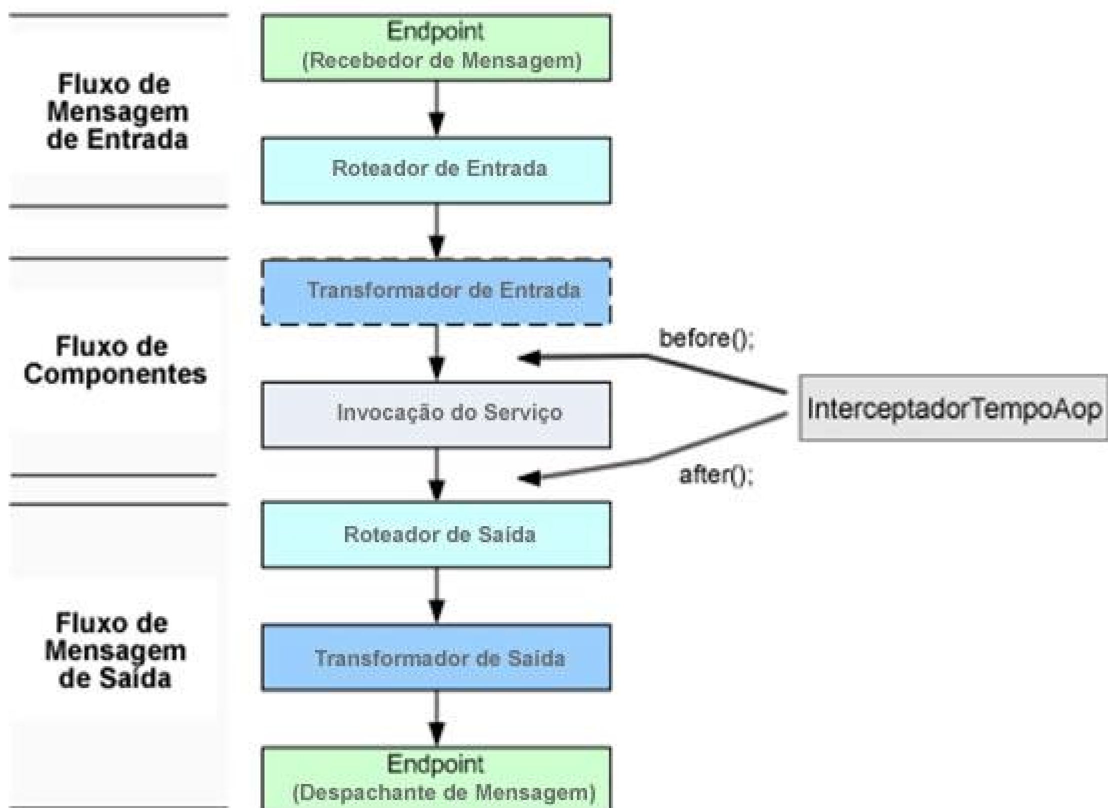


Figura 31. Processamento de Mensagens utilizando o InterceptTempoAop.
Fonte: Elaboração própria, 2007.

O *InterceptadorTempoAop* é composto pela classe *InterceptTempo,* que realiza invocação ao Componente de Serviço através do método *Invocation*. E pelo Aspecto *InterceptTempoAop* que é composto pelos seguintes elementos:

- Dois atributos do tipo *long* chamados de início e fim;
- Um ponto de atuação chamado de pontoTempo( ) que executa o método *InterceptLog.intercept(..)*;
- Um *advice before( )* indicando o ponto de atuação pontoTempo( ), sendo responsável por marcar o tempo de início antes do evento *InterceptLog.intercept(..)* ser processado;
- Um *advice after( )* indicando o ponto de atuação pontoTempo( ), sendo responsável por marcar o tempo final, depois do evento *InterceptLog.intercept(..)* ser processado e imprimindo o tempo total em segundos e milisegundos, conforme apresentado na Figura 32 abaixo.

```
Informe o DDD a ser consultado: 11

INFO  2007-11-07 17:27:49,328 [FoneServiceUMO.2]
org.mule.interceptadorAop.InterceptTempoAop:
Total de tempo 0.032 Segundos / 32 Milisegundos

INFO  2007-11-07 17:27:49,375 [SystemStreamConnector.dispatcher.1]
org.mule.providers.stream.StreamMessageDispatcher: Connected:
StreamMessageDispatcher{this=1989b5, endpoint=strea
m://System.out}

UF: SP | CEP: 07500-000
```

Figura 32. Saída padrão da execução do InterceptTempoAop em tempo total de 0.032 segundos.
Fonte: Elaboração própria, 2007.

E por fim, tem-se o InterceptadorExcecaoAop no Processamento de Mensagem no Mule ESB, responsável por tratar as exceções durante a invocação dos componentes de serviços, podendo ser visto na Figura 33 abaixo.

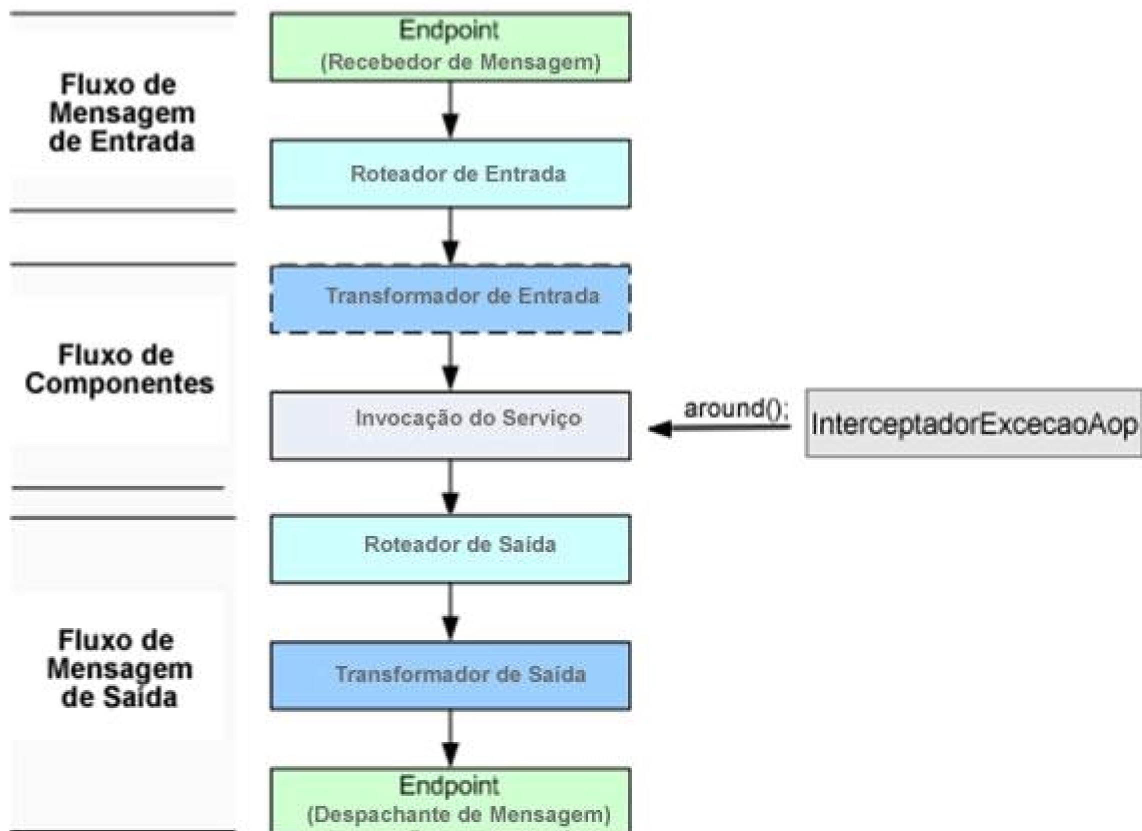


Figura 33. Processamento de Mensagens utilizando o InterceptExcecaoAop.
Fonte: Elaboração própria, 2007.

O *InterceptadorExcecaoAop* contém a classe *InterceptExcecao,* que realiza invocação ao Componente de Serviço através do método *Invocation* e o Aspecto *InterceptExcecaoAop* composto pelos seguintes elementos:

- Um ponto de atuação chamado de pontoExcecao que executa o método *InterceptDebug.intercept(..)* ;

- Um *advice around( )* indicando o ponto de atuação pontoExcecao, para a realização de uma exceção que ocorra durante o processamento do evento *InterceptLog.intercept(..)*;

O InterceptExcecaoAop realiza o tratamento de exceções durante a invocação do Componente de Serviço, mostrando uma exceção quando o cliente passar o parâmetro vazio, conforme mostrado na Figura 34 abaixo.

```
Informe o DDD a ser consultado:

INFO  2007-11-07 17:27:49,328 [FoneServiceUMO.2]
org.mule.interceptadorAop.InterceptExcecaoAop:
Parâmetro nulo !!

INFO  2007-11-07 17:27:49,375 [SystemStreamConnector.dispatcher.1]
org.mule.providers.stream.StreamMessageDispatcher: Connected:
StreamMessageDispatcher{this=1989b5, endpoint=strea
m://System.out}

Informe o DDD a ser consultado:
```

Figura 34. Saída padrão da execução do InterceptExcecaoAop.
Fonte: Elaboração própria, 2007.

Os interceptadores apresentados nesta seção, permitem alterações nos comportamentos dos Componentes de Serviços antes, durante e depois, utilizando Aspectos e o método de invocação do Mule. Criando assim, um meio para o tratamento de interesses transversais utilizando enumeração, baseados na semântica da AspectJ.

Nas próximas seções são apresentadas as ferramentas e a configuração do ambiente para o desenvolvimento e execução dos InterceptadoresAOP.

5.5 Ferramentas

Para o desenvolvimento dos InterceptadoresAOP utilizou-se o Eclipse, que é uma IDE de código aberto para a construção de programas de computador. O projeto Eclipse foi iniciado na empresa IBM (*International Business Machines*) que desenvolveu a primeira versão do produto e doou-o como software livre para a comunidade.

Atualmente o Eclipse é a IDE Java mais utilizada no mundo. Possui como características marcantes o uso da SWT (*Standard Widget Toolkit*) como biblioteca gráfica, a forte orientação ao desenvolvimento baseado em plug-ins e o amplo suporte ao desenvolvedor com centenas de plug-ins que procuram atender as diferentes necessidades de diferentes programadores.

Foram utilizados dois plug-ins no Eclipse para a criação dos InterceptadoresAOP e do estudo caso aplicado neste trabalho, o plug-in Mule IDE e o AspectJ Development Tools (AJDT).

O plug-in Mule IDE oferece um ambiente integrado para o desenvolvimento no Mule nas versões 1.4.x no Eclipse.

O AspectJ Development Tools (AJDT) é um plug-in que fornece a plataforma do Eclipse baseada na ferramenta de suporte do Desenvolvimento Orientado a Aspecto com a AspectJ. O seu principal objetivo é conceder uma experiência de usuário que é consistente com o plug-in Java Development Tools (JDT), quando se trabalha com projetos e recursos da AspectJ.

Utilizou-se também o log4j, que é uma biblioteca livre de código aberto desenvolvido pela Apache Software Foundation. Ele fornece uma API para que o desenvolvedor de software possa fazer log de dados nas aplicações.

5.6 Configuração do Ambiente

Com as ferramentas já apresentadas na seção anterior, nesta, são descritos os passos necessários para o funcionamento correto do estudo de caso.

Inicialmente devem-se ter o JDK (Java Development Kit) e o Mule ESB instalados e corretamente configurados as suas variáveis de ambiente. A versão utilizada do JDK foi a 1.6.0_01, que pode ser obtida no site da SUN e a última versão do Mule ESB que é a 1.4.3.

A versão utilizada da IDE Eclipse, foi a 3.3, para o suporte no desenvolvimento e utilização do cenário.

Na Figura 35, criou-se um projeto com suporte a AspectJ no Eclipse, com um plugin AspetctJ Tools, que pode ser adquirido no site oficial do Eclipse.

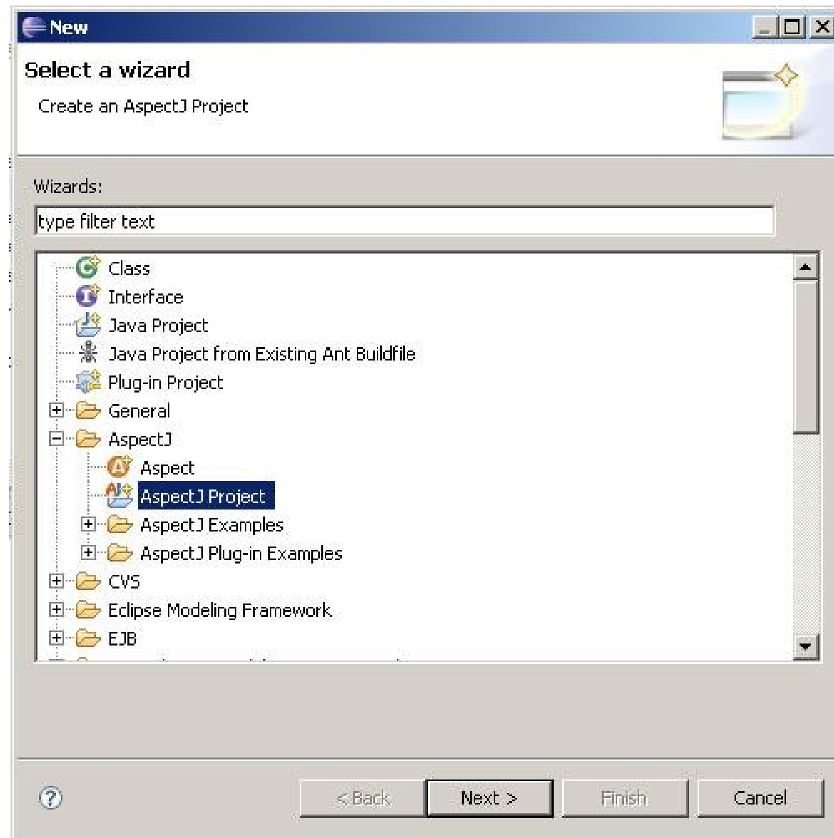


Figura 35: Criação de um projeto com suporte a AspectJ.
Fonte: Elaboração própria, 2007.

Definiu-se o nome do projeto e algumas configurações padrões do JAVA. E por último, foram adicionadas as variáveis de ambiente MULE_HOME e as bibliotecas (arquivos .jars) do Mule ESB localizadas na pasta lib/mule no diretório de instalação do Mule ESB, figura 36.

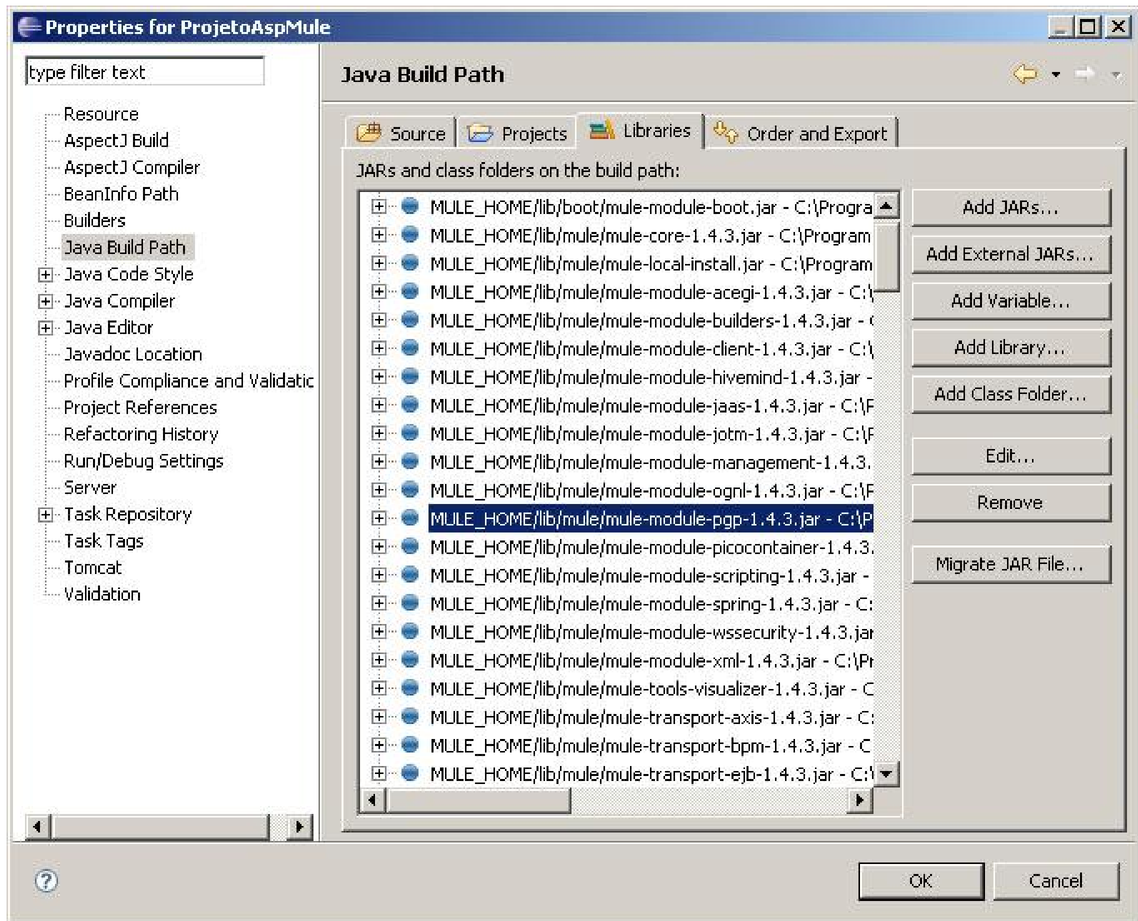


Figura 36: Variáveis de ambiente do MULE no Eclipse.
Fonte: Elaboração própria, 2007.

Depois de criado o projeto com suporte a AspectJ, configurada as variáveis de ambiente do Mule juntamente com suas bibliotecas padrões, deve-se adicionar a biblioteca (arquivo .jar) dos InterceptadoresAOP para que a aplicação deste, funcione satisfatoriamente.

5.7 Avaliação do Modelo

Com o objetivo de deixar mais claro o funcionamento do modelo proposto, esta seção demonstra um estudo de caso, porém, de complexidade reduzida do funcionamento dos InterceptadoresAOP.

5.7.1 Cenários

Para exemplificar de um modo mais simples e claro os InterceptadoresAOP, resolveu-se criar dois cenários para os diferentes tipos de interceptadores suportados pelos InterceptadoresAOP.

No primeiro cenário é apresentada a integração do Mule ESB com os InterceptadoresAOP no lado do Componente de Serviço. Já no segundo cenário, é demonstrada a integração no lado dos clientes que acessam o Componente de Serviço. Deste modo, aborda-se tanto no lado do Componente de Serviço (Serviço Web) quanto no lado do cliente. Os códigos fontes relacionados aos cenários podem ser vistos nos apêndices A, B.

5.7.2 Cenário 1

Uma empresa de telefonia, nomeada de Z, possui um sistema de grande porte desenvolvido em Java (JEE, EJB e outros) com banco de dados Oracle.

A empresa Z fez um convênio com as seguintes empresas, para fornecer acesso aos dados de seus clientes:

- Empresa de venda de roupas, com um sistema implementado em .NET;
- Concessionária de automóveis, com um sistema implementado em Cobol;
- Empresa de eletroeletrônicos que possui uma loja on line B2C (Business-to-Consumer) desenvolvida em PHP;

Com este cenário as empresas esperam obter os seguintes benefícios:

- Comprovação rápida de residência;
- Maior agilidade e comodidade no cadastramento dos clientes nas empresas conveniadas;

- Maior confiança nos dados dos clientes fornecidos pela empresa de telefonia Z;

Por ter uma base de dados extensa em termos de registros armazenados, uma das exigências dos diretores da empresa Z era que fosse cobrado não apenas por acesso, mas também por tempo gasto por cada empresa conveniada na utilização dos serviços. Sendo concordado pelas empresas conveniadas, uma vez que, toda e qualquer exceção por parâmetro de entrada de dados vazios ou nulos, não contabilizasse o tempo.

No levantamento de requisitos ficaram definidos os parâmetros de entrada e saída, no qual estão descritos abaixo.

<u>Parâmetros de entrada:</u>

- DDD (Discagem Direta a Distância);
- Número do telefone;

<u>Parâmetros de saída:</u>

- Nome completo;
- Endereço;
- Bairro;
- Município;
- Estado;
- CEP;

Definido o cenário, agora será apresentada a solução com os InterceptadoresAOP, detalhando o modelo da arquitetura e seus detalhes, implementação e execução destes.

5.7.2.1 Arquitetura

A principal dificuldade encontrada é com relação à integração de diferentes sistemas de um modo que possa ser evoluído. Para isso, teriam várias opções como conectar os quatro sistemas diretamente ou desenvolver uma classe (ou todo um pacote de classes) que faça acesso aos quatro sistemas.

Com tempo, à medida que surgissem outras necessidades de conectar estes quatro sistemas ou aumentasse o número de empresas conveniadas com aplicações em plataformas diferentes ou os interesses transversais mudassem, acabaria-se por ter uma confusão geral de sistemas acessando outros sistemas.

Para evitar isto, inseriu-se o papel do ESB para ser um negociador, uma interface onde os clientes (empresas conveniadas) solicitam a execução de alguns processos, consultas, etc. Ou seja, um elemento intermediário que é o responsável por conectar os diferentes sistemas e fornecer suporte para o tratamento de interesses transversais oferecidos pelos InterceptadoresAOP.

O sistema da empresa de telefonia Z em JAVA nem tem conhecimento de que outro sistema é feito em .NET ou em qualquer outra tecnologia. Porque ele se comunica apenas com o ESB, que por sua vez tem o papel de se conectar a estes outros sistemas.

De um modo geral, o ESB é uma abstração dessa interconexão dos diferentes sistemas do cenário. Sendo que para atendê-lo, criou-se o seguinte modelo que é apresentado na Figura 37 abaixo.

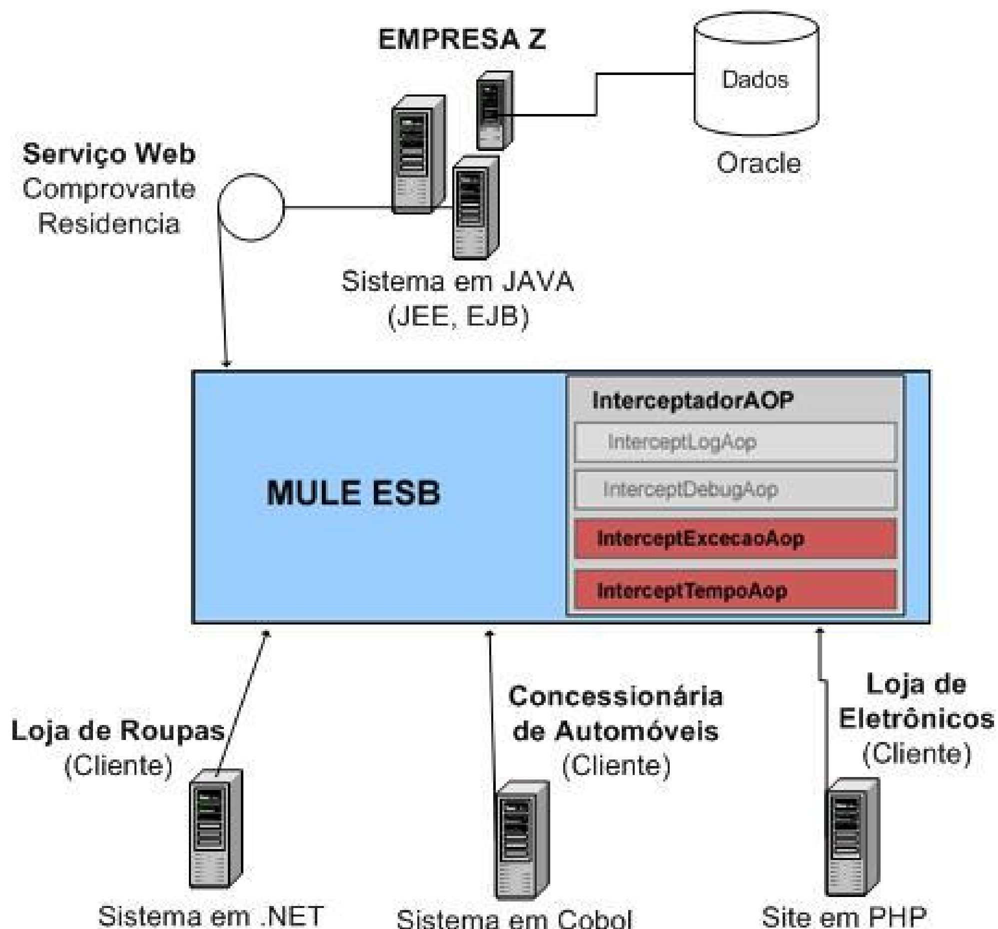


Figura 37. Arquitetura do cenário 1.
Fonte: Elaboração própria, 2007.

Na Figura 37 está descrito o sistema em JAVA da empresa de telefonia Z com o banco de dados em Oracle em seus servidores, onde se criou um Serviço Web chamado de ComprovanteResidencia, para disponibilizar os dados de seus clientes para as empresas conveniadas.

O Serviço Web ComprovanteResidencia foi desenvolvido em Java e integrado com o Mule ESB para fornecer as empresas conveniadas uma integração mais fácil de ser mantida e evoluída.

Com isto, tem-se todos os clientes (empresas conveniadas) acessando o serviço web ComprovanteResidencia via o Mule ESB.

5.7.2.2 Implementação e Execução

O serviço web ComprovanteResidencia tem a função de retornar o Nome completo, Endereço, Bairro, Município, Estado e o CEP de acordo com o parâmetro especificado pelo cliente, o DDD com o número do telefone. Este serviço foi implementado como uma classe JAVA no Mule ESB, conforme pode ser visualizado a sua execução no mais puro formato e sem a noção dos InterceptadoresAop na Figura 38 abaixo.

```
Problems | @ Javadoc | Declaration | Cross References | Spring Explorer | Debug | Console
FoneService [Java Application] C:\Program Files\Java\jdk1.6.0_01\bin\javaw.exe (Feb 7, 2008 10:59:09 PM)
* Server ID: ProjetoAspMule                                          *
* JDK: 1.6.0_01 (mixed mode)                                         *
* OS: Windows XP - Service Pack 2 (5.1, x86)                         *
* Host: VGN-S469 (192.168.1.100)                                     *
*                                                                    *
* Agents Running:                                                    *
*   Mule Admin: accepting connections on tcp://localhost:60504       *
**********************************************************************
INFO  2008-02-07 22:59:13,484 [main] org.mule.MuleServer: Mule Server initialized.
INFO  2008-02-07 22:59:25,609 [connector.http.0.receiver.2] org.mule.providers.stream.Stre
amMessageDispatcher: Connected: StreamMessageDispatcher(this=2d09e0, endpoint=stream://Sys
tem.out)
Luiza Maria Sousa#Av. José Ribeiro#Bequimão#São Luís#MA#65000-000
```

Figura 38. Exemplo do Serviço Web ComprovanteResidencia sem InterceptadoresAOP.
Fonte: Elaboração própria, 2007.

No Serviço Web ComprovanteResidencia na Figura 38, a sua execução solicita ao usuário, que informe o DDD para consulta do CEP e UF. O parâmetro solicitado pelo o usuário é “21” e a resposta do evento é a informação da sigla do estado do Rio de Janeiro – RJ e o CEP do mesmo.

Já na Figura 39, pode-se visualizar a execução no console do Serviço Web ComprovanteResidencia com a noção dos InterceptadoresAOP já com a integração da AspectJ rodando no Mule ESB.

```
Problems  Javadoc  Declaration  Cross References  Spring Explorer  Debug  Console
FoneService [Java Application] C:\Program Files\Java\jdk1.6.0_01\bin\javaw.exe (Feb 7, 2008 10:37:47 PM)
* Server ID: ProjetoAspMule                                          *
* JDK: 1.6.0_01 (mixed mode)                                         *
* OS: Windows XP - Service Pack 2 (5.1, x86)                         *
* Host: VGN-S469 (192.168.1.100)                                     *
*                                                                    *
* Agents Running:                                                    *
*   Mule Admin: accepting connections on tcp://localhost:60504       *
**********************************************************************
INFO  2008-02-07 22:37:51,234 [main] org.mule.MuleServer: Mule Server initialized.

: INFO  2008-02-07 22:39:30,078 [connector.http.0.receiver.2] org.mule.interceptadorAop.In
terceptTempoAop: Total de tempo 0.016 Segundos / 16 Milisegundos

INFO  2008-02-07 22:39:30,093 [connector.http.0.receiver.2] org.mule.providers.stream.Stre
amMessageDispatcher: Connected: StreamMessageDispatcher{this=1b9ef36, endpoint=stream://Sy
stem.out}
Mariana Bastos Sousa Paiva#Av. Souza Peixoto#Entroncamento#Belem#PA#20080-001
```

Figura 39. Execução do Serviço Web ComprovanteResidencia com InterceptadorTempoAOP.
Fonte: Elaboração própria, 2007.

Nesta figura 39, pode-se visualizar o medidor de tempo na execução do evento. O parâmetro solicitado pelo o usuário é o número “21” e a resposta do evento é a informação da sigla do estado do Rio de Janeiro – RJ e o CEP do mesmo. Só que neste caso são utilizados dois tipos dos InterceptadoresAop, o InterceptadorTempoAop e o InterceptadorExcecaoAop.

O InterceptadorTempoAop antes de processar o componente de serviço armazena o tempo inicial da invocação, utilizando o *advice before*. E após processado o componente de serviço ele calcula o tempo total (tempo final - tempo inicial) utilizando o *advice after* e exibe a mensagem “*Total de tempo 0.016 Segundos / 16 Milisegundos*”.

Na Figura 40 abaixo, visualiza-se uma exceção na execução do componente de serviço utilizando o InterceptadorExcecaoAop. O parâmetro solicitado pelo o usuário foi nulo e o retorno durante o processamento do componente de serviço foi a mensagem da exceção “*Parâmetro Nulo !!*”

```
Problems  Javadoc  Declaration  Cross References  Spring Explorer  Debug  Console
FoneService [Java Application] C:\Program Files\Java\jdk1.6.0_01\bin\javaw.exe (Feb 7, 2008 11:05:23 PM)
* For more information go to http://mule.mulesource.org                *
*                                                                      *
* Server started: 2/7/08 11:05 PM                                      *
* Server ID: ProjetoAspMule                                            *
* JDK: 1.6.0_01 (mixed mode)                                           *
* OS: Windows XP - Service Pack 2 (5.1, x86)                           *
* Host: VGN-S469 (192.168.1.100)                                       *
*                                                                      *
* Agents Running:                                                      *
*   Mule Admin: accepting connections on tcp://localhost:60504         *
************************************************************************
INFO  2008-02-07 23:05:27,296 [main] org.mule.MuleServer: Mule Server initialized.

: INFO  2008-02-07 23:05:29,156 [ComprovanteResidenciaUMO.2] org.mule.interceptadorAop.Int
erceptExcecaoAop: Parametro Nulo!!
```

Figura 40. Execução do Serviço Web ComprovanteResidencia com o InterceptadorExcecaoAop.
Fonte: Elaboração própria, 2007.

5.7.3 Cenário 2

Neste segundo cenário, vamos idealizar uma empresa da área financeira que desenvolve e oferece serviços web para seus clientes de uma forma ágil e segura, também conhecida como Centros de Serviços, chamada aqui de empresa Y.

Esta empresa disponibiliza um Serviço Web para realizar consultas ao CPF de pessoas ao Mainframe do SERASA. Para possibilitar a integração das suas aplicações com as dos clientes e sites web.

Dois bancos fecharam contrato com a empresa Y para acessar o Serviço Web que fornece os dados dos usuários com pendência:

- Banco M, com um sistema implementado em JAVA;
- Banco K que possui um sistema web implementado em .NET;

Neste cenário, os bancos esperam obter os seguintes benefícios:

- Verificação rápida da situação das pessoas que desejam abrir novas contas;
- Maior agilidade e comodidade na consulta às informações;

O Banco M necessita de logs de consulta dos usuários ao Serviço Web. Obtendo um histórico de acessos (data e hora) dos funcionários e consequentemente, uma maior tranquilidade em possíveis auditorias ou fiscalizações que porventura vierem a ocorrer.

A equipe de informática do Banco M verificou também a possibilidade de inserir um rastreamento e debug no acesso ao Serviço Web em função de seus desenvolvedores e clientes que geralmente integram suas aplicações locais ou sites web ao serviço. Fornecendo assim uma maior agilidade e compreensão no desenvolvimento de integração de sistemas com outras empresas.

O parâmetro de entrada do serviço é o CPF da pessoa, já disponibilizado pela empresa Y.

Já os parâmetros de saída são os seguintes:

- Nome completo;
- Nome do pai e mãe;
- Situação;

Definido o cenário, agora será apresentada a solução com os InterceptadoresAOP, especificando o modelo da arquitetura, implementação e execução destes.

5.7.3.1 Arquitetura

Como visto neste cenário, quem deseja agora inserir interesses ao serviço é o cliente, no caso o Banco M. Que entrou em contato com a empresa Y para verificar se eles poderiam fornecer estes interesses transversais. Sendo que estes informaram que seria inviável por alguns fatores.

Com isso, o Banco M decidiu utilizar o Mule ESB com os InterceptadoresAOP. Sendo que, o Mule ESB entrou como um elemento intermediário, responsável por conectar os diferentes sistemas e fornecer suporte para o tratamento de interesses transversais. Pois o Banco M necessitaria compartilhar esta aplicação cliente com várias filiais e departamentos espalhados pelo mundo.

Para atender os requisitos deste cenário, criou-se o seguinte modelo que é apresentado na Figura 41 abaixo.

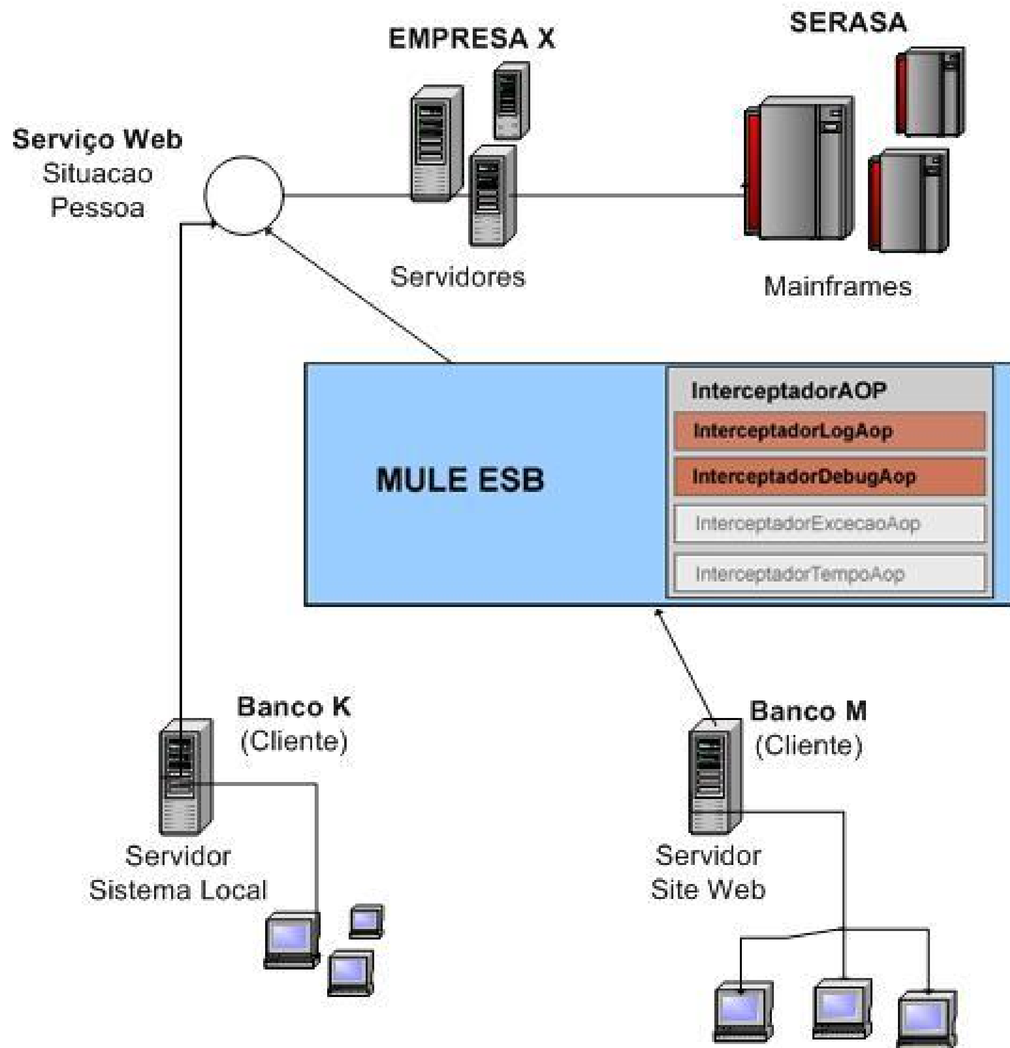


Figura 41. Arquitetura do cenário 2.
Fonte: Elaboração própria, 2007.

Na Figura 41 está descrito o Serviço Web, chamado de SituacaoPessoa, para realizar consultas ao CPF de pessoas no Mainframe do SERASA, os clientes do serviço, o Banco M com um sistema em Java e suas filiais ou departamentos e o Banco K com um sistema web implementado em .NET.

No Banco M foi desenvolvido um cliente em JAVA, agregado com o Mule ESB para fornecer integrações mais flexíveis, mais fáceis de serem mantidas e evoluídas transversalmente.

Com isto, tem-se todos os dependentes do cliente do Banco M acessando o Serviço Web SituacaoPessoa via o Mule ESB.

5.7.3.2 Implementação e Execução

O serviço web SituacaoPessoa tem a função de retornar o Nome completo, Nome do Pai e da Mãe e a Situação da pessoa de acordo com o parâmetro especificado pelo cliente, o CPF. Não sabemos em qual plataforma foi implemetado este serviço.

Na figura 42 abaixo, pode-se visualizar uma parte do código do cliente Mule desenvolvido em Java para acessar o serviço.

```
MuleClient client = new MuleClient();

String[ ] param = new String[ ] {"102.221.222.01"};

UMOMessage result =
    client.send
    ("http://www.bancom.br/situacaopessoa/soapserver.php?method=getPessoa",
    param, null);
```

Figura 42. Parte do código do cliente Mule.
Fonte: Elaboração própria, 2007.

No cliente Mule é definido o parâmetro do CPF com o valor “102.221.222.01”, sendo realizada uma chamada ao método client.send( ) contendo o endereço de localização do serviço com o seu método e parâmetro (Figura 41).

Na Figura 43, pode-se visualizar a execução do Serviço Web SituacaoPessoa com a noção dos InterceptadoresAOP no Mule ESB.

```
Problems  @ Javadoc  Declaration  Cross References  Spring Explorer  Debug  Console
FoneService [Java Application] C:\Program Files\Java\jdk1.6.0_01\bin\javaw.exe (Feb 7, 2008 11:43:29 PM)
INFO  2008-02-07 23:43:33,750 [main] org.mule.MuleServer: Mule Server initialized.

: INFO  2008-02-07 23:43:41,859 [connector.http.0.receiver.2] org.mule.interceptadorAop.In
terceptLogAop: LOG - Antes - UMOMessage org.mule.interceptadorAop.InterceptLog.intercept(I
nvocation)

Entrando no método: execution(UMOMessage org.mule.interceptadorAop.InterceptDebug.intercep
t(Invocation))
Mensagem: intercept
da classe: org.mule.interceptadorAop.InterceptDebug
Parametros:
  0. invocation : org.mule.umo.Invocation = org.mule.interceptors.InterceptorStack$Invoc@1
cea096
Localização no fonte: InterceptDebug.java:18
INFO  2008-02-07 23:43:41,859 [connector.http.0.receiver.2] org.mule.interceptadorAop.Inte
rceptLogAop: LOG - Depois - UMOMessage org.mule.interceptadorAop.InterceptLog.intercept(In
vocation)

INFO  2008-02-07 23:43:41,875 [connector.http.0.receiver.2] org.mule.providers.stream.Stre
amMessageDispatcher: Connected: StreamMessageDispatcher{this=34d75f, endpoint=stream://Sys
tem.out}
Rodrigues Ramos Silva#José Carlos Sobrinho Souza#Maria Cardoso Nunes Almeida#S
```

Figura 43. Execução do serviço web SituacaoPessoa com os InterceptadoresAOP.
Fonte: Elaboração própria, 2007.

Inicialmente, pode ser visualizado um log de dados na execução do evento do Serviço (Figura 43). O parâmetro solicitado pelo cliente é o número “102.221.222.01” e a resposta do serviço é descrita abaixo:

- Nome completo: “Rodrigues Ramos Silva”;
- Nome do Pai e da Mãe: “José Carlos Sobrinho Souza” e “Maria Cardoso Nunes Almeida”;
- Possui restrição: “S”;

Neste cliente estamos utilizando os InterceptadoresAOP antes de processar o evento do serviço, fazendo o Log exibindo a data, o horário e uma mensagem “*INFO 2008-02-07 23:43:41 LOG: Antes*”.

Durante o processamento do Componente de Serviço é realizado o debug descrevendo os métodos e classes acessadas, mensagens transmitidas, parâmetros passados.

E por último, tem-se outro Log exibindo a data, horário e mensagem "*INFO 2008-02-07 23:43:41 - LOG: Depois*".

6. CONCLUSÃO E TRABALHOS FUTUROS

Este capítulo discute sobre as contribuições deste trabalho, considerações finais sobre os resultados alcançados, as limitações e sugestões para trabalhos futuros.

6.1 Contribuições do Trabalho

Dentre as principais contribuições estão relacionados à inserção da semântica da AOP ao modelo proposto, que agrega valores inovadores no desenvolvimento de software, serviços, podendo afirmar que esta forma de programar não corresponde a um novo paradigma, mas sim um complemento de um paradigma predominante (programação orientada a objetos).

Algumas contribuições importantes mais específicas são percebidas:

- A possibilidade de implementar propriedades sistêmicas ou interesses transversais explicitamente em serviços;
- Inserção de novos meios de composição que permitem a descrição modular;
- Redução do espalhamento e entrelaçamento de código fonte por vários serviços, clientes, componentes facilitando a legibilidade e manutenção;
- Flexibilidade na integração de interesses transversais aos serviços web, por conta da Arquitetura Orientada a Serviço;
- Possibilidade de criação de novos interesses transversais ou alteração dos interesses já pré-definidos nos InterceptadoresAOP, pelo fato do código ser open source;
- Solução não intrusiva. Possibilidade de enumerar pontos de atuação e advices *before*, *after*, *around* sem a necessidade de conhecer a estrutura interna do serviço;

- Comprovação do modelo no lado do serviço e também do cliente, sendo apresentados e solucionados dois cenários;
- Baixo acoplamento entre serviços e aplicações, visto que um sistema de origem não necessita conhecer a plataforma ou implementação do sistema de destino e vice-versa;
- Baixo custo entre integrações de soluções empresariais, em função das ferramentas utilizadas nos InterceptadoresAOP serem todas *Open Source*.

6.2 Considerações Finais

Esta dissertação apresentou uma aplicação prática dos conceitos padrões e ferramentas decorrentes do desenvolvimento da Arquitetura Orientada a Serviços, tecnologia de Serviços Web e da Programação Orientada a Aspectos como uma alternativa para aprimorar os atuais mecanismos de tratamento dos interesses transversais em Serviços Web nos *middlewares* ESB.

Enquanto se têm poucos serviços, tudo pode ir bem sem ESB. O problema é quando se começa a ter dezenas ou até centenas de serviços para gerir. Cada um deles terá necessidades e interesses transversais diferentes como autenticação, log de dados, transações, desempenho, cache para aumentar a sua disponibilidade e escalabilidade, etc.

A possibilidade de tratamento de interesses tranversais nos Serviços Web com enumerações, integrações mais flexíveis com outras tecnologias e modularizadas representam um avanço em relação às abordagens existentes na literatura. Com Aspectos que podem ser melhor reutilizados, mantidos, evoluídos e legíveis. Nos quais, novos serviços, clientes podem ser criados, modificados e fornecidos de forma a promover uma melhor integração e maior interoperabilidade com elementos do ambiente interno e externo das organizações.

Nesta, foram propostos também quatro tipos de interceptadores como o InterceptadorLogAop para o log de dados, InterceptadorDebugAop para o *debug* de serviços, InterceptadorTempoAop para medir o tempo de invocação do serviço e o

InterceptadorExcecaoAop responsável por tratar as exceções durante a invocação dos serviços.

Os InterceptadoresAOP foram aplicados em dois tipos de cenários. Sendo abordados no lado do serviço e cliente. Verificando assim, a diminuição da mão-de-obra na manutenção e evolução dos serviços, em função da semântica da AspectJ.

Concluiu-se neste trabalho com a expectativa de ter contribuído para melhor entendimento e divulgação dos assuntos tratados. Conscientes do desafio que ainda tem-se pela frente. Entretanto, o desenvolvimento do protótipo, estudos de casos e contribuições até agora obtidas, encorajam ao aprofundamento dos estudos nesta desafiadora linha de pesquisa. Espera-se também ter despertado o interesse por essa linha de pesquisa naqueles que compartilham o desejo na redução dos esforços para o desenvolvimento e integração de aplicações distribuídas.

6.3 Limitações

Apesar do alcance de pontos positivos com trabalho realizado, alguns pontos não poderam ser ultrapassados, indicando algumas limitações no modelo proposto do trabalho.

Algumas limitações nos InterceptadoresAOP podem ser citadas, como:

- O limite de quatro tipos de interesses transversais pré-definidos, log de dados, exceção, tempo e debug;
- Modelo aplicado e testado em um *middleware* ESB;
- Interesses transversais de menores complexidades em sua lógica interna;

Tais limitações podem ser futuramente sanadas, algumas destas são propostas como trabalhos futuros, levando desta forma a um aprimoramento dos InterceptadoresAOP.

6.4 Trabalhos Futuros

A partir dos resultados e contribuições alcançados, outros estudos podem ser realizados, e novas técnicas e métodos podem ser incorporados. Como propostas para trabalhos futuros, podem ser citados:

- Estudar e implementar nos InterceptadoresAOP novos interesses transversais como transação, sincronização, segurança, entre outros;

- Aplicação e adaptação desse modelo para outros middlewares ESB tais como BEA Web Logic, BizTalk, entre outros;

- Estudo e implementação nos InterceptadoresAOP com padrões de log de dados, debug, medição de tempo e exceções mais complexos em sua lógica interna;

- Implementar ferramentas de suporte e monitoramento dos InterceptadoresAOP, de modo a tornar todo o processo de tratamento dos aspectos algo mais automatizado. Por exemplo, um *plug-in* para o Eclipse.

- Realizar testes de desempenho para o modelo proposto, possuindo cenários com centenas de serviços integrados.

Essas propostas visam aumentar a eficácia do modelo e torná-lo mais completo quanto ao tratamento de interesses transversais em Serviços Web nos *middlewares* ESB.

REFERÊNCIAS

ACTIVE GROUP HOME PAGE. 1997. Disponível em: <http://www.activex.org/>. Acesso em: 04 de julho de 2007.

ALONSO, G.; CASATI, F.; KONU, H.; MACHIRAJU, V. **Web Services – Concepts, Architectures e Applications**. Bookmaker Springer. 2003.

ARBAOUI, S.; DERNIAME, J.; OQUENDO, F.; VERJUS, H. **A comparative review of Process-Centered Software Engineering Environments**. Annals of Software Engineering, vol. 14, p. 311-340. Kluwer, The Netherlands, 2002.

BASS L.; CLEMENTS P.; KAZMAN R. **Software Architecture in Practice**. Boston, Addison-Wesley, 1998.

BEA SYSTEMS. **Introduction to WebLogic Platform**. Versão 8.1, Julho, 2003. Disponível em <http://e-docs.bea.com/platform/docs81/tour/intro.html#1064338>. Acesso em 02 de setembro de 2007.

BONÉR, J., VASSEUR, A. **Enabling Aspect-Oriented Programming in WebLogic Server using the JRockit Management API**. Maio de 2004. Disponível em: <http://dev2dev.bea.com/pub/a/2004/05/boner_vasseur.html> . Acesso em: 10 de junho de 2007.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **The Unified Modeling Language User Guide**, Addison-Wesley, 1999.

BOTTO, R. **Arquitetura Corporativa de Tecnologia da Informação**. Rio de Janeiro: Brasport, 2004.

BROCKSCHMIDT, Kraig. **Inside OLE**, 2nd edition, Microsoft Press, 1995.

CHAMPION, M.; FERRIS,C.; NEWCOMER, E.; ORCHARD, D. **Web Services Architecture W3C Working Draft**. Novembro. 2002. Disponível em: www.w3.org/TR/ws-arch/ . Acesso em: 03 jun. 2007.

CHAVES, Rafael; ALVES. **Aspectos e MDA Criando modelos executáveis baseados em aspectos.** Dissertação deMestrado (Mestrado em Ciência da Computação) Universidade Federal de Santa Catarina, Florianópolis, 2004

CHRISTENSEN, Erik; CURBERA, Francisco; MEREDITH, Greg; WEERAWARANA, Sanjiva. **Web Services Description Language (WSDL) 1.1**. Março de 2001. Disponível em: < http://www.w3.org/TR/wsdl >. Acesso em: 08 de agosto 2007.

COLAN, M. **Service-Oriented Architecture expands the vision of Web services.** Abril de 2004. Disponível em: <http://www.ibm.com/developerworks/webservices/library/ws-soaintro.html?S_TACT=105AGX04&S_CMP=LP>. Acesso em: 15 de agosto de 2007.

CORBA. **The Common Object Request Broker Architecture**. Janeiro de 1997. Disponível em <http://www.sei.cmu.edu/str/descriptions/corba.html>. Acesso em: 19 de agosto de 2007.

COURBIS, Carine; FINKELSTEIN, Anthony. **Towards aspect weaving applications**. In 27th International Conference on Software Engineering, páginas 69–77, Outubro 2005.

DAVENPORT, Thomas **Reengenharia de Processos**. Rio de Janeiro: Campus, 1994.

DCOM. **Distributed Component Object Model**. Disponível em: <http://www.microsoft.com/com>. Acesso em 01 de junho de 2007.

DIJKSTRA, Edsger W. (1982), **On the role of scientific thought**, in Dijkstra, Edsger W., Selected writings on Computing: A Personal Perspective, New York, NY, USA: Springer-Verlag New York, Inc., pp. 60–66, ISBN 0-387-90652-5

FANTINATO, M.; TOLETO, M.; GIMENES, I. **Arquitetura de Sistemas de Gerenciamento de Processos de Negócio Baseado em Serviços**. Relatório técnico, UNICAMP, 2005.

FERRIS, C.; FARRELL, J. **What are web services?** Communications of the ACM, New FERRIS York, v. 46, n. 6, Jun. 2003. p. 31.

FILMAN, R.; ELRAD, T.; CLARKE, S.; AKSIT, M. **Aspect-Oriented Software Development**. Addison-Wesley, 2005.

FUGGETTA, Alfonso. **Software Process: A Roadmap**. Em: Finkelstein (Ed.), Future of Software Engineering. ACM Press, 2000. Também publicado em 22th International Conference on Software Engineering, ICSE'2000, Limerick, Ireland. Proceedings. ACM Press, Junho, 2000.

GIL, Antonio Carlos. **Como elaborar projetos de pesquisa.** 3. ed. São Paulo: Atlas, 1991.

GRADECKI, Joe; LESIECKI, Nicolas. **Mastering AspectJ: aspect-oriented programming in Java.** Indianopolis, Indiana. Wiley, 2003, p. 453.

GRAHAM, Steve. **Building web services with Java: making sense of XML, SOAP, WSDL, UDDI.** Indianopolis: Sams, 2002. 581 p. and UDDI. Indianapolis: Sams, 2002. 581 p.

HANSEN, Roseli Persson et al. **Web services: an architectural overview**. In: SEMINAR ON ELECTRONIC BUSINESS, 1., Nov. 2002. Rio de Janeiro. ADVANCED RESEARCH IN Proceedings… 2002. p. 44-57.

HANSEN, Roseli Persson. **GlueScript: uma linguagem específica de domínio para composição de web services**. São Leopoldo, 2003. 89 fl. Dissertação (Mestrado) – Universidade do Vale do Rio dos Sinos, Centro de Ciências Exatas e Tecnológicas, Programa Interdisciplinar de Pós-Graduação em Computação Aplicada.

HARMON, Paul. **Microsoft transaction Server. Component development Strategies.** Volume IX Número 3. Disponível em: <http://www.cutter.com/cds/1999toc.htm#mar>. Acesso em 16 de julho de 2007.

HORSTMANN, Markus; KIRTLAND, Mary. **DCOM Architecture.** Julho de 1997. Disponível em < http://msdn2.microsoft.com/en-us/library/ms809311.aspx >. Acesso em 21 de agosto de 2007.

IBM. **Definindo soluções em SOA e aplicando governança de projeto, técnica e operacional**. Disponível em: <https://www-304.ibm.com/jct09002c/university/scholars/courseware/repository/SOA/SW718/SW718Topic1.pdf>. Acesso em: 23 de junho de 2007

JEFFREY GRAY, G. **Aspect-Oriented Domain-Specific Modeling: A Generative Approach Using a Metaweaver Framework**, Tese (Ciência da Computação) Submitted to the Faculty of the Graduate School of Vanderbilt University in partial fulfillment of the requirements. Maio de 2002.

KAJKO-MATTSSON, M. **Evolution and Maintenance of Web Service Applications**. In IEEE International Confenrence on Software Maintenance, pages 492–493, 2004.

KICZALES, G.; HILSDALE, E.; HUGUNIN, J.; KERSTEN, M.; PALM, J.; GRISWOLD W. **Getting Started With AspectJ**. Communications of the ACM, Outubro 2001, vol. 44, no. 10.

KICZALES, G.; LAMPING, J.; MENDHEKAR, A; MAEDA, C.; VIDEIRA LOPES, C.; LOINGTIER, J.M.; IRWIN, J. **Aspect-Oriented Programming**. Proc.11th Europeen Conf. Object-Oriented Programming (Ecoop 97), Springer, junho 1997, pp.220-242.

KISELEV, Ivan. **Aspect Oriented Programming with AspectJ**, Ed. Sams Publishing, 2002.

LINTHICUM, D. **Steps to SOA Success**. OMG SOA Information Day. 2006. Disponível em: <http://soa.omg.org/SOA-Info-Day_12-06.htm>. Acesso em 20 de julho de 2007;

MACEDO N.; LEITE J. **Integrando Requisitos não-funcionais aos requisitos baseados em ações concretas.** Anais do II Workshop Iberoamericano de Engenharia de Requisitos e Ambientes de Software, Alajuela, Costa Rica, Março, 1999.

MAFFORT, C. Amaral; VALENTE, M. Túlio de Oliveira. **Aspectos para Construção de Serviços Web**. WASP. 2006.

MULE ESB. **Mule User Guide**, Março, 2006. Disponível em < http://mule.mulesource.org/display/MULEUSER/Home>. Acesso em 21 de setembro de 2007.

NEWCOMER, Eric. **Understanding web services: XML, SOAP, UDDI, and WSDL**. Boston: Addison-Wesley, 2002. 332 p.

NICKULL, D. **An Introduction to the OASIS Reference Model for Service Oriented Architecture**. Disponível em <http://www.oasis-open.org/committees/tc-home.phg?wg-abbrev=soa-rmem>. Acesso em: 25 de março de 2007.

NOTT, C. **Patterns: Using Business Service Choreography in Conjunction with an Enterprise Service Bus.** IBM Redbooks Paper. 2004.

OSSHER, H.; TARR, P. **Using subject-oriented programming to overcome common problems in object-oriented software development / evolution**. International Conference on Software Engineering, ICSE, 1999, pages 688-698. ACM.

PAPAZOGLOU, M.; GEORGAKOPOULOS, D. **Service-oriented computing**. Communications of the ACM: Service-Oriented Computing. 2003.

PIGOSKI, Thomas M. **Practical Software Maintenance: Best Practices for Managing Your Software Investment**. Wiley, 1st edition, Outubro 1996.

PORTAL BPMINSTITUTE. **BPMS on SOA: Still the Exception**. Julho de 2006. Disponível em < http://www.bpminstitute.org/articles/article/article/bpms-on-soa-still-the-exception.html >. Acesso em 28 de setembro de 2007.

PRADO, Antonio; BRAGA, Regina; BRAGA, R. T. **AMGraA: Uma Abordagem para Migração Gradativa de Aplicações Legadas**. 2005. Dissertação (Mestrado em Ciência da Computação) - Universidade Federal de São Carlos.

PULIDO, Javier. **Biztalk Server 2004: la nueva era del 'e-business'.** Revista Perspectivas, Microsoft, 2004. Disponível em:

<http://www.microsoft.com/spain/enterprise/perspectivas/numero_11/n_11_producto.asp>. Acesso em 12 de setembro de 2007.

SOAP W3C. **Simple Object Access Protocol**. Edição 2, Abril, 2007. Disponível em < http://www.w3.org/TR/soap/ >. Acesso em 05 de agosto de 2007.

SOARES, S; BORBA, P. **Progressive implementation with aspect–oriented programming.** 2002. In Verlag, S., editor, The 12th Workshop for PhD Students in Object-Oriented Systems, Malaga, Spain. To appear.

SOFTWARE AG. **Web Services Making the most of your business assets.** Março de 2002.

SOMERVILLE, I. **Software Engineering**. 6. ed. Addison-Wesley, 2004.

STEIN, Dominik. **An Aspect-Oriented Design Model Based on AspectJ and UML.** Dissertação de Mestrado (Mestrado em Gerenciamento de Sistemas de Informação) Universidade de Essen, Germany, 2002.

SUN MICROSYSTEMS. **Object-Oriented Programming Concepts**. Edição 2, Maio, 2005. Disponível em <http://java.sun.com/docs/books/tutorial/java/concepts/ >. Acesso em 01 de julho de 2007.

TARR, P., OSHER, H., HARRISON, W., SUTTON Jr, S. **N Degrees of Separation: Multi-Dimensional Separation of Concerns.** In Proceedings of the 21st International Conference on Software Engineering (ICSE'99), Maio de 1999, pp. 107–119.

VERHEECKE, B.; VANDERPERREN, W.; JONCKERS, V. **Unraveling crosscutting concerns in web services middleware**. IEEE Software, 23(1):42–50, Janeiro 2006.

VERNADAT, F. B., **Enterprise Modeling and Integration: Principles and Applications**. London: Chapman & Hall, 1 ed., 1996.

XML W3C. **Extensible Markup Language (XML) 1.0**, Edição 4, Setembro, 2006. Disponível em <http://www.w3.org/TR/2006/REC-xml-20060816/>. Acesso em 28 de agosto de 2007.

APENDICE A – CÓDIGO FONTE DO CENÁRIO 1

/* Arquivo de Configuração XML do Mule */

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

<!DOCTYPE mule-configuration PUBLIC "-//MuleSource //DTD mule-configuration
XML V1.0//EN""http://mule.mulesource.org/dtds/mule-configuration.dtd">

<mule-configuration id="ProjetoAspMule" version="1.0">

    <connector name="SystemStreamConnector"
    className="org.mule.providers.stream.SystemStreamConnector">
        <properties>
            <property name="promptMessage" value=": "/>
            <property name="messageDelayTime" value="1000"/>
        </properties>
    </connector>

     <transformers>
            <transformer name="HttpRequestToSoapRequest"
            className="org.mule.providers.soap.transformers.HttpRequestToSo
            apRequest"/>
    </transformers>

    <interceptor-stack name="default">
        <interceptor className="org.mule.interceptadorAop.InterceptTempo"/>
        <interceptor
        className="org.mule.interceptadorAop.InterceptExcecao"/>
    </interceptor-stack>

    <!--
        O Mule model incializa e gerencia os seus componentes UMO
    -->

    <model name="comprovante-service">

            <mule-descriptor name="ComprovanteResidenciaUMO"
            implementation="com.mestrado.mule.ComprovanteResidencia">

            <inbound-router>
                <endpoint address="stream://System.in"/>
                        <endpoint
                        address="xfire:http://localhost:8081/services"
                        transformers="HttpRequestToSoapRequest" />
                  <endpoint address="vm://comprovanteresidencia" />
            </inbound-router>

            <outbound-router>
                  <router
                  className="org.mule.routing.outbound.OutboundPassThroughR
                  outer">
                        <endpoint address="stream://System.out"/>
                  </router>
            </outbound-router>

            <interceptor name="default"/>
```

```
        </mule-descriptor>
    </model>
</mule-configuration>
```

/* Serviço Web ComprovanteResidencia */

```
package com.mestrado.mule;

public class ComprovanteResidencia {

      public String getComprovanteRes(String numeroFone) {

             String[] dadosPessoais = new String[7];

            if("9832270011".equals(numeroFone)) {

                  dadosPessoais[0] = "Luís Cardoso Montes";
                  dadosPessoais[1] = "Rua 9";
                  dadosPessoais[2] = "São Carlos";
                  dadosPessoais[3] = "São Paulo";
                  dadosPessoais[4] = "SP";
                  dadosPessoais[5] = "07500-000";

            } else if("9832270012".equals(numeroFone)) {

                  dadosPessoais[0] = "Luiza Maria Sousa";
                  dadosPessoais[1] = "Av. José Ribeiro";
                  dadosPessoais[2] = "Bequimão";
                  dadosPessoais[3] = "São Luís";
                  dadosPessoais[4] = "MA";
                  dadosPessoais[5] = "65000-000";

            } else if("9832270013".equals(numeroFone)) {

                  dadosPessoais[0] = "José Lima Silva Filho";
                  dadosPessoais[1] = "Av. 7";
                  dadosPessoais[2] = "Centro";
                  dadosPessoais[3] = "Rio de Janeiro";
                  dadosPessoais[4] = "RJ";
                  dadosPessoais[5] = "20080-001";

            } else if("9832270014".equals(numeroFone)) {

                  dadosPessoais[0] = "Mariana Bastos Sousa Paiva";
                  dadosPessoais[1] = "Av. Souza Peixoto";
                  dadosPessoais[2] = "Entroncamento";
                  dadosPessoais[3] = "Belem";
                  dadosPessoais[4] = "PA";
                  dadosPessoais[5] = "20080-001";

            } else {

                  dadosPessoais[0] = "";
                  dadosPessoais[1] = "";
                  dadosPessoais[2] = "";
                  dadosPessoais[3] = "";
                  dadosPessoais[4] = "";
                  dadosPessoais[5] = "";
            }
```

```
            return 
dadosPessoais[0]+"#"+dadosPessoais[1]+"#"+dadosPessoais[2]+"#"+dadosPessoai
s[3]+"#"+dadosPessoais[4]+"#"+dadosPessoais[5];
      }
}
```

APENDICE B – CÓDIGO FONTE DO CENÁRIO 2

/* Arquivo de Configuração XML do Mule */

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

<!DOCTYPE mule-configuration PUBLIC "-//MuleSource //DTD mule-configuration
XML V1.0//EN" "http://mule.mulesource.org/dtds/mule-configuration.dtd">

<mule-configuration id="ProjetoAspMule" version="1.0">

    <connector name="SystemStreamConnector"
    className="org.mule.providers.stream.SystemStreamConnector">
        <properties>
            <property name="promptMessage" value=": "/>
            <property name="messageDelayTime" value="1000"/>
        </properties>
    </connector>

     <transformers>
             <transformer name="HttpRequestToSoapRequest"
            className="org.mule.providers.soap.transformers.HttpRequestToSo
            apRequest"/>
    </transformers>

    <interceptor-stack name="default">
            <interceptor
            className="org.mule.interceptadorAop.InterceptLog"/>
            <interceptor
            className="org.mule.interceptadorAop.InterceptDebug"/>
    </interceptor-stack>

    <!--
        O Model Mule inicializa e gerencia seus componentes UMO.
    -->

    <model name="situacao-pessoa-service">

      <mule-descriptor name="SituacaoPessoaUMO"
      implementation="com.mestrado.mule.SituacaoPessoa">

            <inbound-router>
                <endpoint address="stream://System.in"/>
                  <endpoint address="xfire:http://localhost:8081/services"
                  transformers="HttpRequestToSoapRequest" />
                 <endpoint address="vm://situacaopessoa" />
            </inbound-router>

            <outbound-router>

            <router
            className="org.mule.routing.outbound.OutboundPassThroughRouter"
            >
                  <endpoint address="stream://System.out"/>
            </router>
            </outbound-router>

            <interceptor name="default"/>
```

```
        </mule-descriptor>
    </model>
</mule-configuration>
```

/* Cliente MuleClienteNet */

```
package mule.client.mestrado;

import org.mule.extras.client.MuleClient;
import org.mule.umo.UMOMessage;

public class MuleClienteNet {

      public static void main(String args[]) throws Exception  {

            MuleClient client = new MuleClient();

            String[] param = new String[]{"102.221.222.01"};

            UMOMessage result =
                        client.send
                        ("http://www.bancom.br/consultasituacao/soap-
                                   server.php?method=getCpf",
                         param, null);

            System.out.println(result.getPayload());
      }

   }
```

APENDICE C – CÓDIGO DOS INTERCEPTADORES AOP

/* Aspecto InterceptDebugAop */

```
package org.mule.interceptadorAop;

import org.aspectj.lang.*;
import org.aspectj.lang.reflect.*;

public aspect InterceptDebugAop {

      pointcut  pontoDebug() : execution(* InterceptDebug.intercept(..));

      around(): pontoDebug() {

            JoinPoint jp = thisJoinPoint;

            System.out.println("Entrando no método: "+jp);

            System.out.println("Mensagem: "
                        +jp.getSignature().getName());
            System.out.println("da classe:
                        "+jp.getSignature().getDeclaringType().getName());

            System.out.println("Parametros: " );

            Object[ ] args = jp.getArgs();

            String[ ] names =
                  ((CodeSignature)jp.getSignature()).getParameterNames();

            Class[ ] types =
                  ((CodeSignature)jp.getSignature()).getParameterTypes();

            for (int i = 0; i < args.length; i++) {
                  System.out.println("  "  + i + ". " + names[i] +
                              " : " + types[i].getName() +
                              " = " + args[i]);
            }
            System.out.println("Localização no fonte:
                                    "+jp.getSourceLocation());

      }
}
```

/* Classe InterceptDebug */

```
package org.mule.interceptadorAop;

import org.mule.umo.Invocation;
import org.mule.umo.UMOException;
import org.mule.umo.UMOInterceptor;
import org.mule.umo.UMOMessage;


public class InterceptDebug implements UMOInterceptor
```

```
{

    public UMOMessage intercept(Invocation invocation) throws UMOException
    {
        UMOMessage result = invocation.execute();
        return result;
    }
}
```

/* Aspecto InterceptTempoAop */

```
package org.mule.interceptadorAop;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;


public aspect InterceptTempoAop {

      private static Log logger =
            LogFactory.getLog(InterceptTempoAop.class);

      long inicio = 0;
      long fim = 0;

      pointcut  pontoTempo() : execution(* InterceptTempo.intercept(..));

      before(): pontoTempo() {
            inicio = System.currentTimeMillis();
      }

      after(): pontoTempo() {

            long fim = System.currentTimeMillis();
            logger.info("Total de tempo " + ((fim - inicio)/1000.0) + "
             Segundos / " + ((fim - inicio)) + " Milisegundos \n");
      }

}
```

/* Classe InterceptTempo */

```
package org.mule.interceptadorAop;

import org.mule.umo.Invocation;
import org.mule.umo.UMOException;
import org.mule.umo.UMOInterceptor;
import org.mule.umo.UMOMessage;


public class InterceptTempo implements UMOInterceptor
{

    public UMOMessage intercept(Invocation invocation) throws UMOException
    {
        UMOMessage result = invocation.execute();
        return result;
```

```
    }
}
```

/* Aspecto InterceptLogAop */

```
package org.mule.interceptadorAop;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

public aspect InterceptLogAop {

      private static Log logger = LogFactory.getLog(InterceptLogAop.class);

      pointcut greeting() : execution(* InterceptLog.intercept(..));

      before(): greeting() {
            logger.info("LOG - Antes - " +
                  thisJoinPoint.getSignature().toString() + "\n");
      }

      after() returning(): greeting() {
            logger.info("LOG - Depois - " +
                   thisJoinPoint.getSignature().toString() + "\n");
      }
}
```

/* Classe InterceptLog */

```
package org.mule.interceptadorAop;

import org.mule.umo.Invocation;
import org.mule.umo.UMOException;
import org.mule.umo.UMOInterceptor;
import org.mule.umo.UMOMessage;


public class InterceptLog implements UMOInterceptor
{

    public UMOMessage intercept(Invocation invocation) throws UMOException
    {
        UMOMessage result = invocation.execute();
        return result;
    }
}
```

/* Aspecto InterceptExcecaoAop */

```
package org.mule.interceptadorAop;

import org.mule.umo.Invocation;
import org.mule.umo.UMOException;
import org.mule.umo.UMOInterceptor;
import org.mule.umo.UMOMessage;
```

```
public aspect InterceptExcecaoAop {

            pointcut  pontoExcecao() : execution(*
            InterceptExcecao.intercept(..));

            void around(): pontoExcecao() {

                  Object arguments[] = methodInvocation.getArguments();
              String mensagem = ((String)arguments[1]);
              if (mensagem.equals("")){
                  throw new Exception("Parâmetro Nulo!!!");
              }
              return methodInvocation.proceed();

            }

}
```

/* Aspecto InterceptExcecao */

```
package org.mule.interceptadorAop;

import org.mule.umo.Invocation;
import org.mule.umo.UMOException;
import org.mule.umo.UMOInterceptor;
import org.mule.umo.UMOMessage;


public class InterceptExcecao implements UMOInterceptor
{


    public UMOMessage intercept(Invocation invocation) throws UMOException
    {
        UMOMessage result = invocation.execute();
        return result;
    }
}
```