INSTITUTO DE PESQUISAS TECNOLÓGICAS DO ESTADO DE SÃO PAULO

JAVIER DANIEL CASTRO ELIZONDO

Aplicabilidade de MDA em Projetos de Web Services

São Paulo

2007

Javier Daniel Castro Elizondo **Aplicabilidade de MDA em Projetos de Web Services**

JAVIER DANIEL CASTRO ELIZONDO

Aplicabilidade de MDA em Projetos de Web Services

Dissertação apresentada ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo - IPT, para obtenção do título de Mestre em Engenharia de Computação.

Área de concentração: Engenharia de Software

Orientador: Prof. Dr. Jorge Luis Risco Becerra

São Paulo

Fevereiro de 2007

Ficha Catalográfica
Elaborada pelo Departamento de Acervo e Informação Tecnológica – DAIT
do Instituto de Pesquisas Tecnológicas do Estado de São Paulo - IPT

E43a Elizondo, Javier Daniel Castro

Aplicabilidade de MDA em projetos de Web services. / Javier Daniel Castro Elizondo.

São Paulo, 2007.

88p.

Dissertação (Mestrado em Engenharia de Computação) - Instituto de Pesquisas Tecnológicas do Estado de São Paulo. Área de concentração: Engenharia de Software.

Orientador: Prof. Dr. Jorge Luis Risco Becerra

1. Arquitetura de software 2. Model-Driven Architecture - MDA 3. Web services 4. Tese I. Instituto de Pesquisas Tecnológicas do Estado de São Paulo. Coordenadoria de Ensino Tecnológico II. Título

07-78 CDU 004.72(043)

RESUMO

Este trabalho visa demonstrar a aplicação da MDA (*Model-Driven Architecture*) no desenvolvimento de sistemas baseados em *Web Services*.

Os elementos da estratégia para atingir este objetivo são: um processo MDA para *Web Services* (MDA4WS) e um experimento. O processo incorpora o conceito da MDA de pontos de vista, através dos conceitos da recomendação IEEE1471 (*Recommended Practice for Architectural Description of Software-Intensive Systems*), que permitem definir uma arquitetura de software MDA.

O processo MDA4WS é composto por procedimentos, diagramas da UML, conceitos da MDA, conceitos do IEEE1471, conceitos e técnicas de orientação a objetos, resultando em uma estrutura de aplicação prática da MDA em projetos de *Web Services*.

O experimento aplica o processo MDA4WS em um projeto, cujo escopo é um sistema automatizado *e-Procurement* baseado em *Web Services*.

Os resultados do experimento serão avaliados verificando, qualitativamente, se o MDA4WS permite especificar um Web Service.

Palavras chave: Arquitetura de Software, *Model-Driven Architecture*, *Web Services*.

ABSTRACT

This work purpose is to demonstrate the application of MDA (*Model-Driven Architecture*) in the development of Web Services based systems.

To achieve this goal the elements of the strategy are: a MDA process for *Web Services* (MDA4WS) and an experiment. The processes incorporates the MDA points of view through the concepts of the IEEE1471 (*Recommended Practice for Architectural Description of Software-Intensive Systems*) that enable define a MDA software architecture.

The MDA4WS process is composed by procedures, UML diagrams, MDA concepts, IEEE1471 concepts, techniques and concepts of the object orientation, resulting in a MDA practical application structure for Web Services projects.

The experiment applies the MDA4WS process in a Web Services based e-Procurement project.

The results of the experiment will be evaluated qualitatively verifying if the MDA4WS allows specify a Web Services.

Key words: Software Architecture, *Model-Driven Architecture*, *Web Services*.

Agradecimentos

Á Deus por suas bênçãos nesta jornada.

A minha família, meus pais, irmã e avós, pelo apoio e, por sempre acreditarem em mim, em especial, ao meu avô Meli Elizondo, que sempre acreditou no meu sucesso, e que infelizmente não está mais aqui para comemorar esta vitória comigo.

Ao Dr. Jorge Becerra, pela sua orientação no desenvolvimento deste trabalho.

Ao Rubens Sales, pela amizade e conselhos.

Lista de ilustrações

Lista de Tabelas

Lista de Abreviaturas e Siglas

B2B	*Business to Business*
CIM	*Computational Independent Model*
CWS	*Common Warehouse Specification*
DTD	*Document Type Definition*
FTP	*File Transfer Protocol*
HTTP	*HiperText Transfer Protocol*
MDA	*Model-Driven Architecture*
OASIS	*Organization for the Advancement of Structured Information Standards*
OMG	*Object Management Group*
PDA	Processo de Descrição de Arquitetura
PDS	Processo de Desenvolvimento de *Software*
PEP	Ponto de Vista Específico de Plataforma
PIC	Ponto de Vista Independente de Computação
PIM	*Platform Independent Model*
PIP	Ponto de Vista Independente de Plataforma
PSM	*Platform Specific Model*
QoS	*Quality of Service*
RPC	*Remote Procedure Call*
SMTP	*Send Mail Transfer Protocol*
SOC	*Service-Oriented Computing*
UDDI	*Universal Description, Discovery, and Integration*
UML	*Unified Modeling Language*
VEP	Visão Específica de Plataforma
VIC	Visão Independente de Computação
VIP	Visão Independente de Plataforma
W3C	*World Wide Web Consortium*
WS-BPEL	*Business Process Execution Language for Web Services*
WSDL	*Web Service Description Language*
WS-I	*Web Services Interoperability Organization*
XML	*Extensible Markup Language*
XSD	*XML Schema Description*

SUMÁRIO

Capítulo 1. INTRODUÇÃO

Neste capítulo são apresentadas as considerações iniciais, os objetivos, a abrangência, a justificativa e a metodologia usada para esta dissertação intitulada "Aplicabilidade da MDA em projetos de *Web Services*".

1.1. Considerações Iniciais

A identificação da informação e elementos relevantes para o desenvolvimento de sistemas de *software,* é um tema amplo de pesquisa e, na última década, surgiram modelos que propõem grupos de conceitos para certos domínios tecnológicos como o RM-ODP (*Reference Model for Open Distributed Processing*), proposto pela [ISO, 95], que propõe um grupo de elementos necessários para o desenvolvimento de sistemas de computação distribuída ou, a MDA (*Model-Driven Architecture*) que é uma abordagem proposta pela OMG (*Object Management Group*) [OMG, 01][OMG, 03], para o desenvolvimento de sistemas com foco na integração, interoperabilidade e portabilidade de sistemas.

A MDA utiliza um modelo abstrato, denominado modelo independente de plataforma - PIM (*Platform Independent Model*), para especificar o comportamento da aplicação, onde os conceitos de linguagens ou plataforma são irrelevantes. Este modelo, de alto nível de abstração, posteriormente utilizado para criar um novo modelo que expressa os requisitos do sistema em uma plataforma tecnológica específica, denominado modelo específico de plataforma - PSM (*Platform Specific Model*). [OMG, 03 a].

A proposta desta dissertação é a aplicação dos conceitos e elementos propostos pela MDA no processo de desenvolvimento de *software*.

A Computação Orientada a Serviços – SOC (*Service-Oriented Computing,*) é um paradigma de desenvolvimento, que utiliza serviços como elemento fundamental para a construção de soluções tecnologicamente neutras. Este paradigma, promete potencializar a integração e reúso através dos conceitos de serviço que são aplicados durante o projeto, com o fim de garantir o baixo acoplamento, que se traduz em maior potencial de reuso e de manutenção mais fácil. O desenvolvimento de aplicações distribuídas, orientadas a serviços, é inerentemente complexo, devido ao processo de decomposição de um sistema em serviços independentes, com valor de negócio e de alta granularidade por não ser uma tarefa fácil.

Os *Web Services* são tecnologias que permitem expor recursos computacionais, através da Internet, usando linguagens e protocolos padrões, independentes de plataforma. Um de seus maiores benefícios é a interoperabilidade, que facilita a integração entre os sistemas que adotam este grupo de tecnologias.

Os *Web Services* são uma das formas de implementar SOC na Internet.

O desenvolvimento de *Web Services* requer considerações especiais, devido ao uso da Internet, um mecanismo público, inseguro e de baixa confiança para

interações com outros serviços. As tecnologias em que se baseiam os *Web Services* estão em constante evolução gerando demandas de evolução tecnológica.

A utilização da abordagem de MDA, no processo de desenvolvimento de *software,* traz benefícios para o desenvolvimento de *Web Services.* Devido à aplicação do conceito de independência de plataforma, os modelos PIM, podem ser reutilizados na geração do sistema em mais de uma plataforma de implementação.

1.2. Objetivos

O objetivo geral desta dissertação é aplicar os conceitos da MDA no desenvolvimento de sistemas baseados na plataforma de *Web Services*.

O primeiro objetivo é definir como aplicar os conceitos da MDA através da arquitetura de *software*, definindo como deve ser uma descrição de arquitetura de *software* MDA e o processo necessário para sua descrição.

O segundo objetivo é definir um processo de desenvolvimento de *software,* que incorpore a descrição da arquitetura MDA, resultando em um processo de desenvolvimento de *software* MDA, que possa ser aplicado ao domínio tecnológico dos *Web Services*.

O terceiro objetivo é empregar este processo de desenvolvimento MDA para *Web Services,* em um experimento proporcionando um exemplo de aplicação e, a partir deste experimento, analisar os resultados, avaliando a aplicabilidade da MDA para o desenvolvimento de *Web Services*.

1.3. Justificativa

A justificativa para esta dissertação se encontra na ausência de processos de desenvolvimento, para guiar o uso da abordagem da MDA, de modo eficaz, consistente e específico para *Web Services.*

Embora a MDA tenha um conjunto de conceitos, que possibilite a especificação do sistema, a partir de modelos, não oferece um guia para orientar o desenvolvedor a aplicar estes conceitos, e nem um processo com passos bem definidos para serem seguidos [GERVAIS, 02].

Algumas propostas foram desenvolvidas para auxiliar o desenvolvedor no uso da MDA, como a proposta de GERVAIS, que consiste de um processo de desenvolvimento, que combina alguns conceitos do RM-ODP com a MDA [GERVAIS, 02].

De modo semelhante, o trabalho de MACIEL propõe um processo de desenvolvimento, baseado nos pontos de vista do ODP e o utiliza como ferramenta de modelagem com o perfil EDOC (Enterprise Distributed Object Computing Specification).

Cada uma das fases do processo de MACIEL detalha um ponto do ODP. O processo é voltado para o desenvolvimento de sistemas de autoria colaborativa. Neste processo, o modelo PIM é diretamente usado para a geração de código.

No trabalho de MACIEL, a notação usada para o modelo independente de plataforma é o EDOC, isto traz a vantagem de usar uma linguagem mais formal na modelagem se, comparado com a semi-formal UML (*Unified Modeling Language*) básica, porém, traz a desvantagem de que o desenvolvedor deve conhecer o EDOC, um perfil é a técnica de extensão da semântica da UML.

O trabalho de FERREIRA identifica como combinar MDA com o ODP, através de um método de desenvolvimento, que combina a especificação de sistemas distribuídos com o conceito de independência de plataforma e transformações do MDA [FERREIRA, 05], trabalho motivado pelo trabalho de Maciel e Gervais [MACIEL, 04][GERVAIS, 02], que consideram a utilização dos pontos de vista do ODP, em conjunto com o MDA. No trabalho de FERREIRA, a modelagem é realizada usando a linguagem UML e seus diagramas padrões, porém, não é específico para o desenvolvimento de *Web Services*.

O trabalho de Almeida discute como o conceito de serviço pode ajudar no desenvolvimento de sistemas com MDA, o que motiva o seu uso no processo de desenvolvimento de *Web Services* [ALMEIDA, 03a, 03b e 04].

1.4. Abrangência

Esta dissertação usa os seguintes conceitos da MDA: especificação do sistema através dos pontos de vista da MDA, independência de plataforma, processo guiado pelas atividades de modelagem e transformações entre modelos.

O processo proposto por esta dissertação faz uso dos conceitos de SOC para guiar o desenvolvimento de *Web Services*, porém sem ter intenção de ser um processo para SOC.

Quanto ao domínio tecnológico, o processo proposto orienta tanto o desenvolvedor do cliente de um *Web Services* como o fornecedor do Web Service. O cliente do Web Service pode se vincular ao fornecedor de serviço de duas formas, estática ou dinamicamente. Nesta dissertação somente será considerada a forma estática; a forma dinâmica abrange a área de agentes de *software,* que selecionariam dentro de um conjunto de possibilidades, um *Web Service* dinamicamente. Na forma estática, a identificação do serviço a ser vinculada é realizada em tempo de desenvolvimento, e não em tempo de execução como na forma dinâmica.

O processo será exemplificado, através de um experimento de desenvolvimento, em um projeto para um sistema e-Procurement (fornecimento eletrônico). Este projeto não visa ser uma especificação completa para construção, focando apenas na descrição da arquitetura.

1.5. Metodologia de Pesquisa

O seguinte método pesquisa foi utilizado para o desenvolvimento desta dissertação:

- Pesquisa bibliográfica inicial – pesquisa seletiva e analítica do material relacionado ao tema da dissertação: MDA e engenharia de *software.* O material selecionado forma a base de conhecimento do trabalho, e ocorreu durante todo o período do trabalho, com o fim de manter atualizado o conhecimento.
- Definição do tema – definição do escopo, através das conclusões obtidas durante análise do material pesquisado.
- Definição do processo MDA, através da:
  - Definição de uma descrição de arquitetura MDA
  - Definição de um processo de descrição de arquitetura MDA
  - Definição de um processo de desenvolvimento de *software* MDA aplicado a *Web Services*, que inclui o processo de descrição de arquitetura MDA.
- Experimentação – o processo foi instanciado e aplicado ao desenvolvimento de um experimento com o fim de avaliar de resultados.
- Análise dos resultados – a partir do experimento, foram analisados os resultados e identificados dificuldades e pontos de melhorias.

Capítulo 2. FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta a base teórica desta dissertação: Conceitos de Arquitetura de *Software*, MDA - *Model Driven Architecture, SOC – Service-Oriented Computing* e a *Web Services*.

2.1. Conceitos de Arquitetura de Software

2.1.1. Arquitetura

Segundo o SEI – *Software* Engineering Institute, não há uma definição precisa e única para arquitetura de *software*, entre várias definições aceitas, temos a definição clássica de arquitetura de [SHAW, 96], arquitetura de *software* define o sistema em termos de componentes e os relacionamentos entre si, de modo semelhante [BASS, 03], diz que arquitetura são estruturas compostas de componentes e relacionamentos, com propriedades externas, representando uma abstração do sistema. A definição que consta no padrão IEEE 1471 [IEEE, 00] amplia as definições anteriores, definindo que a arquitetura é a organização fundamental de um sistema, composta por componentes, seus relacionamentos entre si e, o ambiente e os princípios básicos que guiam seu projeto e sua evolução. Para [JAZAYERI, 00], a arquitetura de *software,* é uma ferramenta para lidar com a complexidade do *software,* que deve satisfazer os requisitos funcionais e não-funcionais do sistema.

A partir das definições anteriores, pode-se notar que, arquitetura de *software* é mais que uma simples descrição da estrutura do sistema, em termos de componentes. Segundo [CLEMENTS, 02], é a divisão prudente do sistema em partes, com relacionamentos específicos, com o fim de permitir o desenvolvimento, através de grupos, de forma a trabalharem cooperativamente, para resolver um problema maior que poderiam ser de forma individual.

Segundo [PRESSMAN, 02], a arquitetura permite ao engenheiro de *software,* analisar como o sistema atenderá os requisitos capturados, permitindo antecipar a avaliação de alternativas, antes da construção do sistema e, permite identificar e reduzir os riscos associados com a construção do *software*.

Um sistema pode ser dividido, simultaneamente, de diferentes formas. A documentação da arquitetura de *software* indica ao desenvolvedor como executar seu trabalho.

2.1.2. A importância da arquitetura

Segundo [BASS, 03], a arquitetura é importante porque ela é:

1. Meio de comunicação entre os participantes. A arquitetura de *software* representa uma abstração comum do sistema, para os participantes do

projeto, e serve de base para o entendimento, negociação, consenso e comunicação. Isto deve ser realizado através de um documento, utilizando uma notação que possa ser entendida facilmente.

2. Antecipação das decisões de projeto. A arquitetura de *software* serve de base para o entendimento do sistema, e de referência para as demais atividades de desenvolvimento, registrando as decisões de projeto que foram antecipadas, adicionalmente, preocupando-se com tempo de desenvolvimento, custo e manutenção, definição das restrições de implementação e definição da estrutura organizacional, dando ênfase aos requisitos de qualidade do sistema.
3. Abstração reusável do sistema. A arquitetura de *software* se constitui de um conjunto pequeno de modelos que descreve como o sistema é estruturado e como estes elementos trabalham juntos, podendo haver situações em que, este conjunto pode ser reaproveitado na descrição de outros sistemas, que exibam atributos de qualidade e requisitos funcionais semelhantes. O reúso abrange a reutilização de idéias, como por exemplo, estilos ou padrões de arquitetura, ou reuso de componentes de *software*.

2.1.3. Documentação de arquitetura

A documentação de arquitetura de um sistema não é a penas um documento de especificação. Segundo [CLEMENTS, 02], a documentação de arquitetura possui três usos: educacional, permite que novos membros entendam o projeto; veiculo de comunicação entre os envolvidos do projeto como clientes, fornecedores e desenvolvedores; serve de base para executar uma avaliação da arquitetura do sistema.

A prática de descrever uma arquitetura, usando múltiplos pontos de vista, é um consenso na comunidade de engenharia de *software*. Um ponto de vista é uma técnica de abstração, que usa um conjunto de conceitos arquiteturais e regras de estruturação, com foco em certos interesses, dentro de um sistema.

A definição de um ponto de vista, segundo os conceitos do IEEE1471 [IEEE, 00], está baseada nos conceitos de envolvidos e interesses. Um envolvido é qualquer pessoa ou organização, que é afetada direta ou indiretamente, pelo sistema. Um interesse é qualquer aspecto relacionado à operação do sistema.

Para definir um ponto de vista, segundo os conceitos do IEEE1471, deve-se: identificar quais são os envolvidos, aos quais se destinam as visões resultantes e, identificar quais são os interesses e preocupações na arquitetura do sistema, a Figura 2-1, mostra os conceitos do IEEE1471.

Uma documentação de arquitetura é composta por visões criadas a partir dos pontos de vista selecionados e, cada visão modela algum aspecto do sistema [BASS, 03]. Uma visão é uma representação através de diagramas e texto de um conjunto de elementos.

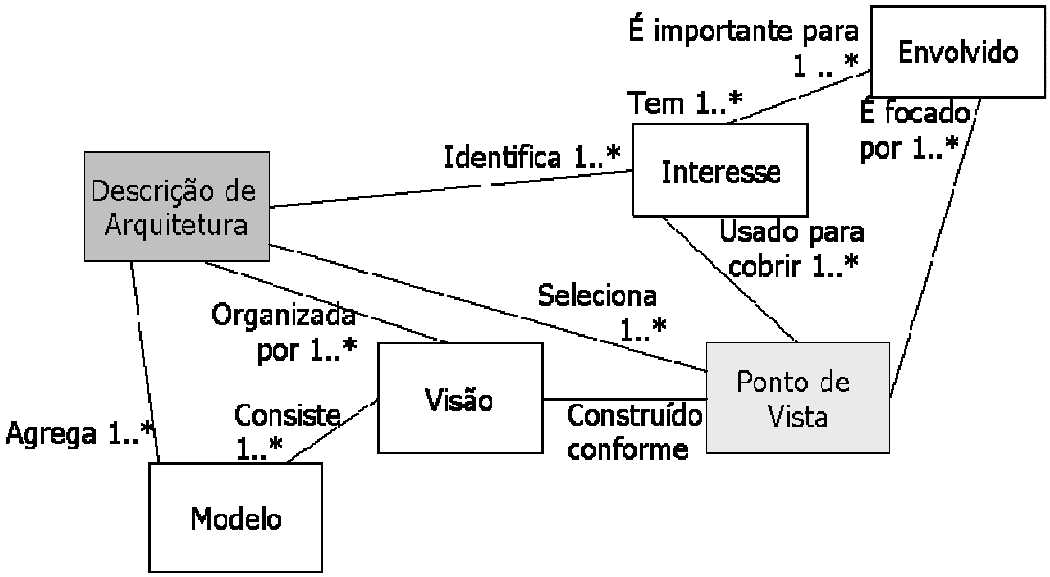

Figura 2-1: Modelo conceitual parcial do IEEE 1471 [IEEE, 00]

2.1.4. Arquiteto de *Software*

O arquiteto de *software* é o responsável pela definição e documentação da arquitetura. Segundo [BASS, 03], as responsabilidades de um arquiteto de *software* são:

- Participar na elaboração do modelo de negócio para o sistema, contribuindo para analisar o custo, o tempo de desenvolvimento e, interfaces com outros sistemas, para que a arquitetura possa alcançar os objetivos do negócio.
- Contribuir para o entendimento dos requisitos, a fim de determinar os requisitos arquiteturalmente significativos.
- Criar ou selecionar uma arquitetura, através da identificação dos componentes e suas interações; identificação das dependências de construção; e seleção das tecnologias que compõem a plataforma de desenvolvimento do sistema.
- Produzir a descrição de arquitetura de *software*, que captura as decisões de *design* mais importantes em várias visões de arquitetura.
- Manter a integridade arquitetural do sistema, durante o processo de desenvolvimento, avaliando todas as mudanças nos elementos de arquitetura significativos, descritos no Documento de Arquitetura de *Software*.

2.1.5. Processo de Arquitetura e o Processo de Desenvolvimento de *Software*

O processo de definição e documentação de arquitetura, é denominado de processo de arquitetura, que consiste de etapas necessárias para produzir e documentar uma arquitetura de *software*. Segundo [BASS, 03], este processo tem como atividades:

- entendimento do modelo de negócio: o arquiteto analisa o sistema, sob diferentes aspectos de como o sistema influencia ou é influenciado pelo negócio e o ambiente, levando em consideração, restrições ao seu desenvolvimento como custo, prazo de desenvolvimento e dependências com outros sistemas, de modo que a arquitetura possa atender aos objetivos do negócio;
- entendimento dos requisitos: o arquiteto participa das atividades de elicitação de requisitos, com a finalidade de entender os requisitos impostos ao sistema, objetivando priorizar a construção das partes mais criticas do sistema;
- criação ou seleção de uma arquitetura: o arquiteto, a partir dos requisitos arquiteturais, cria ou seleciona uma arquitetura de *software,* para o sistema, dentre várias possibilidades;
- representação da arquitetura: o arquiteto deve descrever a arquitetura criada ou selecionada, para que os envolvidos na construção do sistema, possam acompanhar e realizar o desenvolvimento;
- análise ou avaliação da arquitetura: em uma situação, em que pode haver mais de uma possível arquitetura, que atenda as necessidades do cliente, há necessidade de se utilizar um método para fazer a escolha;
- implementação da arquitetura: os desenvolvedores constroem o sistema, seguindo as informações registradas na descrição da arquitetura;
- verificação de conformidade: após a criação da arquitetura ela entra em manutenção e nesta fase o arquiteto deve verificar que a arquitetura do sistema está refletida na sua descrição.

Tradicionalmente o desenvolvimento da arquitetura, em um processo de desenvolvimento de *software* cascata, ocorre após a análise dos requisitos e, antes do projeto de *software*. Esta abordagem traz a desvantagens ao desenvolvimento de grandes projetos, ou projetos onde não é possível definir todos os requisitos de uma só vez, porque a definição da arquitetura fica postergada, já que os requisitos arquiteturalmente significativos, requisitos que podem colocar em risco o sistema, são descobertos tardiamente.

Os modelos de processo de desenvolvimento de *software,* que usam a arquitetura de *software* com artefato primário para conceitualizar, construir, gerenciar e evoluir o sistema, são denominados de processos centrados em arquitetura. Nestes processos, a arquitetura de *software* é desenvolvida e mantida iterativamente por todo o processo. O Processo Unificado (Unified Process), proposto em [JACOBSON, 99], é um exemplo desta categoria de processos.

2.1.6. Estilo arquitetural

Um estilo arquitetural, descreve uma categoria de sistema que estabelece [PRESSMAN, 02]:

- um conjunto de componentes que executam funções do sistema;
- um conjunto de conectores, que permitem a comunicação, coordenação e cooperação entre os componentes;
- restrições que definem como componentes, podem ser integrados para formar um sistema;
- modelos semânticos que permitem ao arquiteto compreender as propriedades gerais de um sistema, analisando as propriedades das partes.

Exemplos de estilos arquiteturais são canos e filtros (*pipe-and-filter*), cliente servidor (*client-server*), chamada de retorno (*remove procedure call*) e, orientado a eventos (*event-driven*).

2.2. MDA

A MDA (*Model Driven Architecture*), é uma abordagem de desenvolvimento de *software,* guiada por modelos proposta pela OMG (*Object Management Group*), em [OMG, 01] e [OMG, 03a], com o objetivo de solucionar problemas de portabilidade, interoperabilidade e reusabilidade. A MDA, se baseia na prática estabelecida de separar a especificação de operação do sistema e dos detalhes de como este sistema usa os recursos de sua plataforma tecnológica.

Em [OMG, 01], o termo plataforma é usado para se referir aos detalhes tecnológicos e de engenharia, que são irrelevantes para descrever a funcionalidade de um componente de *software*. Segundo [OMG, 03a], plataforma é um conjunto de subsistemas e tecnologias, que fornecem um conjunto coerente de funcionalidades, através de interfaces e padrões específicos de uso, que qualquer aplicação suportada poderá usar, sem preocupação com os detalhes de como as funcionalidades da mesma serão implementadas.

Em um processo de desenvolvimento, baseado em MDA, os modelos produzidos guiam o curso do entendimento (*model-driven*), projeto, construção, instalação, operação e manutenção do sistema. Cada atividade do processo, requer como insumo, modelos que gerarão outros modelos, através de transformações. O processo pode ser visto como um conjunto de transformações de modelos e informação que levará ao sistema final. O sistema pode ser visto então, através destes modelos que o descrevem, bem como, o domínio da aplicação e seus requisitos, através de diferentes visões e níveis de abstrações.

2.2.1. Objetivos primários

Os três objetivos da MDA, para o desenvolvimento de sistemas são:

- Portabilidade - habilidade de uma implementação ser transferida de um ambiente para outro. Na MDA, este conceito é tratado através da portabilidade da especificação, obtida através da separação de interesses na especificação do sistema, permitindo assim, independência do modelo PIM de elementos de plataforma permitindo que esta parte da especificação do sistema, possa ser transformada em várias plataformas [OMG, 03 a].
- Interoperabilidade - capacidade de dois ou mais sistemas interagirem trocando informação. Na MDA, este conceito é atendido através da geração de pontes (gateways) de comunicação, entre os elementos de plataformas distintas [OMG, 03 a].
- Reusabilidade - capacidade de reconhecer partes comuns, entre diversos domínios em diferentes níveis de abstração (de negócio a código). Na MDA, o conceito é previsto através de uma biblioteca de reusáveis, elementos comuns que [OMG, 01] descreve nas seguintes categorias: Serviços Pervasivos (Serviços de nomes e diretório, serviços de notificação e tratamento de eventos), Especificações de domínio (padronização de serviços e recursos em áreas de negócio especificas como finanças, telecomunicações e saúde), Transparências e Qualidade de Serviço (modelos de ambientes com atributos específicos de hardware e *software* como escalabilidade, tempo-real e tolerância à falhas) [OMG, 03 b]. Esta biblioteca não está completa atualmente.

2.2.2. Independência de plataforma

O conceito de independência de plataforma é chave na MDA. A Independência de Plataforma, é uma qualidade que um modelo apresenta ao não possuir dependências com os recursos de uma plataforma em particular. Uma conseqüência da independência de plataforma, é a capacidade de realização do modelo PIM, em diferentes plataformas. Idealmente, um PIM deveria ser neutro a qualquer tecnologia, porém, durante a trajetória de desenvolvimento, certos conceitos independentes de plataforma somente se aplicam a um grupo de plataformas [ALMEIDA, 04]. Um exemplo desse conceito é o mecanismo de distribuição.

Os Requisitos de Qualidade de Serviço - QoS (*Quality of Service*) devem ser satisfeitos por mecanismos de QoS implementados na aplicação ou na plataforma de destino. Idealmente os desenvolvedores se beneficiam de transparências de distribuição, quando estas complexidades são tratadas mais pelo middleware, afastando a complexidade da aplicação.

A transparência de distribuição é a propriedade de mascarar das aplicações, detalhes e diferenças entre os mecanismos usados para lidar com os problemas causados pela distribuição. Este requisito surge da necessidade de facilitar a construção de aplicações distribuídas. Os aspectos de distribuição devem ser

mascarados (total ou parcialmente), incluindo, heterogeneidade de suporte de *software* e hardware, localização e mobilidade de componentes, mecanismos para alcançar o nível requerido de QoS, face às falhas (por exemplo, replicação, migração, checkpoint), [ISO, 95].

No nível de independência de plataforma, estas transparências se aplicam ao que se denomina plataforma abstrata. A escolha de uma plataforma abstrata, define quais propriedades ou aspectos (independentes de plataforma) serão considerados no PIM e, quais propriedades ou aspectos (específicos de plataforma) serão abstraídos da modelagem PIM, [ALMEIDA, 04]. Uma característica de uma plataforma abstrata é o estilo arquitetural.

Com a finalidade de definir uma plataforma abstrata, deve-se considerar [ALMEIDA, 04]:

- Requisitos de portabilidade para o PIM - a definição deve levar em consideração a portabilidade dos requisitos, a plataforma abstrata deve ser genérica o suficiente para mapear para diferentes plataformas.
- Necessidades do desenvolvedor - a plataforma abstrata facilita a tarefa modelagem do PIM, através de transparências adotadas no PIM.
- A extensão de quanto a plataforma abstrata e a concreta diferem. A distância entre as abstrações de plataforma abstrata e concreta tem conseqüências diretas nos mapeamentos, entre a plataforma independente e a específica.

2.2.3. Pontos de vista da MDA

A abordagem, conforme especificada no MDA, Guide Version 1.0.1 [OMG, 03a], especifica sob quais pontos de vista o sistema a ser desenvolvido deve ser modelado.

Os pontos de vista propostos em [OMG, 03a] são:

- Ponto de Vista Independente de Computação - foca no ambiente do sistema e os requisitos para o sistema.
- Ponto de Vista Independente de Plataforma – foca a operação do sistema, ocultando os detalhes da plataforma de implementação.
- Ponto de Vista Específico de Plataforma – combina o ponto de vista independente de plataforma com os detalhes de uso da plataforma, pelo sistema.

2.2.4. Modelos da MDA

O [OMG, 03a], especifica quais modelos de uma visão devem ser produzidos, um modelo de uma visão é a representação do todo sistema, a partir da perspectiva de um ponto de vista, escolhido através de diagramas e texto. Os modelos especificados em [OMG, 03a], são:

- Modelo Independente de Computação – CIM (*Computation Independent Model*) – Este modelo é uma vista do sistema, a partir do ponto de vista independente de computação, os requisitos do sistema são descritos ocultando detalhes de estrutura e processamento da aplicação que, neste momento, possivelmente, ainda não foram determinados. Ainda que a MDA não especifique como ocorre à transformação do CIM para o PIM, o relacionamento entre os dois deve ser mapeado, com a finalidade de estabelecer a rastreabilidade entre os requisitos e os elementos do modelo PIM.
- Modelo Independente de Plataforma - PIM – Este modelo é uma vista do sistema a partir do ponto de vista independente de plataforma. O PIM especifica a estrutura, funcionalidade e o comportamento de um sistema, abstraindo os detalhes tecnológicos da plataforma de execução do sistema. O modelo PIM, atende aos requisitos de negócio, permanecendo estável, enquanto a tecnologia evolui, entendendo a vida da aplicação.
- Modelo Específico de Plataforma - PSM – Este modelo é uma vista do sistema a partir do ponto de vista especifico de plataforma. O PSM combina a especificação do PIM, com detalhes que especificam como o sistema usa um tipo particular de plataforma.

2.2.5. Transformação

A transformação entre modelos na MDA, é o processo de converter um modelo em outro modelo do mesmo sistema. Esta transformação é realizada a partir de definições formais denominadas mapeamentos. Um mapeamento, é uma descrição de como um ou mais elementos na linguagem de origem podem ser transformados em um ou mais elementos na linguagem de destino [OMG, 01]. Os mapeamentos podem ser descritos, usando linguagem natural, algoritmos ou linguagem específica para transformação.

A Figura 2-2, a seguir, mostra a transformação do modelo PIM para o PSM.

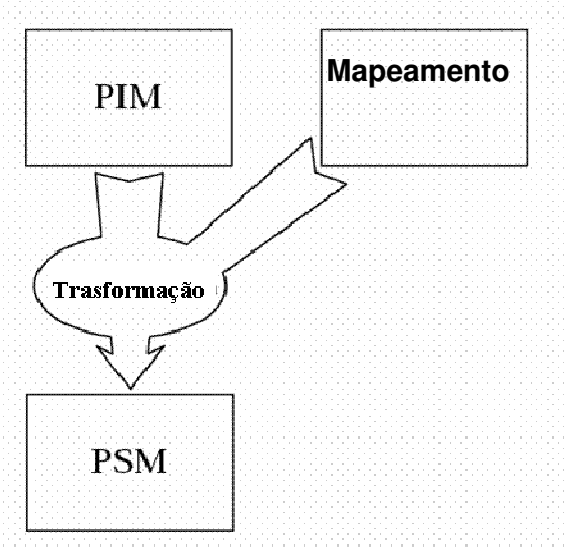

Figura 2-2: Transformação de modelos [OMG, 03a]

As transformações podem ser tipificadas em elaborativa ou direta. Na transformação direta, o modelo PIM é transformado diretamente em código [MELLOR, 02] [MELLOR, 04]. Na transformação elaborativa, ocorrem transformações sucessivas e incrementais do PIM para PSM e PSM em código, e, a cada transformação, vão se adicionado detalhes de plataforma até chegar ao código. A Figura 2.3, resume os conceitos descritos neste item sobre as transformações na estrutura MDA.

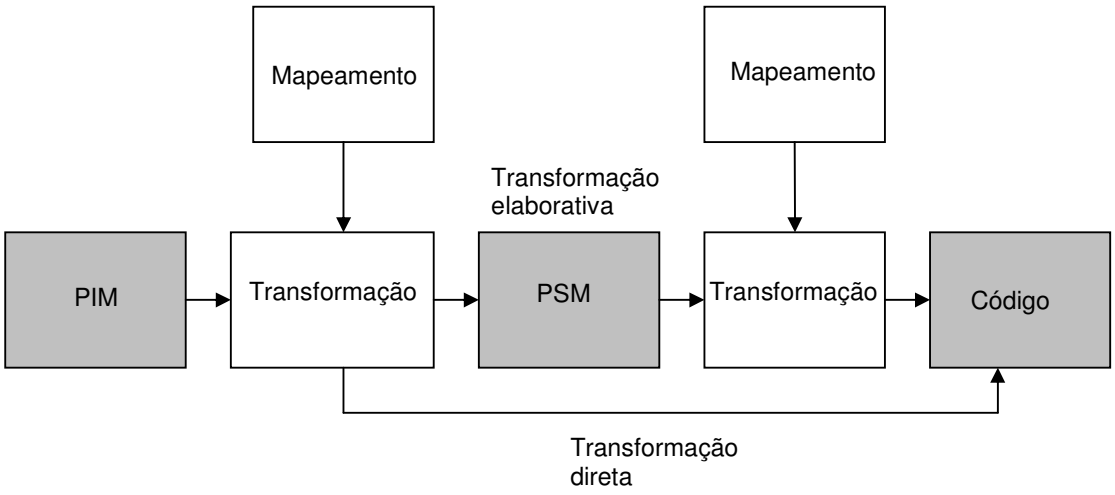

Figura 2-3: Tipos de transformação [KLEPPE, 03].

2.2.6. Base tecnológica MDA

A especificação MDA especifica um grupo de tecnologias, que foram adotadas para padronizar sua implementação [OMG, 03 a]:

- UML (*Unified Modeling Language*) - Linguagem de modelagem utilizada para visualizar, documentar e especificar sistemas de *software*.
- MOF (*Meta-Object Facility*) - Tecnologia que fornece um repositório de meta-modelos, que pode ser usada para especificar e manipular modelos.
- Modelos MOF – Modelos específicos para linguagens ou plataformas, descrevem o que pode ser expresso em um modelo válido de linguagem de modelagem. Exemplos de modelos MOF: CWM (*Common Warehouse Specification*) - grupo de meta-modelos para linguagens utilizadas na modelagem de Data Warehouse como SQL, XML e OLAP.
- Perfil da UML (UML *Profile*) - Técnica de extensão ao UML, que permite criar uma linguagem de modelagem específica adicionando ou retirando elementos da linguagem UML.
- XMI (*XML-based Metadata Interchange*). O objetivo do XMI é o intercâmbio de modelos de forma serializável baseado em XML (*Extensible Markup Language*).

2.3. Computação Orientada a Serviços – SOC

A Computação Orientada a Serviços – SOC (*Service-Oriented Computing*) é um paradigma de computação, que utiliza serviços como elementos fundamentais para o desenvolvimento de aplicações [PAPAZOGLOU, 03a]. Os serviços, são elementos auto descritivos, independentes de plataforma que permitem às organizações expor funcionalidades de negócio, geralmente, através da Internet, usando linguagens e protocolos baseados em padrões abertos [PAPAZOGLOU, 03a].

Os serviços podem ser classificados como simples ou compostos, sob a visão de reúso. Os serviços compostos são aqueles que internamente, acessam e combinam informação e funções de outros serviços internos na empresa ou de outros fornecedores de serviços.

O reuso, é possível sempre que o serviço esteja disponível e, que ele seja gerenciado para manter um QoS requerido.

2.3.1. Princípios de orientação a serviços

Os serviços desenvolvidos, segundo a SOC, possuem algumas características chave a seguir relacionadas:

- Possuem baixo acoplamento – os clientes dos serviços, não necessitam conhecer nenhuma das estruturas internas do mesmo [PAPAZOGLOU, 03B] [HUHNS, 05].
- Compartilham contratos formais – Serviços definem contratos formais de uso, que descrevem e definem o serviço, em termos da informação trocada entre cliente e serviço [HUHNS, 05] [ERL, 05].
- São Reusáveis – Serviços são reusáveis, independentemente, de existirem oportunidades imediatas ou não de reuso [ERL, 05].
- Podem participar de composições – Serviços podem ser compostos de outros serviços, isto permite que a lógica seja dividida em diferentes níveis de granularidade e promovendo a reusabilidade e criação de camadas de abstração [HUHNS, 05] [ERL, 05].
- Abstraem a lógica interna – A única parte visível, externamente, de um serviço, é seu contrato de uso. Toda a lógica por traz do contrato é irrelevante para o cliente do serviço [ERL, 05].
- São autônomos – Dentro da fronteira de um serviço reside a lógica governada pelo serviço de forma independente de outros serviços [ERL, 05].
- Não possuem estado (*stateless*) – Serviços não devem gerenciar informação de estado para manter baixo acoplamento. Deve-se delegar o controle de estado a outros mecanismos [ERL, 05]. O tempo de duração do estado, deve ser longo o suficiente para detectar exceções e tomar ações corretivas.
- São possíveis de descobrir – Serviços devem permitir que, suas descrições de serviço sejam descobertas e entendidas pelos clientes de serviço, para que se possa fazer uso de sua lógica [PAPAZOGLOU, 03B][ERL, 05].
- São tecnologicamente neutros – os serviços são invocados através do menor denominador comum de tecnologia, disponível entre os ambientes dos sistemas de informação. Estes mecanismos (protocolos, descrições e mecanismos de descoberta), devem ser compatíveis com padrões amplamente usados [PAPAZOGLOU, 03B].

2.4. Web Services

O termo *Web Services* é usado para descrever tanto aplicações modulares, auto descritivas acessíveis através da Internet como também um conjunto de especificações de linguagens e protocolos independentes de plataforma usados para intercambio de dados entre aplicações na Internet.

Os *Web Services* são usados para integração de aplicações distribuídas e autônomas na Internet e pode ser usada para construir serviços baseados nos princípios de orientação a serviços. Os *Web Services* podem ser de dois tipos: Simples ou Compostos.

Os *Web Services* ajudam a integrar aplicações que originalmente não foram projetadas para se integrar com outras aplicações distribuídas e permitindo criar novas funcionalidades e ao mesmo tempo integrando as funcionalidades existentes na aplicação.

2.4.1. Colaboração entre *Web Services*

Os papéis que participam de uma colaboração entre *Web Services* são três: o fornecedor de serviço, o consumidor de serviço e o registro de serviços. Os participantes interagem com os seguintes artefatos de serviço: a descrição de serviço e a implementação do serviço.

Os serviços são oferecidos pelas entidades denominadas fornecedores de serviços – organizações que fornecem as implementações de serviço e, as respectivas descrições de serviço.

Os fornecedores de serviço devem, adicionalmente, dar o suporte relacionado com negócio e a infra-estrutura necessária para a integração e colaboração entre os serviços, com os sistemas internos e os sistemas de outras empresas.

Os clientes de serviço são entidades que podem ser outras soluções ou aplicações internas da empresa, ou externas a elas, como clientes ou fornecedores. O consumidor conhece, a funcionalidade de um serviço, a partir da descrição disponibilizada pelo fornecedor. Para recuperar os detalhes, o consumidor realiza uma pesquisa em um Registro de Serviços, por exemplo, um registro UDDI (*Universal Description, Discovery, and Integration*), onde o fornecedor publicou a descrição do serviço, e através do mecanismo de vinculação o cliente chama o serviço do fornecedor.

O registro de serviços é a localização central onde o fornecedor pode inscrever seus serviços, e no qual um consumidor pode pesquisar serviços. Os fornecedores publicam os serviços no registro para que os consumidores possam pesquisar e, em seguida, vincularem-se.

Na Figura 2-4, a seguir, são identificadas três operações que são fundamentais para o funcionamento dos *Web Services* – pesquisar, vincular e publicar. A comunicação entre as aplicações não deve considerar: o tipo de linguagem na qual a aplicação foi escrita, a plataforma onde a aplicação está sendo executada e o protocolo de transporte. Também estão presentes na figura, dois artefatos básicos: a implementação do Serviço e a correspondente descrição do Serviço.

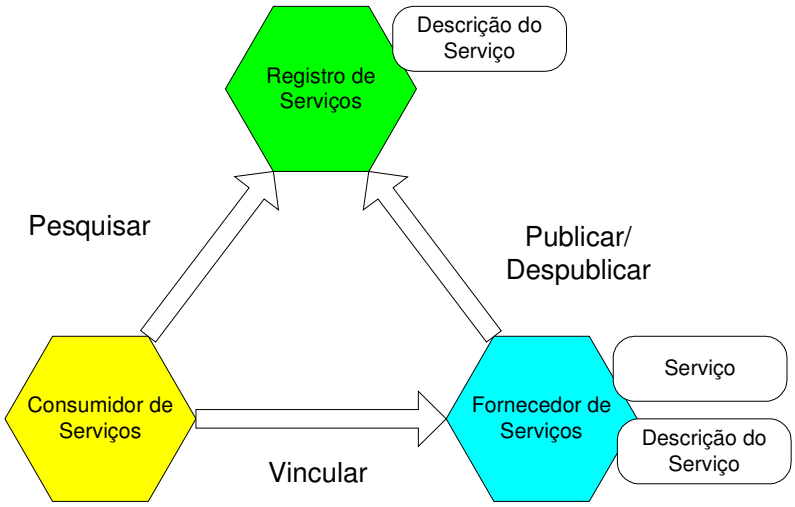

Figura 2-4: Colaboração entre *Web Services*.

As descrições de serviço informam o contrato do serviço (ponto de acesso, operações e todas as mensagens de entrada e saída de cada operação), e, podem descrever o seu propósito conceitual e a qualidade esperada. A descrição da QoS, fornece informação sobre atributos qualitativos como custo, forma de tarifação, métricas de desempenho (tempo de resposta, por exemplo) e segurança.

Tecnologias básicas

O conjunto de tecnologias definido pelo WS-I (*Web Services* Interoperability Organization), um consórcio industrial que promove a interoperabilidade dos *Web Services*, que fazem parte do perfil básico para interoperabilidade entre *Web Services*, denominado de WS-I versão 1.1, é composto por [WS-I, 04]: SOAP 1.1, WSDL 1.1, UDDI 2.0, XML 1.0 v2 e XML Schema partes 1 e 2, HTTP 1.1.

Todas essas tecnologias, com exceção do protocolo HTTP, são tecnologias baseadas na linguagem XML (*Extensible Markup Language*). O XML é uma linguagem para descrever a informação de forma estruturada e auto-descritiva através de marcas (*tags*). A sintaxe de um documento XML é definida, através de XML Schemas, que define as *tags* que um documento XML pode conter.

O SOAP (*Simple Object Access Protocol*), é protocolo de comunicação de serviço baseado em XML, usado para trocar informações entre aplicações, independentemente, do sistema operacional, da linguagem de programação ou do modelo do objeto.

O WSDL (Web Service Description Language), é uma linguagem baseada em XML para descrever *Web Services*. Um documento WSDL, define a interface do *Web Services*, os métodos que estão presentes, os parâmetros de entrada e saída para cada um dos métodos, os tipos de dados, o protocolo de transporte usado e a URL da extremidade onde o *Web Services* será hospedado.

O UDDI (*Universal Description, Discovery, and Integration*), é um padrão baseado XML, que permite aos fornecedores de serviços publicar detalhes sobre os

Web Services fornecidos e suas empresas em um registro central. Também fornece um padrão, para permitir que os consumidores de serviços localizem fornecedores de serviços e, detalhes sobre seus *Web Services*.

A estrutura do conjunto de tecnologias que habilitam *Web Services* é exibida na Figura 2.5, esta figura foi adaptada da referência [ENDREI, 04], onde as camadas e respectivas tecnologias são detalhadas a seguir.

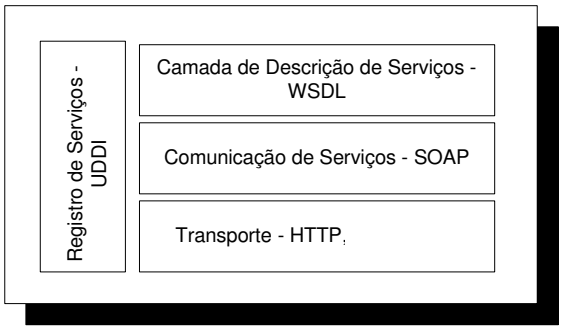

Figura 2-5: Estrutura de protocolos de Web Services [ENDREI, 04]

Transporte

Nesta camada, estão os protocolos responsáveis pela disponibilização dos serviços, tornando-os acessíveis, por intermédio de algum dos protocolos de transporte usado na Internet, [W3C, 00]. Os *Web Services,* são disponibilizados independentes do modo de transporte. O HTTP é o protocolo de comunicação mais amplamente utilizado e, o selecionado pelo WS-I para compatibilidade.

Protocolo de Comunicação de Serviço

Essa camada, também chamada de nível de mensagem, define o formato de mensagem usado na comunicação entre aplicações. O padrão usado com regularidade pelos *Web Services* é o SOAP. Este protocolo, é composto de duas partes: Envelope e *Attachment* (anexos). O elemento Envelope – é usado para descrever o conteúdo da mensagem e fornece informação de como processá-lo. É composto pelos elementos: *Header* (cabeçalho) e *Body* (corpo).

O SOAP, possui as seguintes características [W3C, 00]:

- O SOAP é um protocolo simples, não tenta definir um modelo de programação ou implementar APIs específicas, fornecendo um mecanismo, que duas partes usam para envolver o que esta sendo solicitado e retornado.

- O SOAP, pode ser vinculado a outros protocolos de transporte, atém do HTTP.
- O SOAP, é facilmente extensível por meio da XML. Pode ser ampliado para incorporar recursos empresariais, como transações, segurança e outros.

A Figura 2.6, exibe os elementos do protocolo SOAP.

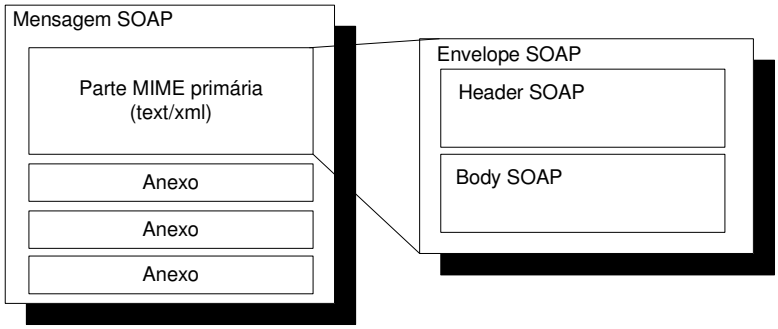

Figura 2-6: Estrutura SOAP.

Há dois atributos específicos que indicam o modo como uma mensagem SOAP é estruturada e representada: o atributo de estilo que estabelece o formato das mensagens no corpo de uma mensagem SOAP, e o atributo de uso que especifica o sistema de tipos a ser usado para representar os dados da mensagem.

O atributo de estilo pode ter dois valores: documento ou RPC (Remote Procedure Call). O primeiro incorpora documentos XML no corpo do SOAP, enquanto o segundo declara a estrutura na forma de parâmetros como na comunicação síncrona RPC.

O atributo de uso pode ter dois valores: literal ou codificado. O SOAP, originalmente, fornecia seu próprio sistema de tipos para representar o conteúdo do corpo, posteriormente, foi incorporado o suporte para XML Schemas, o atributo indica qual sistema de tipos será usado. O valor literal indica que será usado o sistema de tipos do XML Schema.

Descrição do serviço

Esta camada define o formato das descrições de serviço. Um documento WSDL descreve um Web Service como uma coleção de pontos de acesso aos serviços denominados de *endpoits* (extremidades) ou portas, independentemente, do estilo, mensagens orientadas a documento ou orientadas a RPC. A Figura 2.7, mostra a estrutura de WSDL. A especificação apresenta a definição de um Web Service, em duas partes. A primeira parte, representa uma definição abstrata

independente do protocolo de transporte, enquanto a segunda parte, representa uma descrição específica com detalhes do protocolo de transporte.

Os elementos do WSDL são [W3C, 01a]:

- ***portType*** - Os elementos *portType* contêm um conjunto de **Operações** representadas como elementos *operation*, que estão presentes em um Web Service. É análogo a uma interface em Java. Uma operação pode ter uma mensagem de entrada e uma mensagem de saída. Além disso, ela pode ter apenas uma mensagem de entrada ou saída, da mesma maneira que uma chamada de método normal.
- ***message*** - Um elemento *message,* contém uma definição dos dados a serem transmitidos. É semelhante ao parâmetro na chamada do método.
- ***types*** - O elemento *types,* contém os tipos de dados que estão presentes na mensagem.
- ***binding*** - O elemento *binding* (vínculos), mapeia os elementos *operation* de um elemento *portType*, para um protocolo de transporte específico.
- ***Service, po*rt** – Um *service*, contém uma ou mais de portas (*port*) - cada porta está associada a um elemento *portType,* através de um elemento *binding*, que descreve as mensagens aceitas para esta porta.

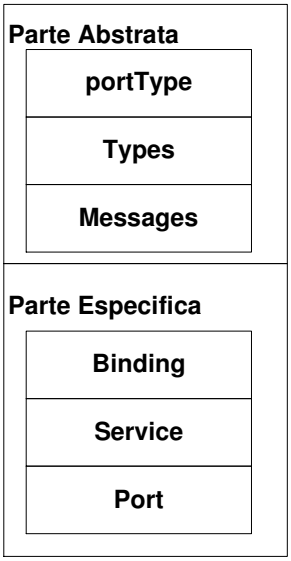

Figura 2-7: Elementos WSDL.

O WSDL, na versão 1.1, possui quatro primitivas de transmissão que um *endpoint* pode suportar [W3C, 01a], porém, o WS-I somente recomenda os dois primeiros:

- Unidirecional – o *endpoint* apenas recebe uma mensagem.

- Recebe-Resposta – o *endpoint* recebe uma mensagem e envia uma mensagem correlacionada.
- Solicita-Resposta – o *endpoint* solicita uma mensagem e recebe uma mensagem correlacionada.
- Notificação – o endpoint apenas envia uma mensagem.

Registro de Serviços

A partir de um documento WSDL, um consumidor de serviço pode determinar os detalhes do *Web Service*, como operações, tipos de dados, extremidades, protocolos de vínculo e outras informações. Em lugar de publicar o documento WSDL, para cada possível cliente, um fornecedor de serviços pode publicar as informações sobre seu *Web Service* em um registro central, que esteja disponível, publicamente, para os consumidores de serviço interessados. Além de publicar a descrição de seu *Web Service*, pode publicar também informações relacionadas ao negócio e complementares importantes para a busca e seleção como QoS. Os consumidores de serviços, podem realizar uma pesquisa de nível mais amplo, com base em categorias comerciais, para encontrar as empresas nessas categorias e, a seguir, pesquisar por detalhes adicionais sobre os *Web Services,* por elas oferecidos.

Estas necessidades, levaram a elaboração da especificação para a publicação e descoberta de informações comerciais em registros, UDDI (*Universal Description, Discovery, and Integration*) [OASIS, 02].

2.4.2. Especificações de segunda geração

O primeiro conjunto de especificações de tecnologias de *Web Services* (SOAP, WSDL, UDDI, HTTP, XML e XML Schema), não dá suporte a funcionalidades complexas como o gerenciamento de contexto, necessário para a coordenação de atividades envolvendo dois ou mais serviços. A seguir, são relacionadas algumas das chamadas especificações de segunda geração ou extensões de *Web Services*. Este conjunto de tecnologias, ainda não faz parte do perfil básico do WS-I e, fornece funcionalidades específicas, como coordenação entre *Web Services*, tratamento para transações atômicas e transações atômicas de longa duração.

WS-Coordination

A especificação [MICROSOFT, 05 a], descreve um *framework* extensível para a definição de protocolos que coordenam ações de aplicações distribuídas. Estes protocolos de coordenação são usados no suporte de aplicações que, por exemplo, necessitem chegar a um acordo sobre o resultado de ações distribuídas.

O *framework*, definido nesta especificação, permite que um serviço de uma aplicação crie um contexto necessário para propagar uma atividade para outros serviços e, registre protocolos de coordenação. Este framework, permite ocultar o

processamento de transações, workflow e outros sistemas de seus protocolos proprietários, permitindo aos participantes operar em um ambiente heterogêneo. Adicionalmente, esta especificação descreve a estrutura do contexto compartilhado e, os requisitos para propagá-lo entre os serviços participantes.

Este framework, define três elementos:

- Coordenador de Contexto – representa o contexto compartilhado, que é propagado entre os participantes distribuídos.
- Serviço de ativação - representa o serviço usado pelos clientes para criar um contexto coordenado.
- Serviço de registro – representa o serviço usado pelos participantes, para a incluir de protocolos de coordenação específicos

WS-AtomicTransaction

A especificação [MICROSOFT, 05 b], descreve um tipo de coordenação para transações atômicas que estende o *framework*, definido pela especificação WS-Coordination. A especificação define três protocolos de coordenação: encerramento (completion), commit de duas fase volátil (volatile two-phase commit), commit de duas fase durável (durable two-phase commit). Estes protocolos, são usados pelos desenvolvedores no desenvolvimento de aplicações que, requerem entrar em acordo sobre o resultado de atividades de curta duração do tipo “tudo ou nada”.

WS-BusinessActivity

A especificação [MICROSOFT, 05 c], descreve um tipo de coordenação para atividades de negócio que estende o *framework,* definido pela especificação WS-Coordination. A especificação define dois protocolos: BusinessAgreementWithParticipantCompletion e BusinessAgreementWithCoordinatorCompletion. Estes protocolos, são usados em aplicações que, requerem entrar de acordo sobre o resultado de atividades distribuídas de longa duração.

WS-BPEL

O WS-BPEL (*Web Services* - Busines Process Execution Language), é uma linguagem XML especificada em [IBM, 03] [OASIS, 05], usada para descrever processos de negócios automatizados, possui uma notação e uma semântica para execução de processos de negócio, dando suporte, tanto para Coreografia quanto para Orquestração, em ambos os casos, a linguagem de especificação fornece recursos para indicar a lógica do processo, como a seqüência ou paralelismo de atividades e condições de falhas.

A Orquestração, se refere à execução de grupo de atividades que um processo implementa, privadamente. A coordenação entre execução do fluxo de atividades e mensagens trocadas, entre os serviços que fazem parte da lógica do processo e são controlados de modo centralizado [PELTZ, 03], de modo semelhante a um *workflow*.

A Coreografia se refere ao comportamento publico e observável entre *Web Services*, não foca na definição da execução de processos de negócio como a Orquestração [PELTZ, 03], e, sua visão de um processo corresponde a uma troca de mensagem entre participantes, sem fornecer detalhes internos de sua implementação, especificando de forma abstrata.

Utilizando WS-BPEL, os processos de negócio podem ser descritos usando duas alternativas, segundo o nível de abstração [IBM, 03]:

- Executável – comportamento real de um participante em uma interação de negócio, definindo os detalhes internos de um processo de negócio em termos de lógica e estados. Um processo de negócio executável define: as interações entre *Web Services,* que suportam o processo de negócio e seu ambiente.
- Protocolo de negócio – especifica o comportamento visível da troca de mensagens entre processos de negócio, sem revelar seus detalhes internos, um protocolo de negócio também é conhecido como processo abstrato. Um processo abstrato, define apenas os aspectos públicos de processos de negócio e seu ambiente, não especificando como as decisões são tomadas internamente.

O BPEL, não fornece suporte para a decomposição, logo, não pode especificar um processo de negócio em termos de uma composição de outros processos de negócio, usando WS-BPEL.

A especificação do BPEL, foi desenvolvida pela empresas: Microsoft, IBM, Siebel, BEA e SAP e, foi submetida para organização de padrões OASIS (*Organization for the Advancement of Structured Information Standard*) [PELTZ, 03] que atualizou o nome para WS-BPEL.

WS-Security

As seguintes características de segurança devem ser consideradas em um projeto de *Web Services*:

- Identificação – o receptor de uma mensagem, deve ser capaz de identificar o remetente.
- Autenticação – o receptor de uma mensagem, deve ser capaz de verificar que, a identidade do remetente é válida.
- Autorização – o receptor de uma mensagem, deve ser capaz de determinar o nível de acesso do remetente, determinando quais operações ou dados, o remetente terá acesso garantido.
- Integridade – o receptor é capaz de determinar que a mensagem não foi alterada durante a transmissão, até o seu recebimento.
- Confidencialidade – o conteúdo da mensagem não pode ser visto durante o seu transito, exceto pelos serviços autorizados.

A especificação WS-Security [OASIS, 04], descreve extensões para mensagens SOAP, que fornecem proteção através de integridade e

confidencialidade. Estes mecanismos, podem ser usados para acomodar vários modelos de segurança e tecnologias de criptografia. Também, fornece um mecanismo genérico para associar *tokens* (ficha) de segurança, com mensagens. A especificação foi projetada para ser extensível, suportando múltiplos formatos de *tokens*.

O *XML-Encryption* [W3C, 01 b], é uma tecnologia usada no *WS-Securitiy,* para assegurar a confidencialidade de mensagens SOAP ou de partes específicas da mensagem.

O *XML-Signature* [W3C, 02], é uma tecnologia usada no *WS-Securitiy* para autenticar e, dar integridade a mensagens SOAP, anexando uma assinatura digital que está vinculada ao conteúdo do documento.

2.5. Suporte dos Web Services aos princípios SOC

O trabalho de [ERL, 05], analisa quais os princípios da computação orientada a serviços – SOC, são suportados pelos *Web Services*, conforme a Tabela 2-1, a seguir.

Como se pode observar, o suporte nativo dos *Web Services* aos princípios SOC, é parcial e, requer que os seguintes princípios sejam tratados durante o seu desenvolvimento:

- Reusabilidade de serviço
- Autonomia de serviço
- Independência de estado de serviço
- Descobrimento de serviço

Sendo que, os dois primeiros são tratados durante a fase de análise do sistema e, os demais, durante a fase de projeto.

Tabela 2-1: Principio SOC e seu suporte em *Web Services* [ERL, 05]

Princípio SOC	Suporte *Web Services*
Reusabilidade do serviço	*Web services,* não são automaticamente reusáveis. Este princípio, está relacionado com a natureza da lógica encapsulada e, exposta através dos *Web Services*.
Contrato de serviço	*Web Services,* são construídos a partir descrições de serviço, isto faz parte fundamental dos *Web Services*.
Baixo Acoplamento	*Web Services,* naturalmente, possuem baixo acoplamento através das descrições de serviço.
Abstrações de Serviço	*Web Services,* ocultam todos os detalhes da lógica interna do serviço.
Composição de serviços	*Web Services,* são naturalmente componíveis, porém, isto deve ser determinado durante o projeto do serviço e, em função da reusabilidade da lógica representada.
Autonomia de serviço	A lógica gerenciada pelo serviço, existe dentro de uma fronteira, mas, o projeto deve assegurar-se que o serviço é proprietário de seu processamento, isto não é, automaticamente, fornecido por *Web Services*.
Serviços sem estado	A independência de estado, é uma consideração de projeto, não é automaticamente fornecido por *Web Services*.
Descobrimento de serviços	O descobrimento de serviços, deve ser definido na arquitetura, isto não é nativamente suportado por *Web Services*.
Tecnologicamente neutros	As tecnologias utilizadas pelos *Web Services,* são baseadas em padrões independentes de plataforma.

2.6. Conclusão do Capítulo.

A MDA não define como seus conceitos podem ser aplicados no processo de desenvolvimento de *software*. O próximo capítulo propõe uma solução através da arquitetura de *software,* como um modo de aplicação de conceitos no processo de desenvolvimento. Este processo possui como foco, o desenvolvimento de *Web Services*.

Capítulo 3. PROCESSO DE DESENVOLVIMENTO DE *SOFTWARE* MDA

Com base nos conceitos apresentados, anteriormente, será definido e apresentado a seguir, o processo proposto por este trabalho, um processo de desenvolvimento de *software,* baseado na MDA.

3.1. Introdução

A abordagem MDA, conforme especificada na mais recente especificação da MDA [OMG, 03a] - MDA Guide Version 1.0.1, propõe que um sistema deve ser especificado sob três pontos de vista (independente de computação, independente de plataforma e específico de plataforma), de modo a separar os interesses relevantes para integração, interoperabilidade e portabilidade.

Tanto [OMG, 01] [OMG, 03a], especificam o que é a MDA e os seus conceitos, mas não especificam como aplicá-los em um processo de desenvolvimento de *software* e, não detalham, quais elementos de um sistema devem ser considerados e modelados, a partir dos pontos de vista de vista propostos.

O restante deste capítulo, propõe respostas para estas questões, inicialmente, descrevendo quais são os conceitos usados para resolvê-las e, posteriormente, definindo uma forma de aplicá-los no processo de desenvolvimento de *software,* através da arquitetura de *software*.

3.2. Modelo Conceitual do Processo MDA

Este tópico define quais são os conceitos fundamentais adotados e como eles estão relacionados entre si, para alcançar o objetivo central deste capítulo, que é definir uma forma de usar os conceitos da MDA, para o desenvolvimento de *Web Services*.

A premissa inicial deste trabalho é a incorporação dos conceitos aplicados na arquitetura do processo de desenvolvimento de *software* (PDS).

O processo de descrição de arquitetura de *software* (PDA) MDA, proposto, especifica a arquitetura de *software,* através de visões criadas a partir dos pontos de vista da MDA, usando os conceitos para definição de um ponto de vista definidos pela recomendação IEEE1471 [IEEE, 00].

O PDA MDA será incorporado em um PDS, com um ciclo de vida definido para o desenvolvimento de sistemas, em uma plataforma tecnológica, determinada pelo objetivo desta dissertação – *Web Services*. O PDS, resultante desta combinação, será denominado doravante de MDA para *Web Services* (MDA4WS), no inglês a pronuncia da palavra “quatro” é semelhante à palavra “para”.

A Figura 3-1, representa o modelo conceitual proposto.

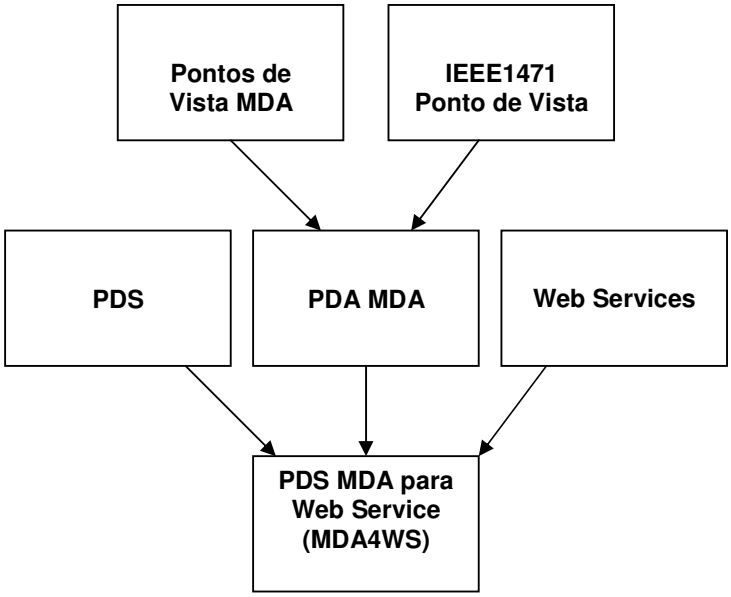

Figura 3-1: Modelo Conceitual do PDS para WS - MDA4WS.

3.3. Processo de Descrição de Arquitetura de Software MDA

3.3.1. Detalhamento dos pontos de vista da MDA segundo o IEEE1471

Este tópico detalha qual a informação necessária para descrever uma arquitetura de *software,* baseado nos conceitos da MDA e do IEEE1471. A MDA especifica que, um sistema deve ser representado sob os seguintes pontos de vista: independente de computação (PIC), independente de plataforma (PIP) e específico de plataforma (PEP), [OMG, 03a].

A especificação [OMG, 03a], somente declara em alto nível, qual a informação que cada ponto de vista deve visualizar, mas, permite concluir que: o PIC foca nos requisitos do sistema e do seu ambiente, descrevendo o sistema no domínio do problema; o PIP foca na operação do sistema, no domínio da solução, de modo independente de plataforma; o PEP foca em uma solução específica de plataforma, dentre várias possibilidades que pertencem ao domínio da solução.

A Figura 3-2, mostra a representação do sistema através dos pontos de vista.

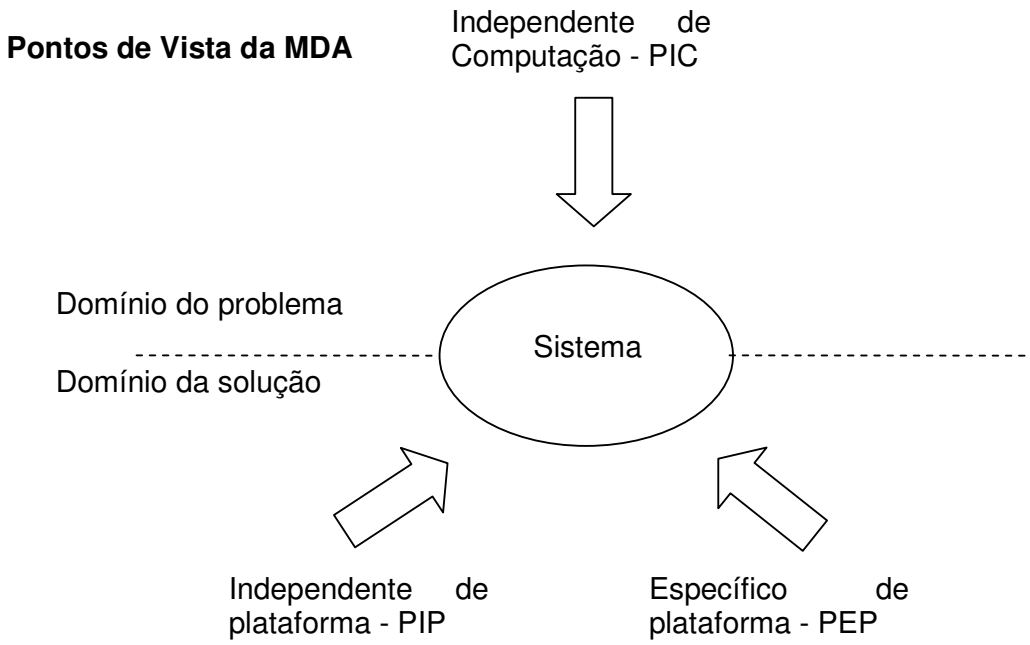

Figura 3-2: Pontos de vista da MDA e a especificação do sistema.

Os conceitos do IEEE1471, usados para detalhar os ponto de vista da MDA, são envolvidos (*stakeholders*) e interesses (*concerns)*.

Os envolvidos considerados que possuem interesses na arquitetura de *software* são:

- Analista de sistema – que a usam para organizar os requisitos e, entender as restrições tecnológicas e riscos.
- Usuários finais ou clientes – que a usam para visualizar em alto nível o que estão comprando.
- Projetistas – que a usam para compreender os princípios básicos e limites na parte designada do projeto.
- Arquitetos – que a usam para considerar sobre a evolução ou reúso.

A seguir, serão determinados os envolvidos focados e seus interesses cobertos por cada ponto de vista da MDA.

Ponto de Vista Independente de Computação – PIC

O objetivo deste ponto de vista, é capturar os requisitos e o ambiente ao redor do sistema [OMG, 03a]. Os requisitos são fundamentais no PDS e sua definição é de interesse do cliente, para determinar o que está comprando, dos usuários do sistema, para determinar como o sistema lhes servirá, assim como os envolvidos na construção do sistema que necessitam saber como o sistema funcionará.

Envolvidos

- Todos os envolvidos de especificação dos requisitos: cliente e usuários finais.
- Todos os envolvidos na construção do sistema: arquiteto de *software*, analistas de sistemas e desenvolvedores.

Interesses

Determinar:

- Qual o objetivo e escopo do sistema.
- Quem são os atores do sistema (usuários e sistemas externos).
- Quais as funcionalidades que o sistema oferecerá.
- Qual a informação usada pelo sistema e, qual o seu significado.
- Quais os requisitos não funcionais para o sistema.

Ponto de Vista Independente de Plataforma – PIP

O objetivo deste ponto de vista é descrever o comportamento e a organização das funcionalidades do sistema, com um grau de independência da plataforma determinado [OMG, 03a]. A análise do comportamento e sua especificação em termos de elementos independentes de plataforma, é tarefa dos analistas de sistema, que devem validar o comportamento com os especificadores dos requisitos, por parte do cliente.

Envolvidos

- Todos os envolvidos de especificação dos requisitos: cliente e usuários finais.
- Todos os envolvidos na construção do sistema: arquiteto de *software*, analistas de sistemas e desenvolvedores.

Interesses

Determinar:

- Qual o estilo arquitetural a ser usado no sistema.
- Como as funcionalidades serão decompostas em elementos independentes de plataforma (subsistemas, interfaces, classes), segundo o estilo arquitetural.
- Como o sistema realiza as operações em termos de elementos independentes de plataforma a resposta a um evento.
- Como os elementos independentes de plataforma estão organizados

Ponto de Vista Específico de Plataforma – PEP

O objetivo deste ponto de vista é descrever como os componentes do sistema fazem uso da plataforma selecionada. Uma visão especificada, a partir deste ponto de vista, indica como os elementos do sistema na plataforma selecionada devem interagir, de forma a implementar o comportamento e funcionalidades especificados na visão anterior, adicionalmente, deve ser indicado como os componentes serão

organizados e empacotados, e, finalmente, como estes pacotes serão distribuídos nos elementos de processamento.

Envolvidos

- Todos os envolvidos na construção do sistema: arquiteto de *software*, analistas de sistemas e desenvolvedores.

Interesses

Determinar:

- Quais são os elementos de plataforma (classes e componentes)
- Como os elementos de plataforma realizam as funcionalidades.
- Como o sistema está estruturado lógica e fisicamente (camadas, subsistemas e componentes).
- Quais os nós envolvidos no processamento e sua capacidade de processamento.
- Quais os tipos e características de comunicação entre os nós.
- Como os componentes estão distribuídos nos elementos de processamento.
- Quais padrões de construção o sistema deve atender

3.3.2. Descrição de Arquitetura MDA

A descrição de arquitetura MDA, proposta neste tópico, visualiza a arquitetura de *software,* a partir dos três pontos de vistas detalhados no item anterior, sendo uma visão para cada ponto de vista. Os nomes dados a estas visões são: VIC – Visão Independente de Computação (resultante do PIC), VIP – Visão Independente de Plataforma (resultante do PIP) e VEP – Visão Específica de Plataforma (resultante do PEP).

As visões da descrição de arquitetura MDA, somente focam nos requisitos arquiteturalmente significativos, isto é, requisitos que representam: risco para o desenvolvimento do sistema; alguma funcionalidade central e significativa que possuem cobertura arquitetural substancial; ou que enfatiza ou ilustra um determinado ponto complexo da arquitetura.

A identificação dos requisitos arquiteturais é influenciada tanto pelo ambiente técnico como pela experiência do arquiteto [SHENG, 03]. Os requisitos arquiteturalmente significativos derivam de uma de três fontes: requisitos não funcionais qualitativos do sistema, objetivos de negócio para o sistema ou objetivos de negócio das pessoas que usarão sistema [SHENG, 03].

A identificação dos requisitos arquiteturais, pode ser respondida através de critérios, desenvolvidos para um domínio específico, uma vez que esta resposta é em função do domínio, porém, isto está além do escopo desta dissertação.

A seguir, está proposta a descrição de arquitetura MDA, suas visões, diagramas e informação a ser identificada. A linguagem usada nos diagramas das visões é UML.

VIC - Visão Independente de Computação

- Descrição de Objetivos e Escopo – declara qual o problema a ser resolvido e, a extensão dos limites da solução proposta em formato texto.
- Diagrama de Casos de Uso – identifica: atores, casos de uso arquiteturalmente significativos em um diagrama de casos de uso UML e descreve, brevemente, os casos de uso em forma de texto.
- Diagrama de Objetos de Negócio – documenta as entidades e a informação usada pelo sistema, representada como classes, atributos e relacionamentos usando um diagrama de classe UML.
- Diagrama de ciclo de vida de Objetos de Negócio – documenta regras, envolvendo as possíveis mudanças de estado de um ou mais objetos de negócio, usando um diagrama de estados UML.
- Relação de requisitos especiais – identificação de requisitos não funcionais (disponibilidade, acessibilidade, integridade, desempenho, confiabilidade, conformidade com normas e segurança, entre outros), em formato de texto.

VIP – Visão Independente de Plataforma

- Diagrama de Estrutura – mostra a estrutura estática dos elementos independentes de plataforma (subsistemas, interfaces e classes), que implementam o estilo arquitetural, selecionado das funcionalidades decompostas. Este modelo usa um diagrama de classes UML.
- Diagrama de Comportamento – mostra como os elementos independentes de plataforma interagem, em resposta a um evento, usando um diagrama de seqüência ou iteração UML.
- Diagrama de Estados – documenta regras, envolvendo as possíveis mudanças de estado de um ou mais classes, usando um diagrama de estados UML.
- Diagrama de Pacotes – documenta como os casos de uso do sistema estão organizados e divididos em subsistemas, usando um diagrama de pacotes UML.

VEP – Visão Específica de Plataforma

- Descrição de plataforma - Descrição textual da plataforma de destino do sistema.
- Diagrama de Camadas – definir como a lógica da aplicação será dividida em grupos coerentes e padronizados, denominados camadas, nas quais, serão alocados e organizados os componentes. Diagrama de pacotes com divisão de camadas.
- Diagrama de Componentes - registra como elementos de implementação estão estruturados, organizados e as dependências entre eles, usando um diagrama de componentes UML.
- Diagrama de Pacotes – mostra como os elementos físicos estão organizados, usando um diagrama de pacotes UML.
- Diagrama de Implantação - mostra a configuração dos elementos de processamento (nós) e, como estão conectados entre si, usando um diagrama de implantação UML.
- Guia de Desenvolvimento - Definir quais os padrões que devem ser seguidos para o desenvolvimento e implementação.

- Diagrama de Estrutura – mostra a estrutura estática dos elementos da plataforma e suas associações, usando um diagrama de classes UML.
- Diagrama de Comportamento – mostra como os elementos de implementação interagem em resposta a um evento usando um diagrama de seqüência/iteração UML.

3.3.3. Processo de descrição arquitetura MDA

O objetivo do PDA MDA é produzir e documentar uma arquitetura de *software,* para o sistema segundo os conceitos da MDA. A arquitetura, evolui conforme cresce o entendimento do sistema durante o todo o PDA, cujas fases são mostradas na Figura 3-3.

A partir da descrição de arquitetura definida anteriormente, observa-se que, a informação necessária, se origina através de três visões sequencialmente, isto é, a visão VIC é necessária para criar a visão VIP, e a visão VIP é necessária para criar a visão VEP.

A visão VIC, resulta do PDA da compreensão e análise do problema, portanto, a primeira fase: Requisitos Arquiteturais, foca na identificação dos requisitos arquiteturalmente significativos, delimitando o que se projetará nas fases subseqüentes, contendo apenas o que foi considerado relevante para a construção da arquitetura de *software*.

A fase Análise Arquitetural Independente de Plataforma, analisa como as funcionalidades são decompostas e organizadas.

A fase Projeto Arquitetural Específico de Plataforma, projeta uma solução de arquitetura de *software* na plataforma de construção do sistema.

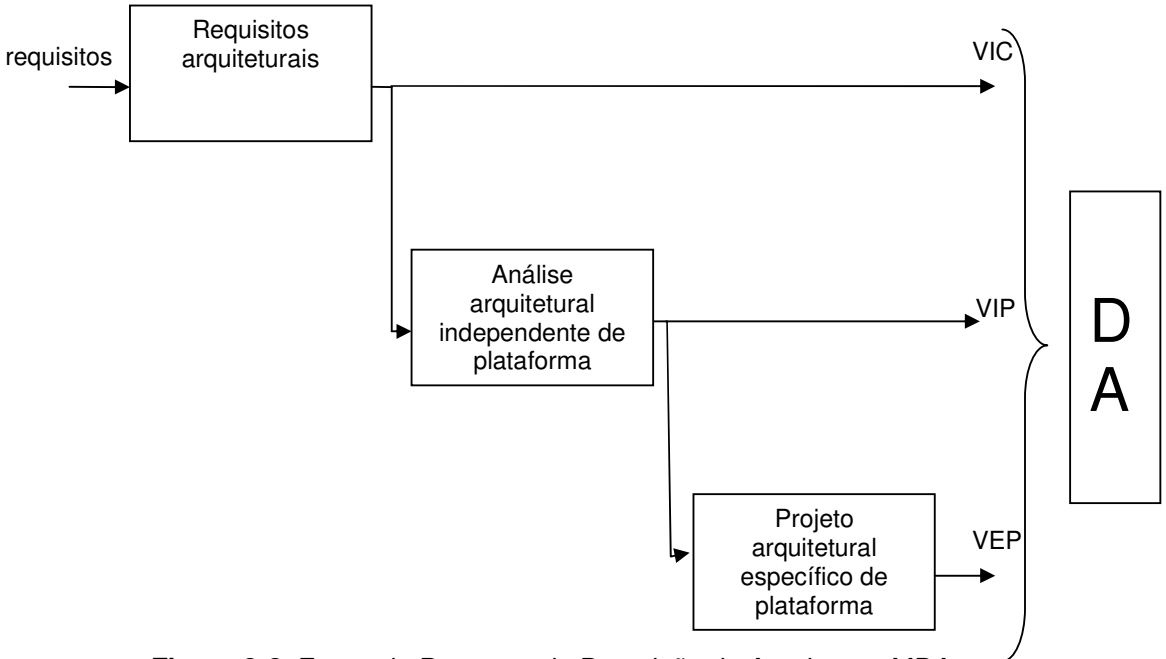

Figura 3-3: Fases do Processo de Descrição de Arquitetura MDA.

Requisitos arquiteturais

O objetivo desta fase é identificar o ambiente, o escopo, as funcionalidades arquiteturalmente significativas e a informação usada por elas. A informação identificada é registrada na VIC da descrição de arquitetura.

As atividades a serem executadas nesta fase são seguintes:

- **Determinar ambiente e escopo -** O ambiente de operação determina o contexto em que o sistema será usado, além de outras influências sobre o sistema. O ambiente pode incluir outros sistemas com o qual ele deverá interagir, isto ajuda a determinar as fronteiras que definem o escopo do mesmo. Um sistema existe para preencher uma ou mais missões neste ambiente, como meio para alcançar um conjunto de objetivos esperados por um ou mais usuários ou clientes.
- **Determinar requisitos arquiteturalmente significativos -** Os requisitos funcionais do sistema são identificados a partir das necessidades dos atores do sistema, devendo ser determinados quais deles serão arquiteturalmente significativos, para fazer parte da descrição de arquitetura. Adicionalmente, devem ser identificados os requisitos não-funcionais como: restrições, segurança, disponibilidade, tempo de resposta, capacidade, entre outras características que influenciarão o projeto do sistema.
- **Identificar objetos de negócio -** Devem ser identificados os objetos de negócio, relacionados com os requisitos identificados. Os objetos de negócio, representam informação com que o negócio lida: seus atributos e relacionamentos. Alguns objetos de negócio podem se comportar de modo diferente, em função do seu estado, podendo seu comportamento ser representado como máquinas de estado.

Análise arquitetural independente de plataforma

O objetivo desta fase é descrever o comportamento e a organização das funcionalidades arquiteturalmente significativas do sistema, com um certo grau de independência da plataforma de implementação. As definições são registradas na VIP da descrição de arquitetura.

O grau de independência de plataforma de uma visão, a partir deste ponto de vista, está restrito ao estilo arquitetural selecionado, como os propostos por [BUSCHMANN, 96], não há como garantir portabilidade entre diferentes padrões arquiteturais, como *Remote Procedure Call* e o *Pipes* e filtros. O sistema é decomposto nos tipos de elementos definidos pelo estilo arquitetural.

As atividades a serem executadas nesta fase são seguintes:

- **Definir estilo arquitetural –** Esta fase é iniciada com a seleção de um estilo arquitetural, esta decisão inicial é crítica, porque influência todo o restante do projeto arquitetural. Esta decisão, é baseada na compreensão das propriedades do padrão e do problema a ser solucionado.
- **Identificar e Decompor as funcionalidades –** A seguir, são identificadas as funcionalidades e, estas são decompostas em elementos independentes de plataforma (subsistemas, componentes e classes), segundo o estilo arquitetural selecionado. As interações entre os componentes, são modeladas

de forma a realizar as funcionalidades, conforme os requisitos especificados na fase anterior.

- **Organizar funcionalidades** – A organização das funcionalidades decompostas nesta visão, tem impacto no desenvolvimento e, portanto, são arquiteturalmente significativas. As funcionalidades são organizadas em pacotes e, as dependências entre os pacotes são analisadas.

Projeto arquitetural específico de plataforma

O objetivo desta fase é selecionar uma plataforma de execução do sistema, organizar como o sistema será construído em termos de camadas, definir como o sistema implementará, em termos de elementos específicos de plataforma (subsistemas físicos e componentes), as funcionalidades analisadas na fase anterior e, definir como o sistema será implantado. As definições são registradas na VEP da descrição de arquitetura.

- **Selecionar plataforma -** O primeiro passo desta fase é selecionar a plataforma de destino do sistema, responsável pelo processamento do sistema. Nesta fase devem ser identificadas alternativas de solução e deve ser definido um critério de seleção. O critério de seleção deve levar em consideração os requisitos não-funcionais e, as restrições identificadas na atividade anterior, assim como custo, risco, complexidade da tecnologia, entre outras restrições de projeto.
- **Definir e organizar implementação -** O objetivo desta atividade é identificar os elementos específicos de plataforma, definindo a alocação dos elementos no projeto. A alocação dos componentes do sistema é representada sob a visão do desenvolvedor, através de diagramas de componentes, pacotes e camadas. Os elementos são organizados em subsistemas e componentes, permitindo organizar o desenvolvimento ou o modelo de projeto, assim como, as interfaces entre os subsistemas e, a dependência entre eles. Adicionalmente, é criada uma guia, contento as diretrizes de desenvolvimento.
- **Definir implantação** – Devem ser identificados os elementos de processamento e a configuração de rede entre eles, devido ao impacto que isto tem na arquitetura de *software*. A alocação dos componentes do sistema é representada, sob a visão do implantador, através do diagrama de implantação.

3.3.4. Processo de descrição de arquitetura MDA no processo de desenvolvimento de Software

O objetivo deste tópico é apresentar como o PDA MDA se insere e se relaciona com um PDS.

Um PDS centrado em arquitetura modela o sistema através dos mesmos pontos de vista usados na descrição de arquitetura (PIC, PIP e PEP), porém, estes pontos de vista originam visões que recebem o nome de modelos (CIM, PIM e PSM

respectivamente), segundo a especificação da MDA. Os modelos MDA, são representações completas do sistema ao contrário das visões de arquitetura VIC, VIP e VEP, que focam apenas no que é arquiteturalmente significativo, conforme mostra a Figura 3-4. Os modelos incluem as respectivas visões da arquitetura.

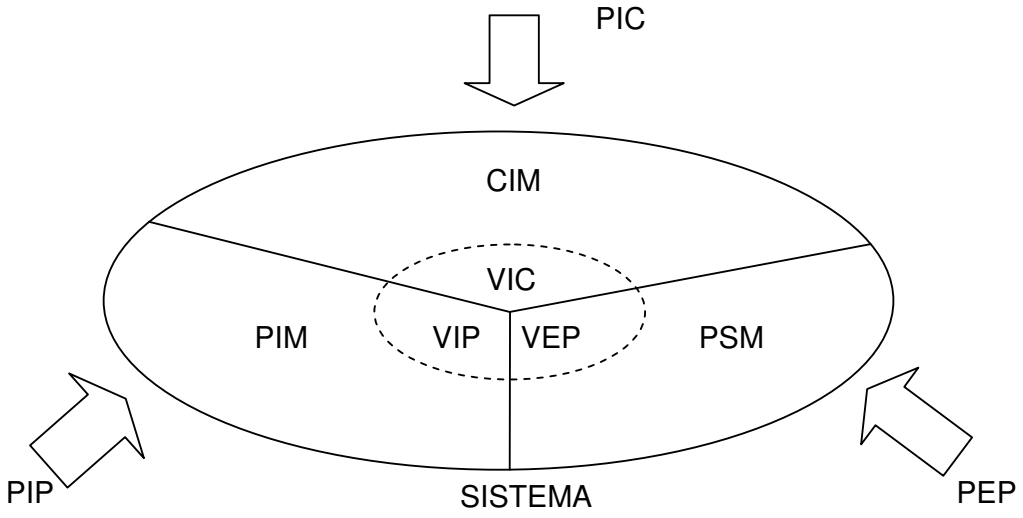

Figura 3-4: Inclusão das visões da arquitetura nos modelos MDA do sistema.

As atividades do PDA MDA são localizadas, a seguir, em relação um processo baseado no modelo de ciclo de vida denominado Waterfall (ou cascada) [ROYCE, 70] com o fim de servir de referência de como contextualizar as fases do PDA em um PDS.

Da comparação entre os objetivos das fases de um PDS Waterfall (requisitos, análise, projeto e construção) e do PDA MDA, podemos tomar as seguintes conclusões:

- A fase de requisitos fornece subsídios para a definição da arquitetura, uma vez que, define o processo de negócio e os requisitos, porém, não é necessário que todos os requisitos estejam identificados, apenas uma parte dos requisitos tem influência na fase de Requisitos Arquiteturais do PDA.
- A fase de análise modela: a decomposição funcional, o comportamental do sistema, conforme um estilo arquitetural e a organização dos elementos da fase de Projeto Arquitetural Independente de Plataforma do PDA.
- A fase de projeto detalha o projeto de *software,* conforme o resultado da fase do PDA Projeto Arquitetural Específico de Plataforma, que organiza os elementos do sistema e identifica as dependências de implementação e, define quais táticas serão usadas para implementar os requisitos não-funcionais que o sistema deve atender.

A Figura 3-5, mostra como as fases do PDA MDA se incorporam ao PDS Waterfall.

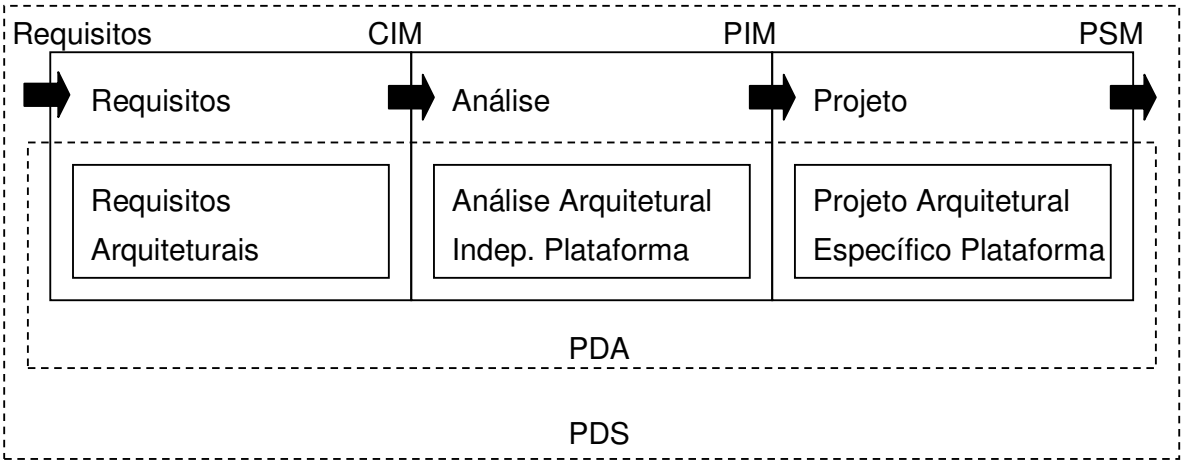

Figura 3-5: Relação entre as fases do PDA MDA e o PDS.

3.4. Processo de Desenvolvimento MDA para Web Services – MDA4WS

Este tópico propõe um PDS MDA, aplicado ao desenvolvimento de um domínio tecnológico, em particular - *Web Services,* com a finalidade mostrar uma instância detalhada do processo de desenvolvimento MDA.

O MDA para *Web Services* (MDA4WS), é um processo de desenvolvimento de *software,* centrado em arquitetura e, guiado por modelos; as atividades de modelagem guiam o curso do entendimento, o projeto e a construção do sistema, cujas fases estão baseadas nas seguintes fases do processo Waterfall: Requisitos, Análise e Projeto.

As atividades do processo MDA4WS são influenciadas pelo produto a ser desenvolvido, sistemas automatizados B2B, neste contexto, os únicos atores do sistema são os sistemas externos.

Este processo serve de guia para os principais papéis dos *Web Services*, tanto o desenvolvedor do cliente do serviço, como o do fornecedor, já que as atividades do fornecedor englobam as do cliente, todo fornecedor potencialmente consome serviços também.

A partir do ponto de vista do cliente, as atividades focam nos serviços a serem consumidos nos processo e, a partir do ponto de vista do fornecedor, as suas atividades focam nos serviços a serem expostos para os clientes e parceiros.

Modelos do MDA4WS

A especificação gerada pelo processo MDA4WS, é composta dos três modelos CIM, PIM e PSM.

A seguir, são listados os diagramas e a informação dos modelos do MDA4WS. Informações adicionais sobre o propósito e estrutura dos diagramas e documentos, são fornecidas no detalhamento do processo.

CIM

O modelo CIM é uma especificação de requisitos do sistema composta de:

- Documento de escopo do sistema
- Diagrama de Casos de Uso
- Documento de Fatos e Regras de Negócio
- Diagrama de Objetos de Negócio
- Diagrama de ciclo de vista de Objetos de Negócio
- Diagrama de Processo de Negócio
- Diagramas e informação do VIC
- Documento de Requisitos Especiais

PIM

O modelo PIM é uma especificação do sistema independente de plataforma composta de:

- Diagrama de Serviços e Mensagens
- Diagrama de Protocolo de Serviço
- Diagrama de Fluxo de Trabalho
- Diagrama de Classe
- Diagrama de Seqüência
- Diagrama de Estado
- Diagrama de Pacotes
- Diagrama de Entidades
- Diagramas e informação do VIP

PSM

O modelo PSM é uma especificação do sistema especificaespecifico de plataforma composta de:

- Descrição de plataforma
- Diagrama de Camadas
- Diagrama de Componentes
- Diagrama de pacotes
- Diagrama de Implantação
- Regras de Mapeamento PIM para PSM
- Diagrama de descrição de serviço
- Diagrama de orquestração
- Diagrama de banco de dados
- Diagrama de classe
- Diagrama de seqüência
- Diagrama de estado de classes
- Guia de Desenvolvimento

Descrição do Processo

Os seguintes elementos foram usados na especificação detalhada do processo, com o fim de fornecer uma descrição dessas tarefas, de forma prescritiva e ordenada:

- Fase – agrupa e delimita um conjunto de atividades, com um objetivo comum.
- Atividade – descreve a execução de tarefas, operações e ações para produzir ou transformar um artefato.
- Artefato – descreve uma porção de informação produzida ou usada pelas atividades do processo como modelos, documentos e código.
- Lista de verificação – descreve pontos de revisão entre os artefatos, produzidos em uma fase

3.4.1. Fase de Requisitos

Esta fase especifica o sistema sob o PIC, modelando as funcionalidades, ocultando a estrutura de processamento do sistema, sendo identificados os objetivos, as regras de negócio e as funcionalidades do sistema. Além do funcionamento do sistema devem ser identificados requisitos não-funcionais como: restrições, segurança, disponibilidade, tempo de resposta, capacidade entre outras características, que influenciarão o projeto do sistema. A informação identificada é modelada em um conjunto de diagramas e texto que, juntos compõem o modelo CIM – produto desta fase.

Esta fase visa descrever:

- Os objetivos, regras e processos de negócio.
- A informação utilizada nos processos identificados.

3.4.1.1. Atividades

Para produzir o CIM são necessárias as seguintes atividades:

- Levantar escopo e requisitos do sistema.
- Identificar termos, fatos e regras de negócio.
- Identificar objetos e processos de negócio.
- Identificar Requisitos arquiteturais.

Levantar escopo e requisitos do sistema

Esta atividade visa identificar o escopo do sistema e os requisitos funcionais e não-funcionais. O processo não indica qual técnica de levantamento de requisitos deve ser usada (por exemplo, entrevistas, questionários, workshop de requisitos),

deixando esta decisão para a instanciação do processo, em função do contexto do projeto.

Os requisitos funcionais são documentados através de casos de uso. Devem ser identificados os atores do sistema, pessoas ou organizações que interagem com o sistema fornecendo ou recebendo informações. A partir das necessidades dos atores, devem ser identificados os casos de uso, funções de negócio que serão respostas a eventos de negócio. Cada caso de uso deve ser detalhando através da descrição do fluxo de eventos, que o ator realiza ao interagir com o sistema.

Os artefatos resultantes desta atividade são:

- Documento de escopo do sistema
- Diagrama de caso de uso

Identificar termos, fatos e regras de negócio.

Nesta atividade, identificam-se os termos, os fatos e as regras de negócio. Inicialmente, são identificados os termos que indicam objetos com que o negócio lida, formando o vocabulário básico da especificação de requisitos, que darão origem ao glossário de termos, e que, posteriormente, poderão ser identificados como objetos do negócio, atributos ou estados. Posteriormente, serão identificados fatos que descreverão a natureza ou estrutura operacional de uma organização e, relacionam termos.

O último passo é identificar as regras de negócio. Uma regra de negócio é uma diretriz que influencia ou guia o comportamento do negócio. As regras de negócio são computacionalmente independentes.

Os artefatos resultantes desta atividade são:

- Glossário de termos
- Documento de Fatos e Regras de Negócio

Identificar objetos e processos de negócio

O objetivo desta atividade é analisar a informação usada no sistema e representar e decompor os processos de negócio identificados.

A informação identificada a partir dos casos de uso será representada como objetos de negócio. Os objetos de negócio representam as coisas com que o negócio lida seus atributos e relacionamentos. Alguns objetos de negócio podem se comportar de modo diferente, em função do seu estado, podendo esse comportamento ser representado como seqüência de estados.

A identificação dos processos de negócio pode ser realizada a partir dos casos de uso ou de processos já modelados na organização. Os processos de negócio identificados são decompostos em passos, com a menor granularidade possível, de forma a expor a lógica do fluxo de trabalho. Os passos do processo servirão de ponto de partida para as atividades da próxima fase.

Os artefatos resultantes desta atividade são:

- Diagrama de objetos de negócio
- Diagrama de ciclo de vida de objetos de negócio
- Diagrama de processo de negócio

Requisitos Arquiteturais

Esta atividade executa a fase Requisitos Arquiteturais do PDA MDA.

Os artefatos resultantes desta atividade fazem parte da:

- Visão VIC da descrição de arquitetura.

3.4.1.2. Artefatos

Documento de escopo do sistema

Objetivo: Descrever o escopo do sistema, os eventos e as funções de negócio.

Formato: Texto

Resultante da atividade: Levantar requisitos e escopo do sistema

Considerações:

- Identificar os atores envolvidos.
- Identificar funções de negócio - o nome de uma função de negócio é uma expressão que começa por um verbo no infinitivo.
- Identificar os casos de uso preliminares.
- Identificar requisitos não funcionais e restrições impostas à construção do sistema.

Diagrama de casos de uso

Objetivo: Descrever os requisitos do sistema e as funcionalidades do sistema através de casos de uso.

Formato: O modelo de caso de uso é composto de diagramas de casos de uso e descrições textuais dos fluxos primários e alternativos de cada caso de uso.

Resultante da atividade: Levantar requisitos e escopo do sistema

Considerações:

- O nome de um caso de uso deve ser um comportamento único, identificável e razoavelmente atômico do sistema ou parte dele.
- Comportamento comum em mais de um caso de uso deve ser fatorado,chamando tal comportamento a partir de outros casos de uso, através de inclusões.
- Alternativas de comportamento podem ser fatoradas, chamando tal comportamento através de extensões.
- Descrever o fluxo de eventos de modo claro o suficiente para que o cliente seja capaz de lê-lo. Usar as seguintes sessões: nome, objetivo, fluxo principal e fluxo alternativo. Opcionalmente Pré-condições e Pós-condições e requisitos especiais (não funcionais).
- Descrever um conjunto de cenários que especifique as condições normais e variantes do caso de uso.

Ao modelar diagramas de casos de uso:

- Exibir apenas os casos de uso que sejam importantes para o entendimento do sistema ou, parte do sistema dentro de um contexto.
- Mostrar apenas os atores relacionados com esses casos de uso.
- As associações entre casos de uso, devem ser direcionais, indicando com a seta quem inicia a comunicação.

Documento de fatos e regras de negócio

Objetivo: Detalhar a informação usada no negócio e identificar regras de negócio.

Formato: Texto

Resultante da atividade: Identificar termos, fatos e regras.

Considerações:

- Os fatos representam possibilidades e não obrigatoriedades.
- Um fato pode ser expresso pelo menos de duas formas diferentes em cada uma delas um dos termos vinculados é o sujeito e o outro é o objeto da frase.
- Espécies de fatos [BRG, 00]:
    - Atributos (<termo>é atributo/pode possuir<termo>)
    - Generalização (<termo>é uma espécie de/pode ser<termo>)
    - Agregação (<termo>é composto/é parte de<termo>)
    - Associação (<termo>pode <verbo><termo>)
    - Papel (<termo>pode ser <papel> de<termo>).
    - Regras de negócio são estruturas descritivas em linguagem natural que definem os processos e fatos com o papel que desempenha cada usuário.
    - As regras de negócio são construídas a partir de termos e fatos de negócio.
    - As regras ser categorizadas em: cálculos e derivações, restrições, habilitação de ações.
- Tipos de regras [BRG, 00]:
    - Cálculos – definem como os termos são calculados a partir de outros termos.
    - Fatos derivados –um fato é derivado de outros fatos quando ele pode ser deduzido ou inferido a partir de outros fatos.
    - Restrições – estabelecem condições para a ocorrência de fatos. Restrições não são absolutas, podendo ser definidos procedimentos em caso de violação.
    - Habilidade de ações – define condições em que certas ações são executadas. As ações devem estar definidas como funções de negócio.

Diagrama de objetos de negócio

Objetivo: Representar a estrutura e atributos dos objetos de negócio.

Formato: Diagrama de classe

Resultante da atividade: Identificar objetos e processos de negócio

Considerações:

- Os objetos de negócio representam coisas com que o negócio lida normalmente, são coisas ou documentos que existem fisicamente.
- Os nomes dos objetos identificados devem constar do glossário de termos
- Os relacionamentos entre objetos derivam de fatos de associação e, são representados como associações entre objetos no diagrama.

Diagrama de ciclo de vida de objetos de negócios

Objetivo: Representar os possíveis estados que um objeto de negócio pode passar.

Formato: Diagrama de estado

Resultante da atividade: Identificar objetos e processos de negócio

Considerações:

- Representar os estados do objeto e os eventos de que provocam a transição entre estados.
- Os nomes dos estados são termos que devem ser definidos no glossário.
- As transições devem estar definidas nas regras de negócio.

Diagrama de processo de negócio

Objetivo: Representar as atividades de um processo de negócio que envolvem um ou mais casos de uso.

Formato: Diagrama de atividades

Resultante da atividade: Identificar objetos e processos de negócio

Considerações:

- Os passos devem ser da menor granularidade possível
- Deve se dar foco no fluxo e passos
- Não utilizar raias do diagrama

Descrição de Arquitetura – visão VIC
Ver descrição de arquitetura MDA, visão VIC.

3.4.1.3. Lista de verificação

- Cada termo usado na descrição de um fato ou regra de negócio deve estar definido no glossário de termos.
- Cada função de negócio identificada deve ser realizada em um caso de uso.
- Cada objeto de negócio criado, usado ou transformado em um caso de uso, deve constar no diagrama de objetos de negócio.
- Cada transição do diagrama de ciclo de vida de objetos deve ter uma função de negócio identificada.
- Cada caso de uso deve ser representado em um processo de negócio.

3.4.2. Fase de Análise

O objetivo desta fase é analisar como o sistema será decomposto funcionalmente em elementos independentes de plataforma. O resultado desta atividade é um modelo denominado PIM, que descreve como o sistema satisfaz os requisitos identificados no modelo CIM. O PIM é analisado e modelado sob o PIP da MDA, ocultando detalhes de como os elementos de análise usam recursos da plataforma de destino do sistema.

A modelagem do PIM se inicia identificando um conjunto de operações de serviço preliminares, descobertas a partir dos passos da decomposição de processos de negócio. As operações serão alocadas em grupos coerentes, denominados serviços que definem a fronteira do sistema. Estes passos permitem definir a fronteira do sistema, identificando quais serviços serão desenvolvidos. Nesta fase adicionalmente busca-se identificar elementos com potencial de reúso.

Posteriormente, cada serviço é analisado e decomposto em elementos independentes de plataforma, definidos na fase de análise arquitetural independente de plataforma.

A modelagem desta fase visa:

- Definir um estilo arquitetural para decompor os serviços
- Descrever a responsabilidade dos serviços
- Identificar as estruturas de informação usadas no sistema

As funcionalidades arquiteturalmente significativas são modeladas na Descrição de Arquitetura – visão VIP.

3.4.2.1. Atividades

Para a modelagem do PIM são necessárias as seguintes atividades:

- Análise Arquitetural Independente de Plataforma
- Identificar serviços
- Identificar realização de serviços
- Refinar sistema

Análise Arquitetural Independente de Plataforma

Esta atividade executa a fase Análise Arquitetural Independente de Plataforma do PDA MDA.

Os artefatos resultantes desta atividade fazem parte da:

- Visão VIP da descrição de arquitetura.

Identificar serviços

O objetivo desta atividade é identificar quais serviços serão construídos e suas responsabilidades.

A modelagem do PIM se inicia identificando as operações preliminares, a partir do processo de negócio modelado na fase anterior. As operações são depuradas excluindo tarefas manuais que não possam ser automatizadas ou as que estão fora do escopo como sistemas legados, que não possam ser integrados com o sistema.

As operações selecionadas são agrupadas em serviços que identificam serviços candidatos, e as operações pelas quais são responsáveis. As operações podem agrupadas segundo vários critérios, por exemplo, por entidade, por tarefa ou por aplicação.

A granularidade dos serviços deve ser analisada, um serviço pode ser responsável por um passo de um processo, um sub-processo ou mesmo por um processo todo. Os serviços responsáveis por processos ou sub-processos podem delegar parte de suas atividades a serviços de menor granularidade, se for identificado potencial de reúso. O serviço passa a se denominar de serviço composto e, seu comportamento, será descrito através de um fluxo de trabalho ou um diagrama de seqüência.

Após a descoberta de serviços e suas operações, deverá ser identificada a informação usada nas operações e sua ordem de chamada.

A todo instante devem ser avaliado os serviços e o agrupamento de operações, deve-se verificar que não há sobreposição de responsabilidades nem duplicidades, garantindo o principio de autonomia dos serviços.

As operações e serviços identificados são inicialmente preliminares durante o início desta atividade, devendo suas definições serem encerradas antes do fim da fase.

Os artefatos resultantes desta atividade são:

- Diagrama de Serviços e Mensagens
- Diagrama de Protocolo de Serviço
- Diagrama de fluxo de composição

Identificar a realização dos serviços

O objetivo desta atividade é analisar a lógica de processamento de cada serviço, através de elementos de análise. O foco é identificar o que o serviço deve fazer, deixando para a fase seguinte projetar a realização do serviço.

Do ponto de vista do cliente do serviço, esta atividade identifica como o serviço será consumido. Do ponto de vista do fornecedor de serviço, esta atividade identifica como o serviço oferecido será realizado, através de objetos computacionais independentes de plataforma. O sistema é modelado estaticamente e dinamicamente, baseado no estilo arquitetural selecionado.

Os artefatos resultantes desta atividade são:

- Para cada caso de uso:
    - Diagrama de classe
    - Diagrama de seqüência
    - Diagrama de estados
    - Diagrama de entidades

Refinar sistema

Nesta atividade, o sistema é verificado, os serviços são revisados e, é analisada a possibilidade de reúso da lógica dos serviços em outros contextos.

Inicialmente, o sistema é verificado, analisando se ele atende a todos os requisitos e regras de negócio especificados no CIM, se não há incoerência entre os modelos produzidos e, se o modelo está completo e organizado.

Os serviços identificados são revisados para verificar se eles estão aderentes aos princípios SOC: reusabilidade e autonomia.

Para atender o principio de reusabilidade, busca-se identificar operações existentes de outros sistemas, que possam ser reusadas ou identificar operações a serem desenvolvidas, que possam ser utilizadas por outros sistemas. Caso haja a identificação de uma possibilidade de reuso, devem ser executadas de novo as atividades anteriores, com o fim de atualizar os modelos devido à inclusão do serviço identificado.

Os serviços podem ser agrupados em subsistemas de análise, com a finalidade de organizar e facilitar seu gerenciamento no caso de um sistema com grande número de serviços.

3.4.2.2. Artefatos

Descrição de Arquitetura – visão VIP

Ver descrição de arquitetura MDA, visão VIP.

Diagrama de Serviços e Mensagens

Objetivo: Descrever os serviços identificados e a estrutura das mensagens trocadas.

Formato: Diagrama de Classe

Resultante da atividade: Identificar serviços

Considerações:

- Cada serviço identificado deve ser representado com uma classe com o estereótipo serviço.
- Cada operação do serviço deve ser representado como método na classe
- Identificar qual estrutura de mensagem se necessita para chamada do método e representa-la como parâmetro do método.
- Identificar qual estrutura de mensagem de retorno na chamada do método e representa-la com retorno do método.
- Para cada estrutura de mensagem identificada representar como uma classe com o estereótipo de documento.
- Representar a dependência entre cada interface e as mensagens que envia ou recebe.
- Para cada método detalhar de forma textual quais as pré e pós-condições, quando necessário para aclarar.

- Usar anotações no modelo para registrar requisitos não funcionais

Diagrama de Protocolo de Serviço

Objetivo: Representar uma visão geral de como os serviços identificados colaboram para implementar o processo de negócio.

Formato: Diagramas Colaboração ou Seqüência

Resultante da atividade: Identificar serviços

Considerações:

- Identificar cliente do serviço como um ator
- Identificar serviços externos como um ator
- Representar as operações do serviço chamadas pelo ator
- Representar as operações chamadas internamente pelo serviço.
- Enumerar a ordem das mensagens
- Não representar elementos internos ao serviço

Diagrama de fluxo de trabalho

Objetivo: Representar o fluxo do trabalho de serviços identificados.

Formato: Diagrama de atividades

Resultante da atividade: Identificar serviços

Considerações:

- As atividades de envio e recebimento de mensagens, devem ser representadas com estereótipos gráficos para atividades *"Send"* e *"Receive"*, respectivamente.

Diagrama de Classe

Objetivo: Representar a estrutura e relacionamento das classes de análise que estão envolvidas em um ou mais casos de uso.

Formato: Diagrama de classe

Resultante da atividade: Realização dos serviços

Considerações:

- Para cada serviço fornecido identificado adicionar interface.
- Adicionar uma classe controladora que gerencie o processo que está sendo modelado.
- Dividir o controlador se ele atender a dois cenários com distintas regras de negócio.
- Adicionar um controlador para explicitar regras de negócio importantes.
- Adicionar no modelo uma entidade para conceito manipulado
- Adicionar uma interface para serviço consumido externo
- Identificar as associações entre os elementos de análise e seus contratos.
- Não representar atributos das classes
- Usar anotações no modelo para registrar requisitos não funcionais

Diagrama de seqüência

Objetivo: Representar como os objetos de análise interagem, para executar o comportamento de cenário ou um caso de uso.

Formato: Diagrama de seqüência ou colaboração

Resultante da atividade: Realização dos serviços

Considerações:

- Exibir a interação entre o ator e os elementos de análise.
- Identifique as mensagens entre os elementos de análise.
- Representar cenários alternativos de um caso de uso, se for relevante.

Diagrama de estado

Objetivo: Representar os possíveis estados que uma entidade de análise pode passar, indicando o que dispara a transição.

Formato: Diagrama de estados

Resultante da atividade: Realização dos serviços

Considerações:

- Para cada entidade com comportamento determinado por um ciclo de vida no processo representa-la em um diagrama de estados

Diagrama de pacotes

Objetivo: Organizar os elementos de análise

Formato: Diagrama de pacotes

Resultante da atividade: Realização dos serviços

Considerações:

- Deve-se organizar o diagrama de classes em pacotes, com a finalidade de poder dividir e organizar as classes, segundo algum critério como caso de uso, atores ou subsistemas.
- Representar as dependências entre os pacotes no diagrama

Diagrama de Entidades

Objetivo: Representar os elementos de informação do sistema, seus os atributos e relacionamento.

Formato: Diagrama de Classes

Resultante da atividade: Realização dos serviços

Considerações:

- Este diagrama pode não ser necessário se não houverem entidades persistentes no sistema.
- O foco é a representação lógica da informação.
- Representar como atributos a informação relacionada com a entidade
- Representar como associações os relacionamentos entre as entidades

3.4.2.3. Lista de Verificação

- O sistema deve oferecer pelo menos um serviço
- Cada caso de uso deve ter pelo menos um elemento de controle que o implemente.
- Cada objeto de análise deve pertence a um pacote

3.4.3. Fase de Projeto

Esta fase especifica o sistema sob o PEP, mostrando como a plataforma é usada para realizar o funcionamento do sistema descrito no PIM. O ambiente de implantação do sistema é identificado nesta fase e, o sistema é projetado arquiteturalmente, de modo a atender os requisitos funcionais e não funcionais. A plataforma de destino é *Web Services*, alvo deste processo, como também tecnologias relacionadas que possam gerar componentes que as implementem.

Nesta atividade é definida e executada a transformação entre os modelos PIM e PSM. O PSM resultante será refinado com detalhes específicos de plataforma que não foram gerados na transformação.

As atividades desta fase são divididas em quatro partes, sendo que a primeira define as tecnologias a serem usadas na plataforma de destino; a segunda identifica os elementos computacionais e sua distribuição e a terceira parte, define quais modelos farão parte do PSM e, as regras de mapeamento que definem como os elementos de análise serão transformados em elementos de projeto dos modelos. A quarta parte executa essa transformação. A quinta parte complementa o modelo resultante da transformação, refinando o modelo de modo a realizar os requisitos modelados no PIM.

A modelagem desta fase visa definir:

- Estrutura do sistema
- Componentes físicos
- Distribuição dos componentes

3.4.3.1. Atividades

Para a modelagem do PSM serão necessárias as seguintes atividades:

- Projeto Arquitetural Específico de Plataforma
- Definir mapeamentos PIM para PSM
- Transformar PIM
- Detalhar PSM

Projeto Arquitetural Específico de Plataforma

Esta atividade executa a fase Projeto Arquitetural Específico de Plataforma do PDA MDA.

Os artefatos resultantes desta atividade fazem parte da:

- Visão VEP da descrição de arquitetura

Definir mapeamentos PIM para PSM

Nesta atividade, serão definidos os mapeamentos entre os elementos do modelo PIM e os elementos do modelo PSM. A partir das camadas selecionadas na atividade anterior, serão identificados os modelos que farão parte do PSM como:

- Diagrama de descrição de *Web Service* (que representa a parte abstrata do WSDL a estrutura de dados através de *Schemas XML*)
- Diagrama de Orquestração *Web Services* (que representa a coordenação de processos entre *Web Service*)
- Modelo de dados (representa tabelas, campos e relacionamentos das entidades persistidas em um banco de dados relacional)
- Diagrama de Classe e seqüência dos componentes gerados (representam os elementos de código da plataforma selecionada)

As regras de transformação são escritas em linguagem natural. Os artefatos resultantes desta atividade são:

- Regras de mapeamento PIM para PSM

Transformar PIM

O objetivo desta atividade é aplicar as regras de mapeamento no modelo PIM, para transformá-lo na primeira versão do modelo PSM.

Os artefatos resultantes desta atividade são:

- Modelo PSM inicial

Detalhar PSM

Nesta atividade, o modelo PSM inicial é revisado e complementado com a informação que não foi gerada na transformação. Os diagramas do PSM são revisados, verificando se não há incoerência entre os modelos produzidos e, se o modelo está completo e atende ao que foi especificado no PIM.

Novamente, pode-se considerar o reúso de *Web Services*.

Os artefatos resultantes desta atividade são:

- Diagrama de descrição de serviço
- Diagrama de orquestração
- Diagrama de banco de dados
- Diagrama classe
- Diagrama de seqüência
- Diagrama de estados

3.4.3.2. Artefatos

Descrição de Arquitetura – visão VEP

Ver descrição de arquitetura MDA, visão VEP. A seguir são detalhados os artefatos do VEP com algumas considerações para *Web Services*.

VEP - Descrição de plataforma

Formato: Texto.

Considerações:

- Elaborar alternativas de solução
- Definir um critério de seleção
- Relacionar as tecnologias da alternativa selecionada
- Devem ser registrados os motivos de escolha da alternativa selecionada.

Critérios para seleção de uma plataforma para *Web Services*:

- Suporte ao conjunto básico de tecnologias, que permita a construção de *Web Services* básicos (XML, SOAP, WSDL, UDDI).
- Suporte às tecnologias de segunda geração, quando necessário (WS-Coordination, WS-BPEL, WS-Security)
- Meios para a construção e instalação de *Web Services*

VEP - Diagrama de camadas

Objetivo: Este artefato registra a definição de como a lógica da aplicação será dividida, em grupos coerentes e padronizados, denominados camadas, nas quais serão alocados e organizados os *Web Services* e componentes.

Formato: Diagrama de pacotes com divisão de camadas

Considerações:

- Deve-se avaliar o impacto da composição de *Web Services* no desempenho – composições podem gerar sobrecarga, especialmente quando *Web Services* intermediários necessitem processar o conteúdo de uma mensagem SOAP, realizando verificação, deserialização e serialização.
- Deve analisar o impacto da distribuição dos *Web Services* no desempenho – *Web Services* centralizados com alto grau de reuso podem gerar gargalos de processamento, que podem ser resolvidos com distribuição de carga.
- Se a empresa já organizou uma arquitetura de representação de dados XML, deve analisar a sua compatibilidade com os processos de negócio a serem desenvolvidos.
- Deve-se analisar como os requisitos de segurança (identificação, autenticação, autorização, integridade, confidencialidade, não repudio) serão forçados entre as camadas. As decisões tomadas devem ser registradas como anotações no modelo.

VEP - Diagrama de componentes

Objetivo: Representar a estrutura estática do modelo de implementação, representado como os *Web Services* são alocados nos componentes e como estes são distribuídos nas camadas da arquitetura.

Formato: Diagrama de componente

Resultante da atividade: Projetar arquitetura de *software*.

Considerações:

- Verificar se as dependências entre os componentes foram representadas.
- Componentes devem ser alocados em subsistemas.

VEP - Diagrama de pacotes

Objetivo: Identificar a estrutura de sistema, em termos de subsistemas e, exibir a alocação de componentes nos subsistemas.

Formato: Diagrama de pacotes

Resultante da atividade: Projetar arquitetura de *software*.

Considerações:

- Cada subsistema é representado como um pacote com o estereótipo subsistema
- Devem-se exibir as dependências entre os subsistemas

VEP - Diagrama de implantação

Objetivo: Capturar a configuração dos elementos de processamento (nós) e a conexões entre eles no sistema. O modelo é constituído de um ou mais nós, e canais de comunicação entre os nós. O modelo permite mapear processos para esses elementos de processamento, permitindo representar a distribuição dos artefatos nos nós.

Formato: Diagrama de implantação

Resultante da atividade: Projetar arquitetura de *software*.

Considerações:

- Identificar cada servidor que faça parte da solução como um nó.
- Representar os canais de comunicação entre os nós através de associações.
- Representar como artefatos os componentes e subsistemas a serem instalados nos nós.
- Identificar o ambiente com anotações.

VEP - Guia de desenvolvimento

Objetivo: Definir a padronização de desenvolvimento dos elementos físicos.

Formato: Texto

Resultante da atividade: Projetar arquitetura de *software*.

Considerações:

- *Web Services*
    - Detalha a nomenclatura de endpoints, operações e mensagens.
    - Definir se o corpo da mensagem do envelope SOAP será orientado a documentos ou a procedimentos (RPC).
    - Definir se será usada a definição de tipos do SOAP ou será usada a definição de um arquivo XSD externo.
    - Definir se o WSDL será construído de forma modularizada ou não.
    - Definir se será seguida a padronização do WS-I.

- Outros elementos, definir:
    - Padrão de nomenclatura de banco de dados
    - Padrão de desenvolvimento da linguagem de programação dos componentes

Regras de mapeamento - PIM para PSM

Objetivo: Identificar como os elementos do PIM serão transformados em elementos do PSM.

Formato: Texto e diagrama de classes

Resultante da atividade: Definir mapeamentos PIM para PSM

Considerações:

- O formato das regras de transformação, será em linguagem natural, se não for usado como uma ferramenta de transformação.
- Identificar o modelo e o elemento origem da transformação.
- Identificar os modelos e elementos destino da geração.
- Representar a transformação entre os elementos, através de um diagrama de classes, para exibir a dependência entre o elemento de origem e os elementos gerados.
- Devem ser seguidas as instruções da guia de desenvolvimento para geração *Web Services.*

Diagrama de descrição de serviço

Objetivo: Representar os elementos da descrição de serviço (WSDL) de cada *Web Service*.

Formato: Diagrama de classes

Resultante da atividade: Transformar PIM

Considerações:

- A descrição de serviço é representada como uma classe com o estereótipo *Web Service*
- Os esquemas de dados devem ser também representados como classes com o estereótipo schema
- Os schemas usados pela descrição de serviço são associados com o estereótipo usa.

- Devem se identificar os tipos físicos conforme decisão na guia de desenvolvimento
- Descrições de serviço existentes devem ser importadas para o modelo

Diagrama de orquestração

Objetivo: Representar a orquestração de processos (BPEL)

Formato: Diagrama de atividades

Resultante da atividade: Transformar PIM

Considerações:

- Este modelo somente é criado se houver a camada de orquestração
- Os *Web Services* orquestrados devem estar especificados no artefato diagrama de descrição de serviço.

Diagrama de banco de dados

Objetivo: Representar os elementos físicos de banco de dados como tabelas, campos e relacionamentos.

Formato: Diagrama de classes ou modelo de físico de banco de dados

Resultante da atividade: Transformar PIM

Considerações:

- O foco é a representação física do banco de dados
- Devem-se identificar as chaves primárias e estrangeiras
- Devem-se identificar os tipos físicos dos campos

Diagrama de Classe

Objetivo: Representar a estrutura estática e relacionamento das classes que implementam os *Web Services*.

Formato: Diagrama de classe

Resultante da atividade: Transformar PIM

Considerações:

- Analisar operação e revisar a estrutura de classes
- Revisar tipos e associações

Diagrama de seqüência

Objetivo: Representar como os objetos de projeto interagem para executar o comportamento na chamada de uma operação.

Formato Diagrama de seqüência

Resultante da atividade: Transformar PIM

Considerações:

- Analisar cada operação e revisar o comportamento gerado pela transformação.
- Revisar e detalhar cenários dos processos envolvidos.

Diagrama de estado de classes

Objetivo: Descrever o comportamento dinâmico de um elemento de projeto.

Formato: Diagrama de estados

Resultante da atividade: Transformar PIM

Considerações:

- Revisar se as transições previstas no modelo estão conforme especificado no modelo PIM.

3.4.3.3. Lista de Verificação

- Cada elemento de análise deve gerar um ou mais elementos de projeto na transformação.
- Cada classe deve participar em pelo menos um diagrama de seqüência
- Cada objeto que participa em um diagrama de seqüência deve ser instância de uma classe identificada no diagrama de classes.
- Cada transição de um diagrama de estados de uma classe deve estar associada a uma operação da classe.

3.5. Conclusão

Este capítulo apresentou como os conceitos da MDA foram combinados com os conceitos do IEEE1471 para a definição de um processo de arquitetura MDA que foi aplicado para o desenvolvimento de *Web Services* no processo MDA4WS.

Capítulo 4. EXPERIMENTO

O objetivo deste experimento é aplicar o processo MDA4WS com o fim de demonstrar o uso em um projeto de *Web Services* e verificar a aplicabilidade da MDA para projetos de *Web Services*.

A aplicabilidade será avaliada verificando se é possível construir um *Web Services* a partir da especificação gerada. Adicionalmente, será realizada uma avaliação qualitativa dos resultados. Como o experimento não visa fornecer uma especificação completa para codificação, não será executada a transformação do PSM em código.

O cenário do experimento é um sistema hipotético *e-Procurement* (fornecimento eletrônico), que automatiza o processo de compras de uma empresa com os seus fornecedores.

A estrutura deste capítulo é a seguinte:

- Requisitos do projeto
- Resultados do MDA4WS
- Análise qualitativa dos resultados e conclusões

4.1. Requisitos do Projeto

A empresa ABC, deseja implantar um sistema de *e-Procurement*., tem como objetivos simplificar e otimizar o processo de compra, reduzir e controlar os custos do processo de compra e, avalancar a eficiência dos fornecedores.

O departamento de TI (tecnologia de informação) decidiu não usar redes proprietárias EDI *Eletronic Data Interchange* (intercambio eletrônico de dados), para comunicação com os fornecedores, devido a seu alto custo e pouca capilaridade, escolhendo a Internet como canal de comunicação, devido a seus riscos inerentes, espera-se que a nova solução garanta um alto nível de segurança.

A primeira versão do sistema fornecerá duas funções: abertura da concorrência e recebimento de propostas. O sistema deve ser desenvolvido com o menor custo possível, utilizando a infra-estrutura existente na empresa.

A função de abertura de concorrência consiste no recebimento das seguintes informações do módulo de compras do sistema de gestão integrada ERP (*Enterprise Resource Planning*): produto, quantidade, número de propostas para encerramento e data máxima de encerramento. Após o recebimento, o sistema consulta os fornecedores cadastrados, que atendem aos requisitos da concorrência, e envia solicitações de proposta para cada um. Os fornecedores têm até a data máxima de encerramento para enviar suas propostas.

A função de recebimento de proposta consiste na revisão e avaliação de propostas da concorrência. Uma proposta deve ser aceita, somente se a concorrência estiver aberta. A concorrência se encerra quando o número de propostas recebidas for igual ao número de propostas para encerramento, informado

na abertura. Ao encerrar a concorrência é selecionada a melhor proposta, segundo os critérios de seleção. Após a seleção da melhor proposta, o sistema informa a proposta vencedora para o sistema ERP.

As partes cliente do *software* que acessarão os *Web Services* do sistema (o sistema ERP e os sistemas dos fornecedores), serão desenvolvidas por outras equipes e estão fora do escopo.

4.2. Resultados do MDA4WS

Serão apresentados os artefatos os artefatos do MDA4WS organizados pelos modelos da MDA: CIM, PIM e PSM.

4.2.1. CIM – Computational Independent Model

Conforme o capítulo anterior, o artefato inicial desenvolvido é o documento de Descrição de objetivos, e escopo do sistema que traduz os requisitos expressos na linguagem do cliente, para uma forma padronizada, identificando os objetivos, funcionalidades a serem desenvolvidas e os atores que interagirão com o sistema, conforme mostrado na Tabela 4-1, a seguir.

Tabela 4-1: Artefato de descrição de objetivos e escopo do sistema.

Descrição de objetivos e escopo do sistema
Objetivo do sistema Os objetivos do sistema são simplificar e otimizar o processo de compra, reduzir e controlar os cursos do processo de compra e avalancar a eficiência dos fornecedores. Escopo do sistema O escopo do sistema se limita a fornecer duas funcionalidades: abrir concorrência e receber propostas. Atores AT01 – Sistema ERP, módulo de compras, daqui em diante denominado “sistema de compras”. AT02 – Sistema do fornecedor Casos de uso UC01 – Abrir concorrência – O sistema recebe do sistema de compras as instruções para abertura da concorrência. UC02 – Receber proposta – O sistema recebe do fornecedor uma proposta.

Diagrama de Casos de uso

O próximo artefato a ser desenvolvido segundo o PDA, descrito no capítulo anterior, é o Diagrama de Casos de uso. O diagrama a seguir, ilustrado na Figura 4-1, mostra os dois atores identificados do sistema, e os casos de uso com que se relacionam. Os casos de uso ocorrem em dois instantes distintos, já que o processo de elaboração de reposta a uma concorrência pelo fornecedor não é automatizado.

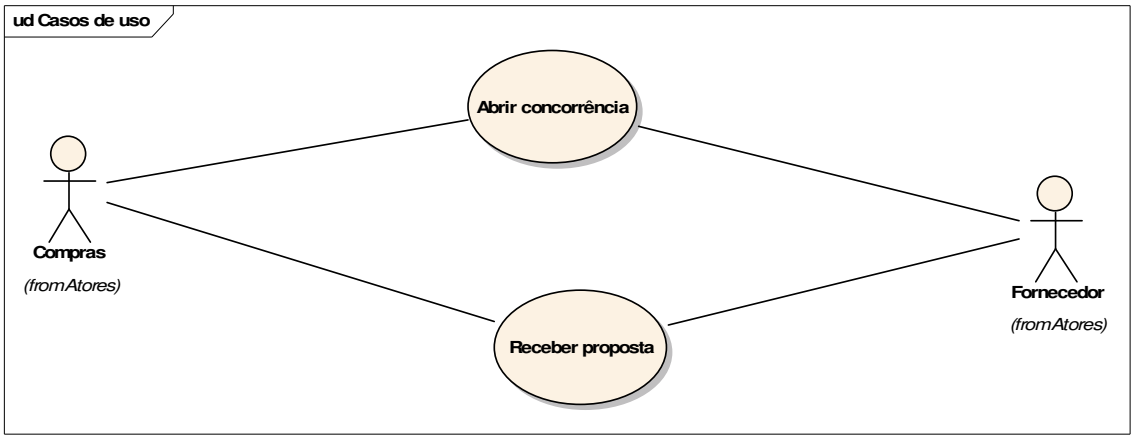

Figura 4-1: Diagrama de caso de uso.

Nas Tabelas 4-2 e 4-3, a seguir, estão exemplificadas as descrições dos casos de uso mostrados na figura anterior.

Tabela 4-2: Artefato de descrição de objetivos e escopo do sistema.

Caso de uso UC01 – Abrir concorrência
Descrição: A abertura de concorrência consiste no recebimento das informações da concorrência do sistema de compras e notificação dos fornecedores. Fluxo Principal 1. O sistema de compras envia ao sistema as informações da nova concorrência (produto, quantidade, número de propostas para encerramento, data de encerramento). 2. O sistema busca fornecedores para o produto solicitado (A1) 3. O sistema envia solicitação de cotação para cada fornecedor encontrado 4. O sistema encerra caso de uso Fluxo Alternativo A1 – Não há número de fornecedores suficiente para o número de propostas esperado. 1. O sistema não abre concorrência e notifica o sistema de compras. 2. O sistema retorna ao fluxo principal passo 4

Tabela 4-3: Artefato de descrição de objetivos e escopo do sistema.

Caso de uso UC02 – Receber proposta
Descrição: O recebimento de proposta consiste na revisão e avaliação de propostas da concorrência. Fluxo Principal 1. O fornecedor envia proposta ao sistema 2. O sistema recebe proposta e verifica se a concorrência correspondente ainda está aberta (A1) 3. O sistema aceita proposta e a armazena 4. O sistema encerra caso de uso (A2) Fluxo Alternativo A1 – Concorrência encerrada 1. O sistema rejeita proposta e notifica fornecedor. 2. O sistema retorna ao fluxo principal passo 4. Fluxo Alternativo A2 – Número de propostas para encerramento da concorrência foi alcançado 1. O sistema avalia a melhor proposta 2. O sistema notifica sistema de compras 3. O sistema encerra concorrência 4. O sistema encerra caso de uso

Os artefatos mostrados a seguir, nas Tabelas 4-4 e 4-5, documentam sobre o significado da informação usada no sistema no Glossário de Termos e, como esta informação está relacionada no Documento de fatos e regras de negócio.

Tabela 4-4: Glossário de termos.

Glossário de Termos
Produto é insumo a ser comprado
Fornecedor é o vendedor de produto
Concorrência é um processo de compra
Proposta é um compromisso de fornecimento por um determinado valor

Tabela 4-5: Documento de fatos e regras de negócio.

Documento de fatos e regras de negócio

Fatos

Concorrência

- Produto é um atributo de concorrência
- Quantidade é um atributo de concorrência
- Número de propostas para encerramento é um atributo de concorrência
- Data máxima de encerramento é um atributo de concorrência

Proposta

- Valor é um atributo de proposta
- Uma concorrência pode receber propostas
- Um fornecedor pode propor uma proposta

Regras de negócio

Concorrência

- Um fornecedor convidado pode propor uma proposta por concorrência.
- Uma proposta pode ser recebida se a concorrência estiver aberta.
- Uma concorrência somente pode ser aberta, se o número de possíveis fornecedores for maior ou igual ao previsto na concorrência
- Uma concorrência somente pode ser encerrada, quando o número de propostas válidas recebidas for igual ao previsto na concorrência.
- Uma proposta é considerada a melhor de uma concorrência, quando seu valor é menor que o valor das demais.

Proposta

- Valor deve ser maior que zero

Diagrama de objetos de negócio

O artefato a ser elaborado a seguir, conforme capítulo anterior, é o Diagrama de objetos de negócio, mostrado na Figura 4-2. A análise dos casos de uso identificou três objetos: Concorrência, Fornecedor e Proposta.

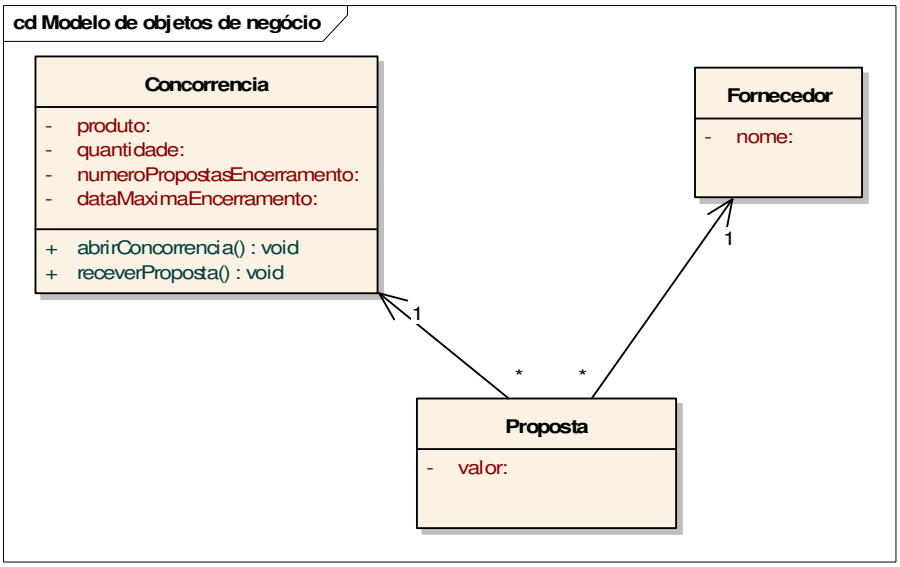

Figura 4-2: Modelo de objetos de negócio.

Diagrama de ciclo de vida de objetos de negócio

Ao analisar os objetos de negócio, foi identificado que o objeto concorrência se comporta de modo diferente após certas operações, seu comportamento foi modelado e, está representado na Figura 4-3. Foram identificados dois estados para uma concorrência: Aberta e Encerrada. Enquanto não for alcançado o número máximo de propostas para encerramento uma concorrência permanece aberta.

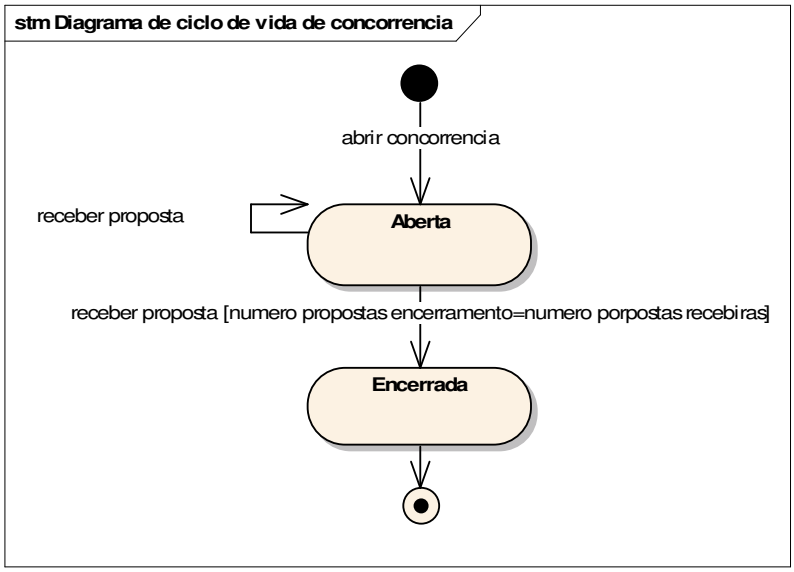

Figura 4-3: Diagrama de ciclo de vida de concorrência.

Diagrama de processo de negócio

O artefato a seguir, representado na Figura 4-4, representa todas as atividades do processo de negócio de *e-Procurement* a ser implementado pelo sistema.

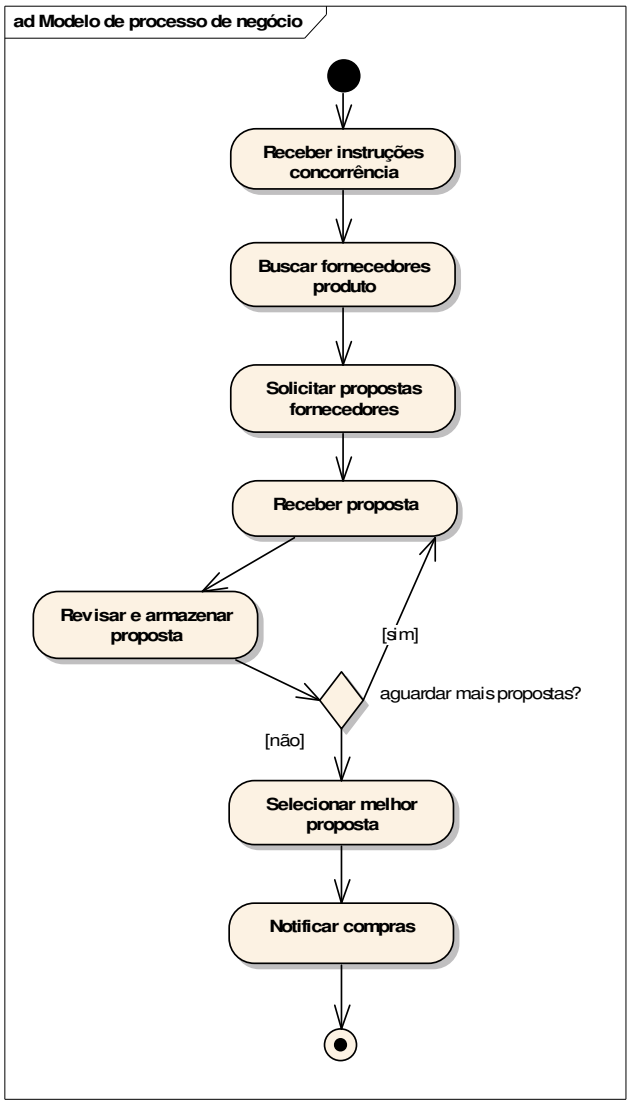

Figura 4-4: Processo de e-Procurement.

O último artefato do CIM, mostrado na Tabela 4-6, é a Relação de Requisitos Especiais, que relaciona características especiais para o sistema a serem consideradas no seu projeto.

Tabela 4-6: Relação de Requisitos Especiais.

Relação de Requisitos Especiais
O requisito de segurança requer especial atenção deste tipo de projeto onde os *Web Services* usam a Internet como rede de comunicação. Neste projeto deseja-se integridade e confidencialidade da informação a ser trafegada.

4.2.2. PIM – Platform Independent Model

Conforme o capítulo anterior, o PIM representa como os requisitos serão implementados em termos de elementos independentes de plataforma. A decisão de como estes elementos serão estruturados, é tomada durante a fase de Análise Arquitetural Independente de Plataforma do processo de descrição de arquitetura.

Como o sistema será implementado com *Web Services,* o processo de modelagem representa o sistema em duas etapas, a primeira modela e decompõe o sistema em serviços, modelando nos diagramas de serviços e mensagens o protocolo de serviços.

Em uma segunda etapa, os serviços são modelados em termos de elementos independentes de plataforma, segundo um estilo arquitetural auxiliar, denominado *Model-View-Controler*, onde, para serviço, é considerado uma *View.* O sistema é modelado a seguir, estaticamente e dinamicamente, utilizando classes de análise que capturam a especificação de requisitos, através de três tipos elementos: fronteira, controle e entidade.

Diagrama de Serviço e Mensagens

Este diagrama representa as operações dos serviços e as mensagens trocadas, sem detalhar partes internas. Devido ao escopo simplificado, os serviços do fornecedor e de compras reutilizam mensagens já existentes, como mostra o diagrama representado na Figura 4-5.

Os serviços foram identificados através do método informal SRC (Service-Responsibility-Coordination, serviço-responsabilidade-coordenação) [STOJANOVIC, 05 a] [STOJANOVIC, 05 b], para identificar os serviços. Devem ser preenchidos os três elementos de um cartão SRC:

- Serviço – nome do serviço que reflete seu alvo, objetivo e escopo.
- Responsabilidade – descrição das operações que é o serviço responsável.
- Coordenação – descrição das composições que este serviço coordena.

Foram identificados dois serviços a serem oferecidos pelo sistema: Serviço de Concorrência, responsável por abrir uma concorrência e o Serviço de Proposta, responsável por receber as propostas dos fornecedores.

Nome: Serviço de Concorrência	
Responsabilidade	**Coordenação**
Receber instruções de concorrência	Serviço do fornecedor
Buscar fornecedores para produto	
Solicitar propostas para fornecedores	
Nome: Serviço de Proposta	
Responsabilidade	**Coordenação**
Receber propostas	Serviço de compras
Revisar e armazenar proposta	
Selecionar melhor proposta	
Comunicar proposta vencedora	

O serviço do fornecedor será desenvolvido por cada empresa habilitada a participar do *e-Procurement*. Os serviços de compras são serviços a serem desenvolvido pela equipe de desenvolvimento do ERP.

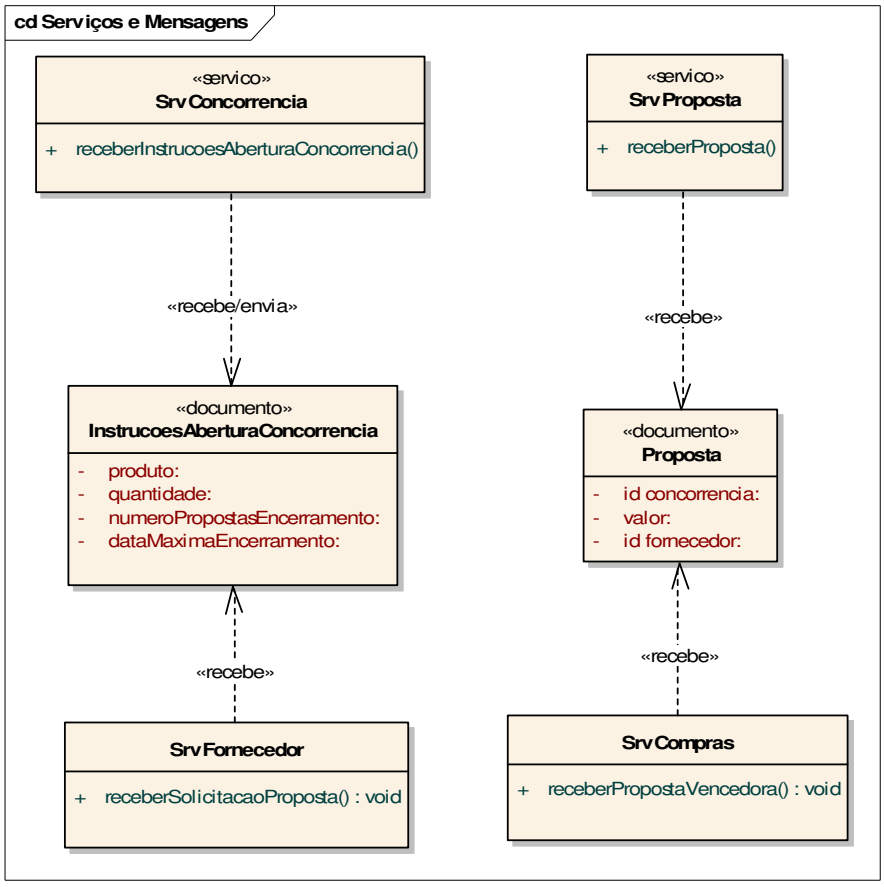

Figura 4-5: Diagrama de serviços e mensagens.

Diagrama de Protocolo de Serviço

Este diagrama de comportamento, somente representa o comportamento externo observável entre dois serviços, sem detalhar o comportamento interno. O diagrama da Figura 4-6, permite delinear como o processo será implementado com os serviços identificados.

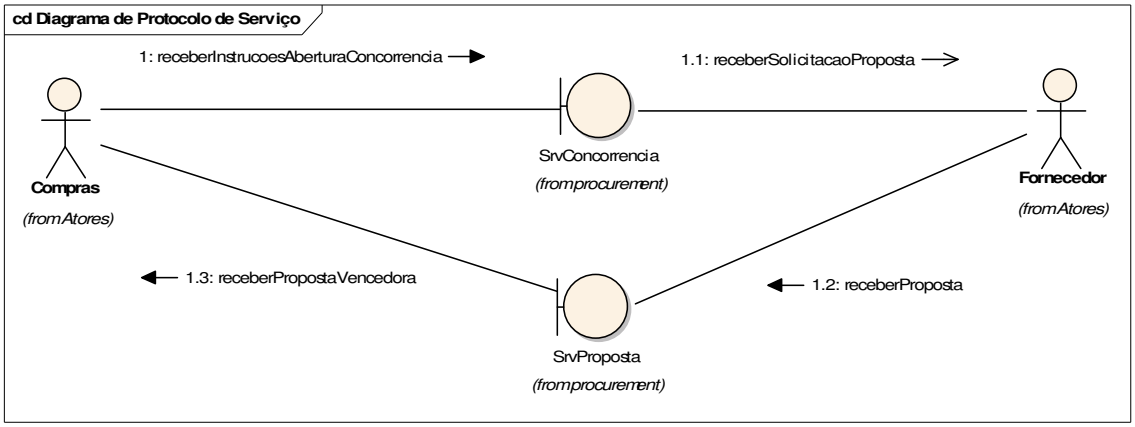

Figura 4-6: Diagrama de protocolo e serviço.

Diagrama de fluxo de trabalho

Este diagrama descreve como os serviços colaboram entre si. As Figuras 4-7 e 4-8, que exibem o fluxo dos serviços de concorrência e recebem proposta, respectivamente.

O primeiro símbolo da atividade “Receber Instrução de Abertura de Concorrência”, representa uma atividade que somente é iniciada quando o serviço é chamado. O símbolo da atividade “Solicitar Proposta”, representa uma atividade que aciona outro serviço para ser completada.

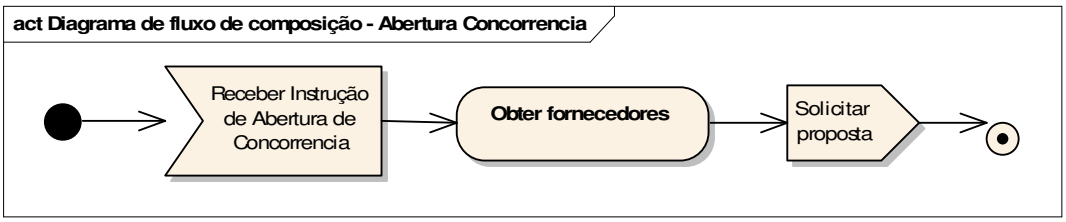

Figura 4-7: Serviço Concorrência.

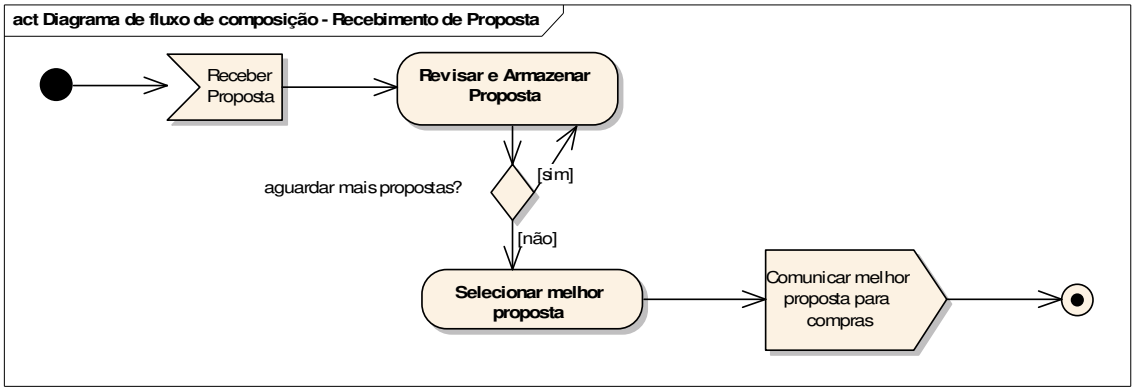

Figura 4-8: Serviço de Proposta.

Os diagramas a seguir, modelam como cada um dos serviços identificados conforme o estilo arquitetural MVC.

Diagrama de Classse

O diagrama da Figura 4-9 mostra os dois serviços, com o símbolo de *View*, e os elementos, independente de plataforma, com que devem interagir, para realizar as funcionalidades conforme o modelo CIM.

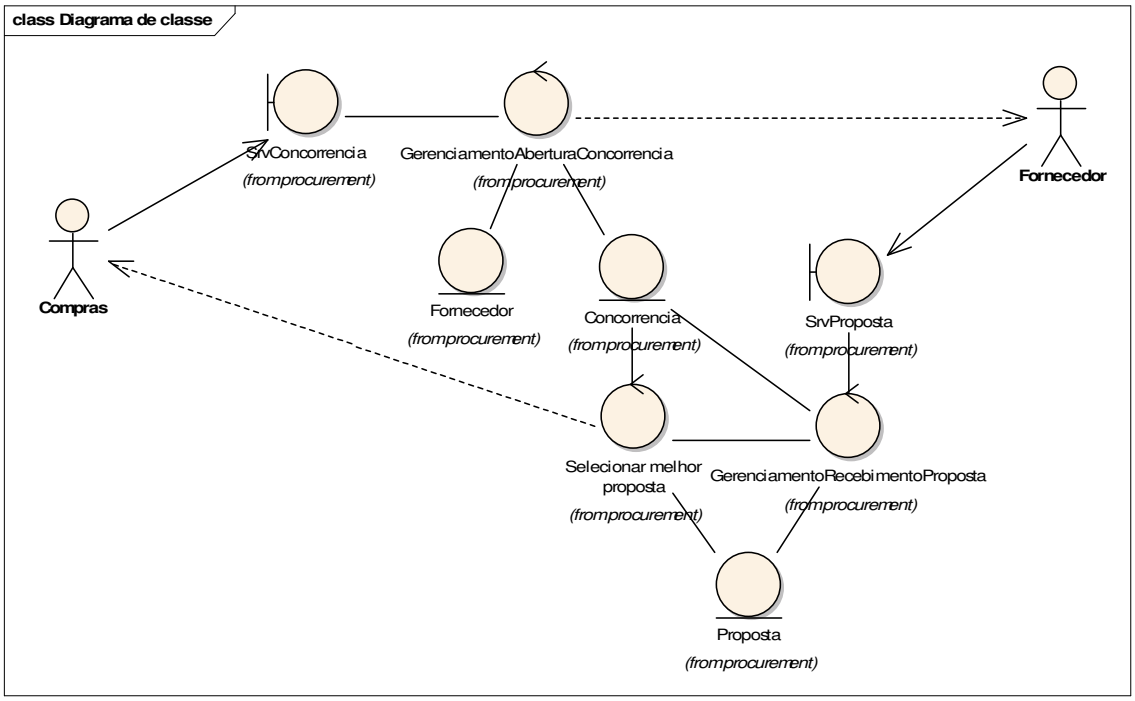

Figura 4-9: Diagrama de classe.

Diagrama de Seqüência

A seguir são modelados os dois cenários, o primeiro de abertura de concorrência, Figura 4-10, e o segundo cenário de recebimento de proposta, Figura 4-11.

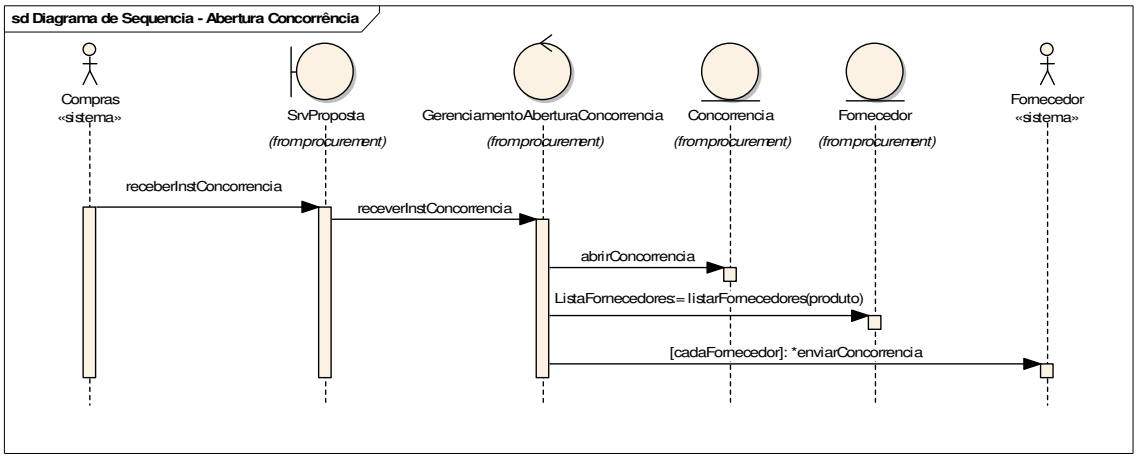

Figura 4-10: Diagrama de seqüência - Abertura de concorrência.

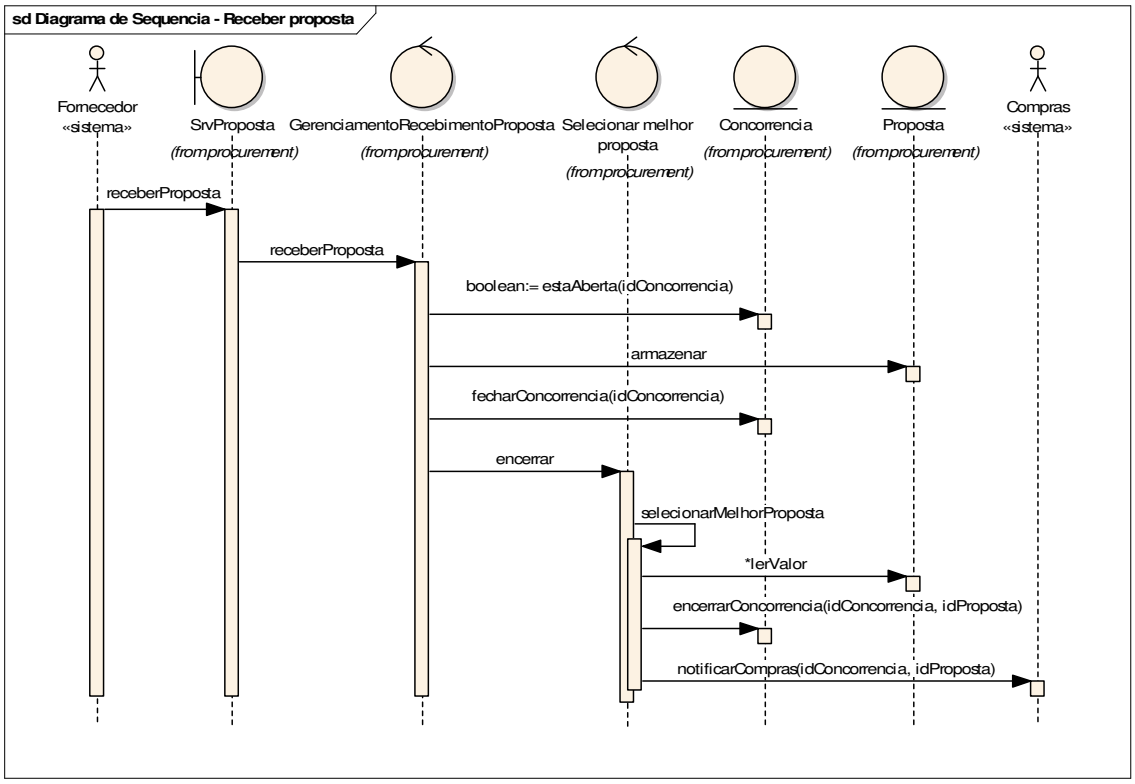

Figura 4-11: Diagrama de seqüência - Receber proposta.

Diagrama de estados

A Figura 4-12, mostra o diagrama de estados da entidade Concorrência, que se comporta de modo diferente, conforme os estados aberta e encerrada, identificado no modelo CIM, na Figura 4-3. O modelo é semelhante, mas os nomes das transições estão conforme os nomes dos métodos modelados nos diagramas de seqüência, mostrados anteriormente.

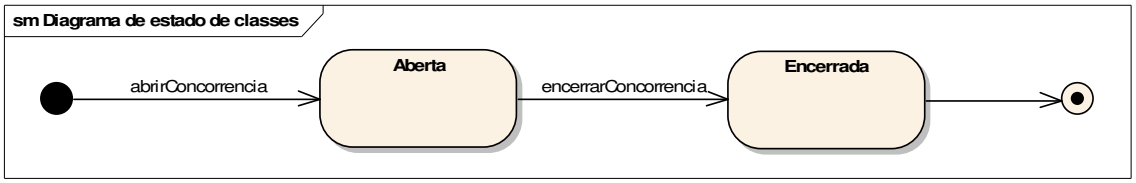

Figura 4-12: Diagrama de estados de concorrência.

Diagrama de pacotes

O pacote de análise, mostrado na Figura 4-13, organiza os elementos de análise usados nos diagramas anteriores deste sistema. Considerando uma regra informal, que se o número de elementos é inferior a dez, não será necessário dividí-los.

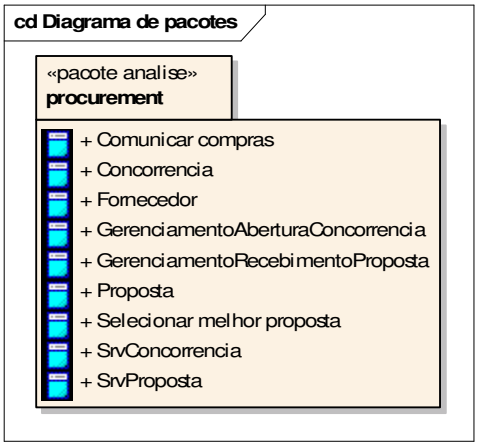

Figura 4-13: Diagrama de pacotes.

Diagrama de entidades

Este diagrama, detalha como estão modeladas as classes identificadas como entidades. O diagrama de entidades, mostrado na Figura 4-14, exibe as entidades do sistema: Fornecedor, Concorrência e Proposta.

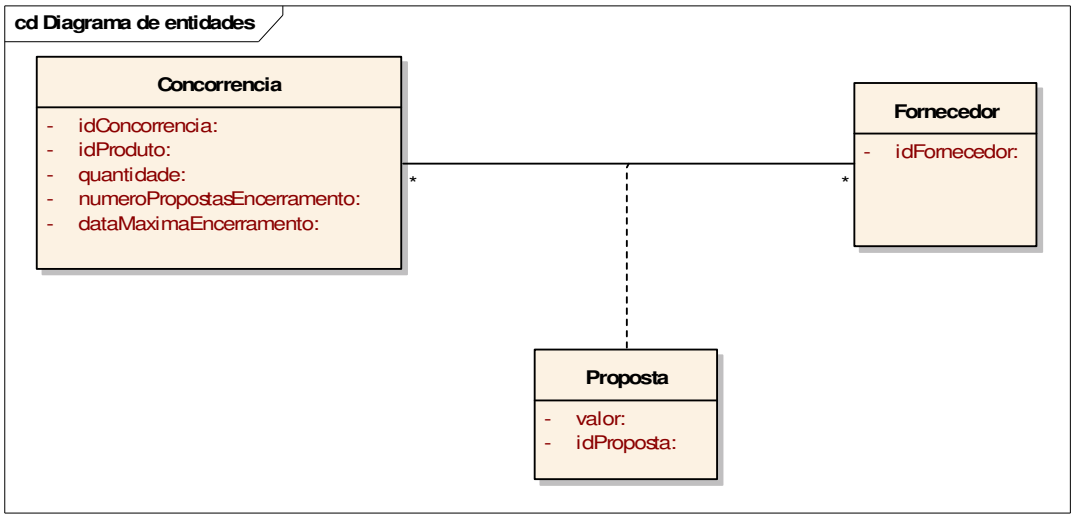

Figura 4-14: Diagrama de entidades.

4.2.3. PSM – Platform Specific Model

A seguir, são mostrados a visão VEP do processo de descrição de arquitetura, posteriormente, será mostrado como este insumo servirá para definir as regras de mapeamento e, modelo PSM resultante da transformação e detalhamento.

VEP - Descrição de plataforma

Na Tabela 4.7, estão listadas as decisões de tecnologia adotadas para o sistema, baseadas nas seguintes justificativas:

- Por uma restrição de custos e, por ser um piloto, não será adotada nenhuma tecnologia adicional à adotada na empresa ABC.
- O ambiente de desenvolvimento, será o padrão homologado para sistemas Web, é servidor de aplicação J2EE (*Java 2 Enterprise Edition*).
- Foi decidido usar um servidor para Coordenador de Processos BPEL, para poder ter controle declarativo dos processos dos *Web Services* compostos, expostos para a Internet, separado do servidor de aplicação J2EE.
- Com esta decisão, o *Web Service* de recebimento é dividido em coordenação do processo, deixando a parte de armazenamento temporário e seleção em *Web Services* auxiliares no servidor de aplicação.
- Foi decidido que, não há necessidade de descoberta de *Web Services* dentro dos *Web Services* planejados, portanto, UDDI foi excluído como parte dos padrões da solução.

Tabela 4-7: Identificação de plataforma

Descrição de Plataforma
Linguagem: Java 1.4.2, J2EE 1.3, XML, SOAP.
Coordenador de processos: BPEL Engine
Servidor de Aplicação: Servidor J2EE
Frameworks: JDBC

Uma vez definida a plataforma de destino, o próximo passo é analisar em quais camadas serão divididas as funcionalidades do sistema.

VEP - Diagrama de camadas

A Figura 4-15, mostra as camadas selecionadas para o sistema, sendo que as responsabilidades estão explicitadas na Tabela 4-7.

Tabela 4-8: Diagrama de camadas.

Diagrama de camadas
As responsabilidades das camadas são: • Acesso: consome os *Web Services* oferecidos pelas camadas de orquestração e serviço. • Orquestração: composição e coordenação de *Web Services* compostos. • Serviços: apresentação e processamento das mensagens recebidas • Componentes: processamento da lógica de negócio. • Recursos: acesso à informação em banco de dados ou sistemas legados.

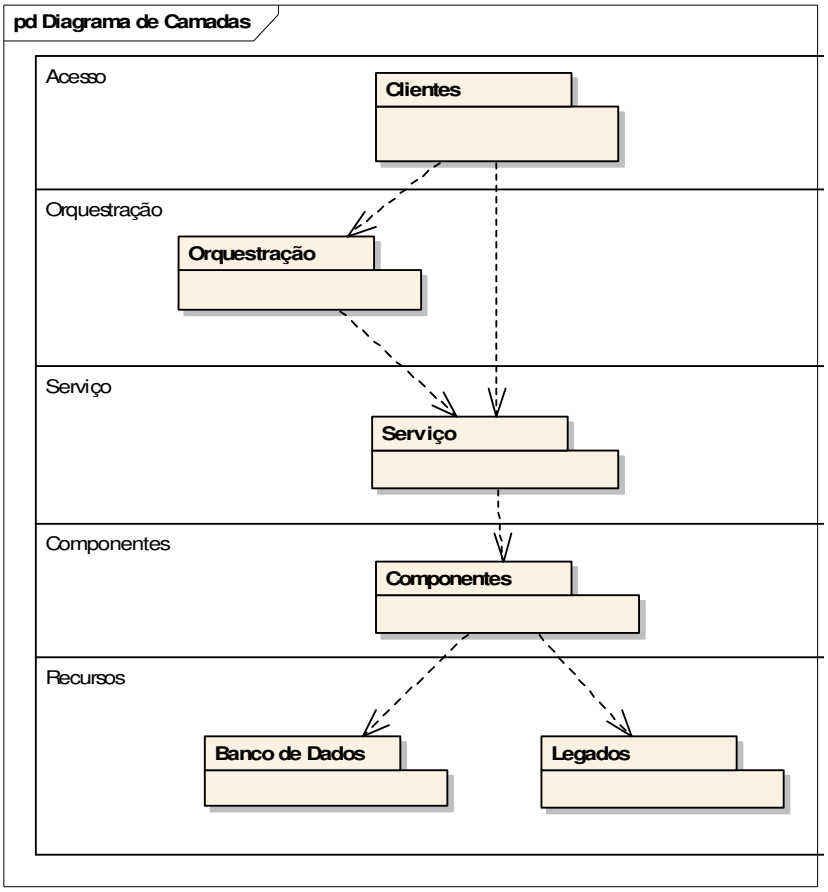

Figura 4-15: Diagrama de Camadas.

VEP - Diagrama de componentes

Devido à inclusão da camada de orquestração, as responsabilidades do *Web Service* de recebimento de proposta são divididas. Os *Web Services* da camada de orquestração não realizam acesso direto à camada de recursos, devendo ser criados novos *Web Services* na camada inferior.

O *Web Services* de recebimento passa a coordenar o uso de dois novos *Web Services,* conforme mostra a Figura 4-16: *Web Services* de armazenamento e *Web Services* de seleção, assume-se que há possibilidade de reúso em outros processos de compra. O *Web Services* de armazenamento possui a responsabilidade verificar se a concorrência está aberta e, caso afirmativo, armazenar a proposta. O *Web Services* de seleção de propostas possui a responsabilidade de selecionar a melhor proposta, encerrar a concorrência e notificar a proposta vencedora ao sistema de compras.

O fluxo de trabalho do *Web Service* Abrir Concorrência não possui uma lógica de fluxo de trabalho complexa, e é um *Web Service* interno, de menor granularidade, por isso foi decidido implementá-lo na camada de *Web Services*.

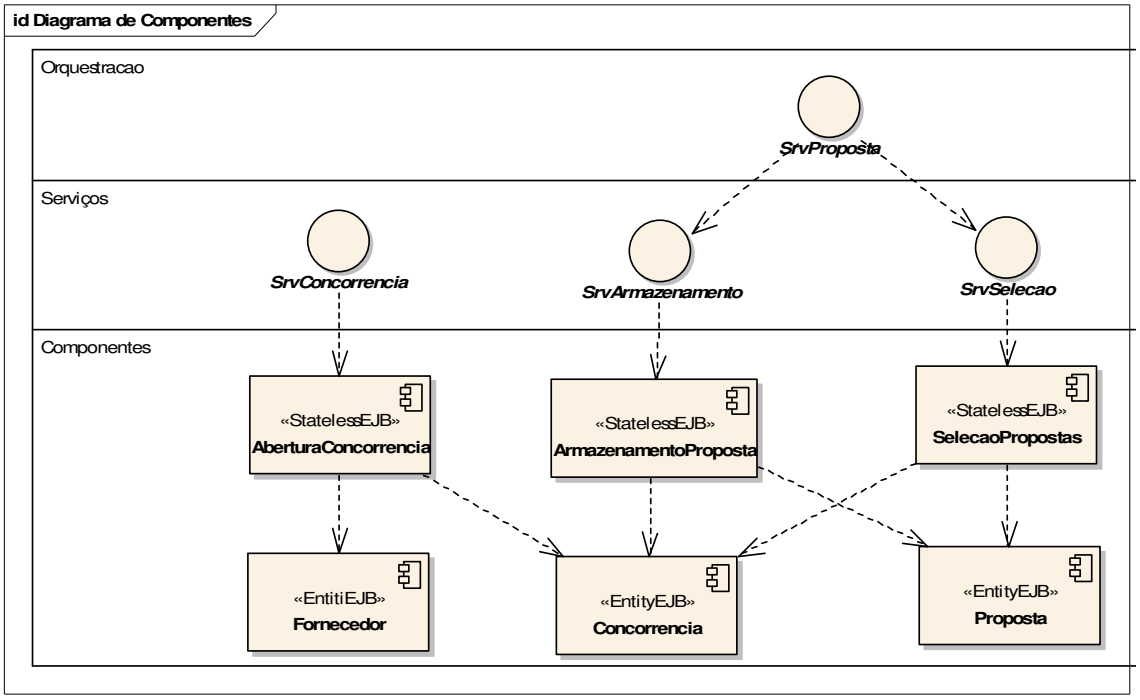

Figura 4-16: Diagrama de componentes.

VEP - Diagrama de pacotes de subsistemas

Este artefato foi suprimido, devido a que todos os componentes pertencem a um único sistema, não havendo a necessidade de dividí-los em subsistemas.

VEP - Diagrama de implementação

Na Figura 4-15, abaixo, é mostrado como o sistema será implantado nos nós que compõem a solução.

A comunicação entre o sistema dos fornecedores e o sistema ocorre através de um *Firewall* e, entre o sistema e o sistema de compras, dentro da área desmilitarizada (DMZ) do ambiente de produção.

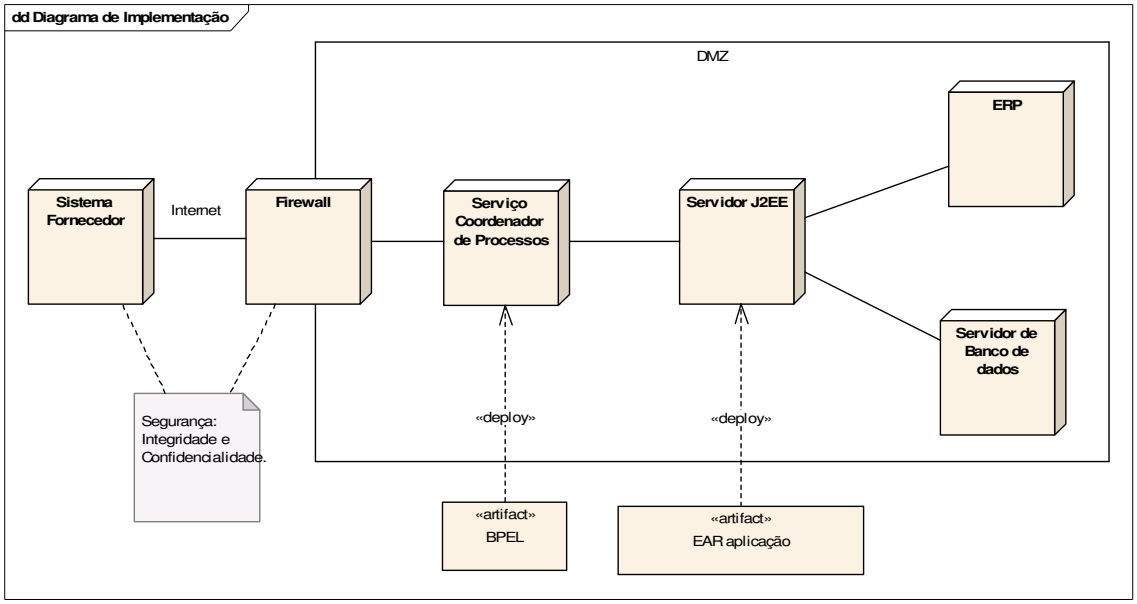

Figura 4-17: Diagrama de implementação.

VEP - Guia de desenvolvimento

Na Tabela 4-7, estão os itens de padronizações para a plataforma selecionada.

Tabela 4-9: Guia de desenvolvimento.

Guia de desenvolvimento
Web Services Adotar as Basic-Proifile do WS-I, para garantir um nível de interoperabilidade entre os *Web Services* a serem desenvolvidos futuramente. Nomenclatura de elementos Operações: verbos seguidos de objetos, SOAP Estilo de mensagem: Documento (para maior compatibilidade com as especificações de segunda geração) Esquema de tipos: Literal (seguindo recomendação do WS-I) Segurança Usar os padrões: WS-Security, XML-Encription e XML-Signature para integridade e confidencialidade das mensagens.

Java Utilizar o guia de estilo de programação interno. Tratamento baseado em dois tipos de exceções – de sistema e de aplicação. As exceções de sistemas para erros de acesso à recursos. Erros de aplicação para exceções previstas nos casos de usos. A plataforma J2EE 1.4 oferece dois modos de criar *Web Services*: Web Tier *Endpoint* e EJB *Endpoint*. O primeiro expõe uma *servlet* como *Web Service* e, o segundo, um EJB. Optou-se pela segunda opção, porque permite um controle de segurança declarativo por operação.
Segurança O uso da Internet impõe requisitos críticos de segurança sobre o acesso ao *Web Services* de Recebimento de Proposta, foram adotadas soluções para os seguintes aspectos: • Identificação, Autenticação e Autorização – será utilizada a autenticação do contêiner Web J2EE. Os dados de usuário e senha irão dentro da mensagem SOAP no *header* de segurança do WS-Security • Integridade – foi adotado o padrão XML Signature para garantir a origem e a integridade das mensagens SOAP. • Confidencialidade - foi adotado o padrão XML Encription para encriptar as mensagens SOAP

Regras de mapeamento PIM para PSM

A partir da definição da arquitetura do sistema, sob a visão específica de plataforma, é possível definir como transformar o modelo PIM no modelo PSM.

Na Tabela 4-10, relacionam-se as regras de mapeamento organizadas por diagrama, esta primeira versão do documento define apenas as transformações dos elementos estáticos.

A Figura 4-18, resume de forma gráfica, as dependências entre os elementos de origem e os gerados.

Tabela 4-10: Regras de mapeamento PIM para PSM.

Diagrama de origem: Serviço e Mensagens
Diagrama de destino: Diagrama de descrição de serviço
• Para cada serviço no modelo de origem será criada uma classe com o estereótipo *Web Service*, no modelo de destino que representa um WSDL. • Para cada operação de serviço no modelo de origem será criado um método de *Web Services,* no respectivo *Web Services* no modelo de destino. • Para cada documento será criada uma classe com o estereótipo *schema* que, representa um elemento de informação no Schema XML, no modelo de destino com seus respectivos elementos de informação.
Diagrama de origem: Serviço e Mensagens
Diagrama de destino: Diagrama de classes do PSM
• Para cada serviço no modelo de origem, criar uma classe no modelo de destino, com estereótipo Stateless EJB com suas respectivas operações.
Diagrama de origem: Diagrama de Entidade
Diagrama de destino: Diagrama de banco de dados
• Para cada entidade no modelo de origem, criar uma tabela no modelo de destino. • Para cada atributo com nome id+nome tabela, criar um campo chave primária. Para os demais atributos, criar os respectivos campos. • Para cada classe de associação, criar uma nova tabela com os campos e, duas associações para tabelas das classes relacionadas com cardinalidade de destino um e de origem n.
Diagrama de origem: Diagrama de Entidade
Diagrama de origem: Diagrama de classes
• Para cada entidade no modelo de origem, criar uma classe no modelo de destino, com o estereótipo EntityBean. • Para cada atributo com nome id+nome tabela criar um atributo com estereótipo chave primária. Para os demais atributos, criar os respectivos atributos.

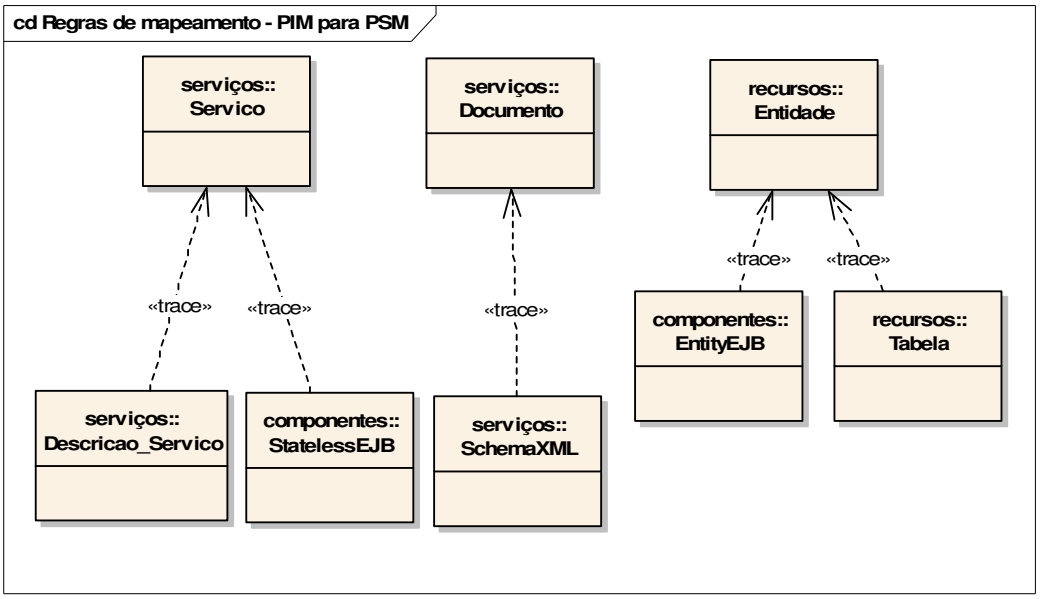

Figura 4-18: Regras de mapeamento PIM para PSM.

PSM Detalhado

O restante do capítulo mostra o PSM gerado e complementado. As regras de transformação, definidas no item anterior, somente geram a parte estática, devendo a parte dinâmica do sistema ser completada manualmente.

Diagrama de descrição de serviço

A Figura 4-19 mostra o modelo de descrição de serviço dos dois *Web Services* oferecidos e, dos dois *Web Services* consumidos, necessários para a implementação das funcionalidades do sistema. A partir deste modelo, é possível gerar os arquivos WSDL de descrição de serviço.

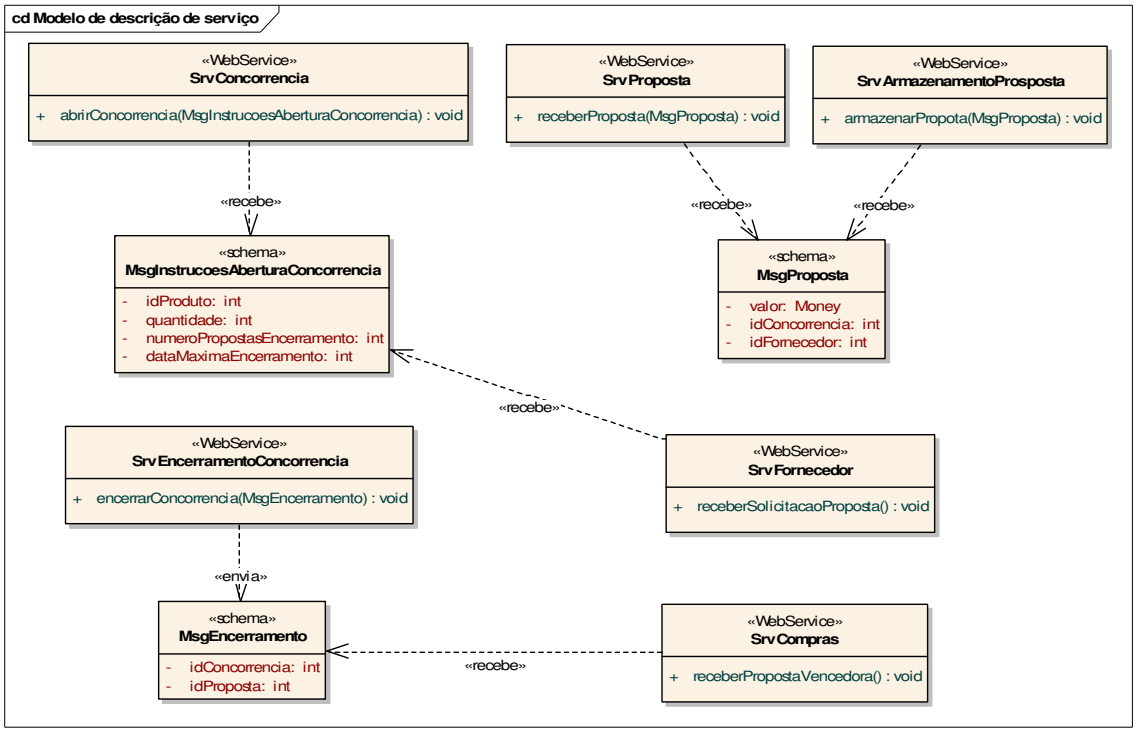

Figura 4-19: Modelo de descrição de serviço.

Diagrama de orquestração

O modelo de orquestração da Figura 4-20, representa o fluxo de trabalho do processo - Receber Proposta, levando em consideração a divisão de responsabilidades entre a camada de orquestração e os serviços. As atividades Armazenar Proposta e Encerrar Concorrência, é chamada elementos da camada de serviços.

A partir deste modelo, é possível produzir o arquivo BPEL, com o processo de negócio, que o coordenador de processos irá mediar.

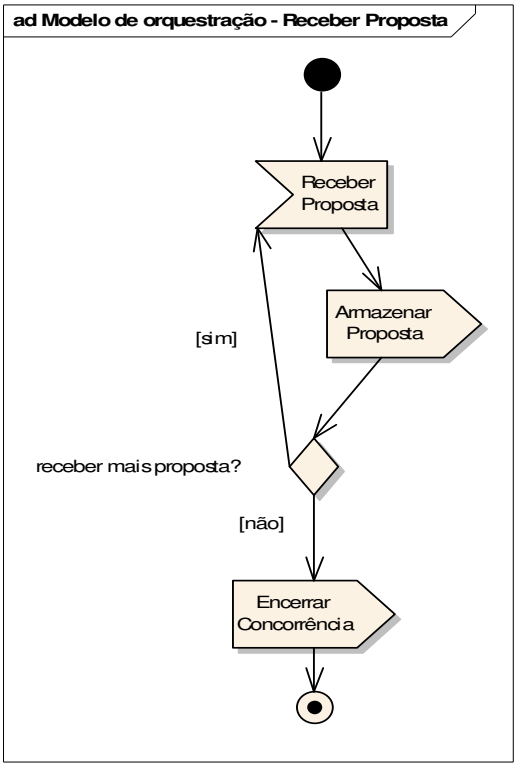

Figura 4-20: Modelo de orquestração - Serviço Receber Propostas.

Diagrama de banco de dados

Na Figura 4-21, está representado o modelo de banco de dados, a partir deste modelo é possível gerar o esquema para criação do banco de dados.

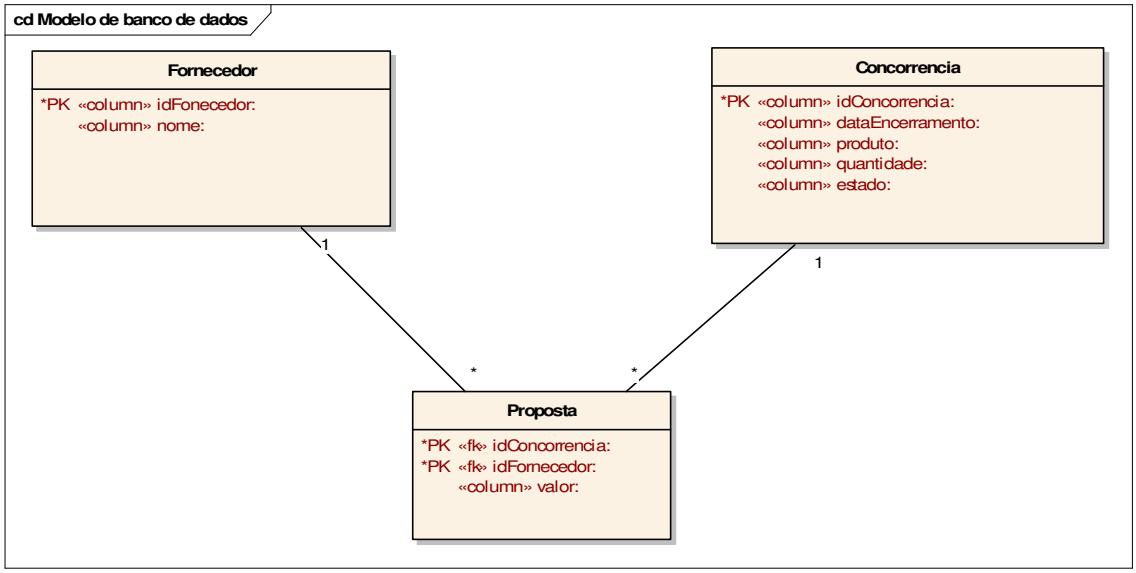

Figura 4-21: Modelo de banco de dados.

Diagrama classe

O diagrama de classes mostrado na Figura 4-22 representa os EJB que fazem parte da camada de componentes. Um EJB é implementado com três classes: interface remota, interface home e a implementação do bean. O nível de dependência de plataforma adotado, não modela as interfaces dos diagramas do PSM. A partir deste diagrama é possível gerar a estrutura dos arquivos fonte em Java.

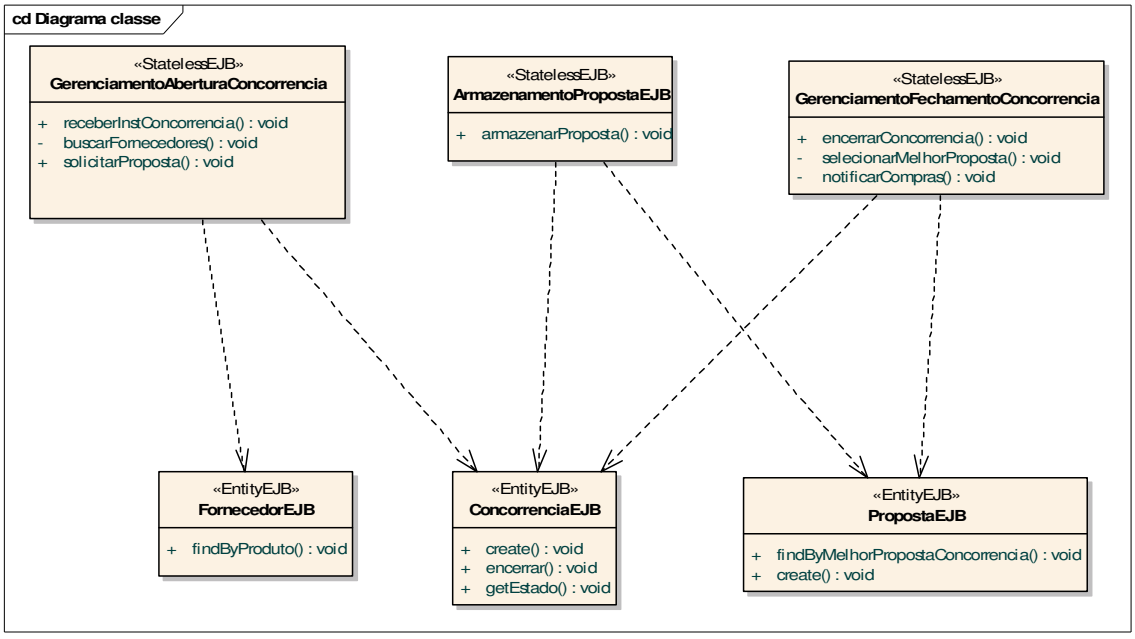

Figura 4-22: Diagrama de classes.

Diagrama de seqüência

Nas figuras a seguir, são mostrados os principais cenários em que colaboram os elementos do sistema.

Na Figura 4-23, está representado o cenário de Abrir Concorrência; optou-se por não representar o cenário alternativo, do caso de uso UC01, A1 – Fornecedores insuficientes.

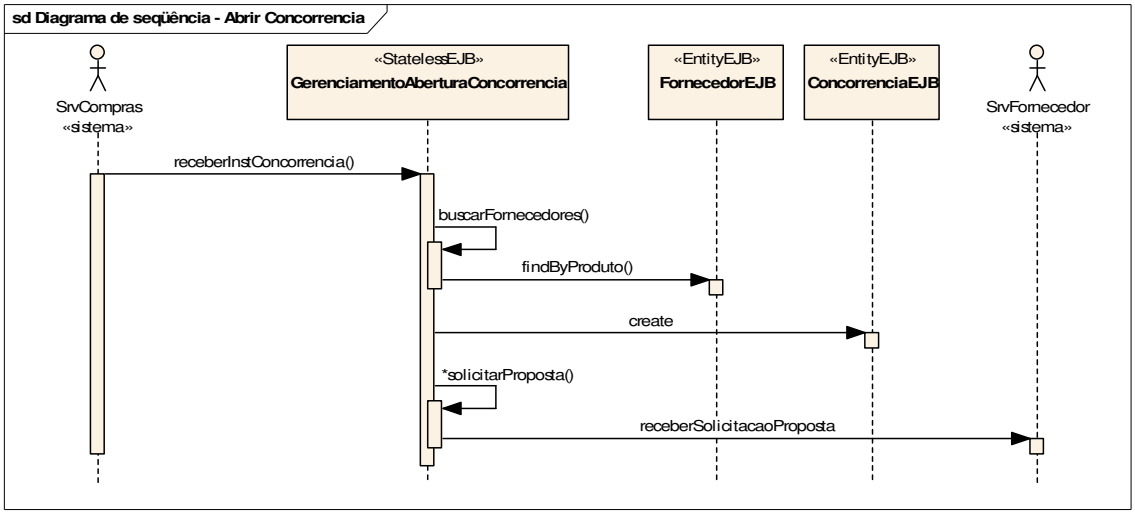

Figura 4-23: Diagrama de seqüência - Abrir Concorrência.

Na Figura 4-24, estão o Recebimentos de Propostas, num cenário em que não existem propostas suficientes para encerrar a concorrência. Na Figura 4-25, está representado o cenário de encerramento de concorrência.

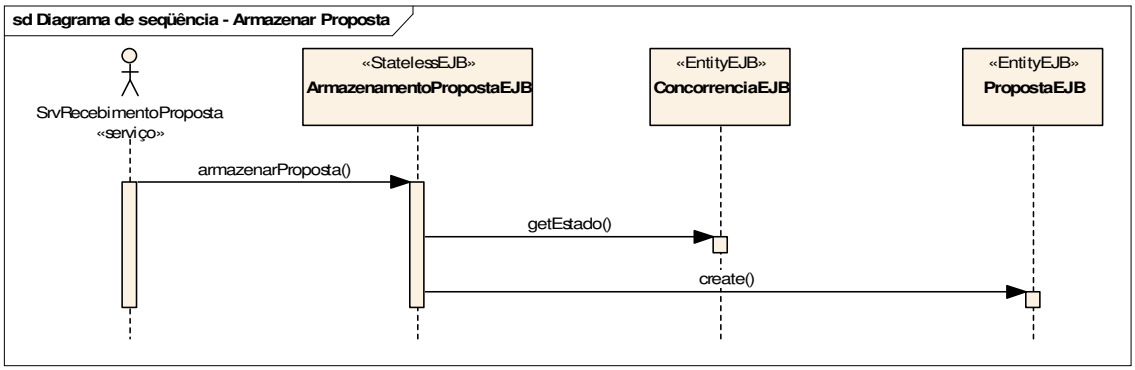

Figura 4-24: Diagrama de seqüência - Armazenar Proposta.

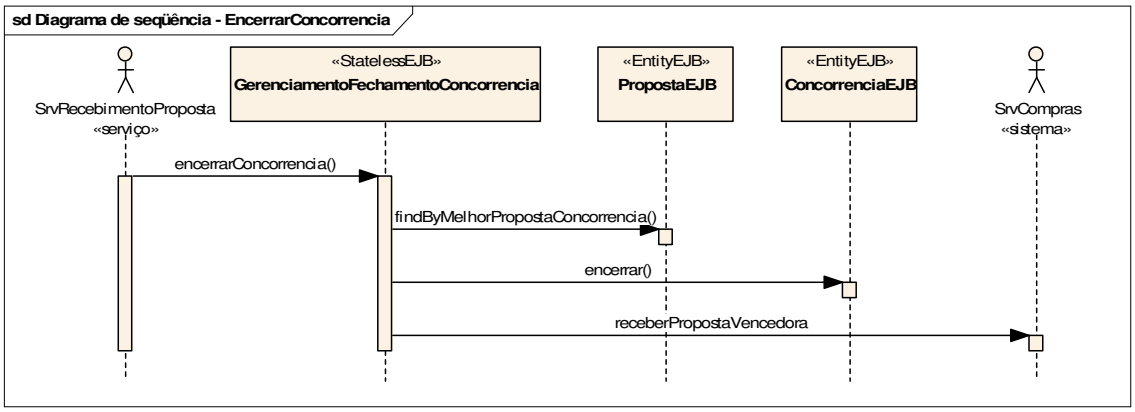

Figura 4-25: Diagrama de seqüência - Encerrar Concorrência.

Diagrama de estados

A Figura 4-46, mostra o diagrama de estados de concorrência. As transições de estado são controladas pela classe *entity bean* ConcorrenciaEJB. O objeto é criado com o estado Aberto e quando alcançada as condições de encerramento, passa a ser considerado Encerrado.

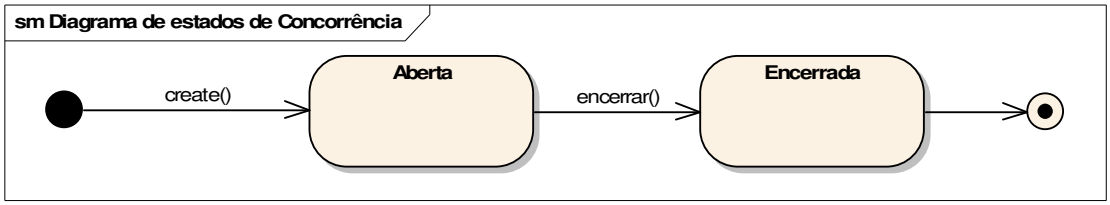

Figura 4-26: Diagrama de estados – Concorrência.

A partir dos arquivos de seqüência e estados, é possível implementar manualmente Java, o código fonte dos métodos criados a partir do diagrama de classe.

4.3. Analise dos Resultados

O uso do processo MDA4WS no experimento gerou como resultado, uma série de modelos, diagramas e informações de projeto. Este resultado, será avaliado para verificar a aplicabilidade da MDA no desenvolvimento de Web SErvices.

4.3.1. Critérios

A aplicabilidade da MDA para o desenvolvimento de *Web Services,* pode ser avaliada considerando algumas características da MDA esperadas nos resultados. A especificação pode ser avaliada segundo as variáveis heurísticas e qualitativas a seguir. Foram adotados os seguintes valores de avaliação: baixo, médio e alto.

Independência de plataforma do PIM: variável que avalia o quanto o modelo PIM é independente da plataforma de destino.

- Baixo – baixa independência de plataforma – elementos independentes de plataforma só podem ser implementados em uma única plataforma.
- Médio - média independência de plataforma– elementos independentes de plataforma são abstratos o suficiente para serem implementados em mais de uma plataforma.
- Alto – alta independência de plataforma – elementos independentes de plataforma são abstratos o suficiente para serem implementados em qualquer plataforma.

Transformação PIM para PSM: variável heurística que avalia o quanto o modelo PSM pode ser gerado automaticamente, a partir dos PIM, através das regras de mapeamento. Quanto maior o trabalho na atividade Detalhar PSM da fase de projeto, menor será o nível de transformação.

- Baixo – regras de mapeamento não geram nenhum diagrama.
- Médio – regras de mapeamento não geram uma parte do diagrama.
- Alto – regras de mapeamento geram todo ou quase todas os diagrama.

Transformação PSM para Código: variável heurística que avalia o quanto da implementação pode ser gerada automaticamente, a partir do PSM, através das regras de mapeamento. Quanto maior o trabalho de programação manual, menor será o nível de transformação.

- Baixo – regras de mapeamento não geram nenhum elemento.
- Médio – regras de mapeamento não geram uma parte do elemento.
- Alto – regras de mapeamento geram todo ou quase todas as partes do elemento.

Definição de arquitetura: variável heurística que avalia os mecanismos de definição de arquitetura do processo.

- Baixa – especificação não possui descrição de arquitetura.
- Médio – especificação possui descrição de arquitetura particular.
- Alto – especificação possui descrição de arquitetura baseada nos conceitos da recomendação 1471.

4.3.2. Resultados

O nível de Independência de plataforma do PIM deste primeiro experimento, é considerado Médio, este nível não é maior porque em um projeto de *Web Services,* as alternativas de estilos arquiteturais estão restritas aos estilos suportados pelos *Web Services*.

O nível de transformação PIM para PSM, neste primeiro experimento, foi considerado alto. A Tabela 4-11, resume a avaliação realizada por diagrama. Devido à complexidade da atividade de definição das regras de transformação, não foram geradas regras para definir o mapeamento para transformar os diagramas de fluxo de trabalho, de seqüência e estados.

Tabela 4-11: Nível de transformação PIM para PSM.

Diagrama de Origem	Diagrama de Destino	Nota
Descrição de serviço e mensagem	Diagrama de descrição de serviço	Alto
Protocolo de Serviço	Diagrama de Orquestração	Alto
Diagrama de: Classe, Seqüência, Estado de classes.	Diagrama de: Classe, Seqüência, Estado de classes.	Médio
Entidades	Modelo de banco dados	Alto

De maneira semelhante, foi avaliado o nível de transformação PSM para Código, como médio. Apesar de não ser escopo do processo a geração de código, é possível prever a possibilidade de transformação dos elementos físicos, a partir dos diagramas do PSM, conforme resumido na Tabela 4-12.

Tabela 4-12: Nível de transformação PSM para código.

Diagrama de Origem	Elemento Físico	Nota
Descrição de serviço	WSDL	Alto
Orquestração	BPEL	Alto
Classe, Seqüência, Estado de classes.	EJB	Médio
Banco de dados	Esquema de banco de dados	Médio

O nível de definição da arquitetura foi considerado alto, já que o processo incorpora os conceitos do IEEE1471 e fornece mecanismo para definir uma descrição da arquitetura, que ajude a compreender a arquitetura de um sistema baseado em *Web Services,* antes do seu desenvolvimento.

A partir dos resultados anteriores, resumidos na Tabela 4-13, é possível construir o sistema do experimento proposto baseado Web Services, a partir da especificação gerada pelo MDA4WS, e, por conseqüência, é aplicável o uso dos conceitos da MDA, na construção de Web Services para este experimento.

Tabela 4-13: Resultados do experimento MDA4WS.

Variáveis	Projeto MDA4WS
Independência de plataforma do PIM	Médio
Transformação PIM para PSM	Alto
Transformação PIM para PSM	Médio
Definição de arquitetura	Alto

Capítulo 5. CONCLUSÕES

Este capítulo apresenta as conclusões sobre este trabalho e, como ele se relaciona com MDA e *Web Services*, assim como comentários gerais e possíveis pontos de prosseguimento de pesquisa.

Aplicabilidade da MDA em Projetos de *Web Services*

Este trabalho mostrou como os conceitos da MDA podem ser aplicados no desenvolvimento de *Web Services.* A estratégia utilizada foi criar um processo de desenvolvimento de software, centrado em arquitetura.

Os conceitos da recomendação IEEE1471 serviram de ferramenta para incorporar o conceito de ponto de vista da MDA, na arquitetura de software, em especial, os conceitos sobre especificação de pontos de vista que serviram para definir e especificar os mesmos, usando de diagramas da UML, preferencialmente, organizados por visão. Para a descrição de arquitetura proposta, foi elaborado um processo para guiar a sua construção.

O processo MDA4WS incorporou as atividades do processo de arquitetura e as especializou para o domínio tecnológico, objetivo desta dissertação: Web Services.

Os pontos de vistas utilizados na descrição de arquitetura são os elementos-chave para estruturar o processo de modo sistemático, sendo que, o ponto de vista PIC, é fundamental para definir o contexto dos demais.

O modelo CIM traz ao processo uma visão das necessidades dos envolvidos, de forma estruturada e completa dos requisitos do sistema, esta visão é necessária para descrever o que o sistema deve cumprir para atender aos requisitos propostos.

No experimento, conforme análise dos resultados do capítulo anterior, o modelo PIM possui um nível de portabilidade médio, devido à separação de interesses, resultante da incorporação do conceito de independência de plataforma na especificação, que não foi maior porque em um projeto de *Web Services,* as alternativas de estilos arquiteturais estão restritas aos estilos suportados pelos *Web Services*. O grau de independência está diretamente ligado ao estilo arquitetural selecionado. A seleção de um estilo arquitetural, durante a fase de análise arquitetural independente de plataforma é o ponto crítico, porque esta decisão afeta diretamente os modelos PIM e PSM.

O MDA4WS usa métodos orientados a objetos e UML para a especificação de *Web Services*. Os diagramas convencionais da UML, permitiram descrever a estrutura de artefatos, não orientados a objetos como a estrutura dos serviços e as mensagens trocadas entre eles, no diagrama de Serviços e Mensagens e o de Protocolo de Serviço. Um ponto de melhoria do processo é criar um perfil UML, para especificar os elementos que podem ser usados para a modelagem destes diagramas, de modo formal.

Conforme observado no experimento a complexidade das atividades de projeto passam para a atividade de definição de regras de mapeamento PIM para PSM. As regras de mapeamento podem ser reaproveitadas em outros projetos, desde que o

estilo arquitetural do modelo PIM e a plataforma de destino especificada no PSM, sejam os mesmos.

Conforme avaliação realizada no final do capítulo anterior, a partir da especificação gerada pelo processo MDA4WS, é possível gerar um *Web Service.* Por conseqüência, os conceitos da MDA são aplicáveis ao desenvolvimento de *Web Services*, porém, esta conclusão está restrita a este experimento.

O uso do MDA4WS no experimento gerou um grupo de resultados que foram analisados, porém, para se obter conclusões mais gerais, devem ser realizados outros experimentos.

Prosseguimento

Este trabalho não explorou como o domínio de informação, resultante da definição das transformações poderia ser representado através de uma visão adicional, que se relaciona com as visões independente e dependente de plataforma.

Outro ponto de continuidade será o desenvolvimento de ambiente de desenvolvimento integrado, que dê suporte de automação ao processo MDA. Um ambiente integrado de desenvolvimento (*IDE – Integrated Development Enviroment*) MDA pode incluir um ambiente para execução das transformações, ambiente para modelagem, armazenamento e gestão da configuração de *software* (*SCM - Software Configuration Management*) multi-usuário permitindo que vários desenvolvedores trabalhem em partes diferentes de um modelo simultaneamente que, posteriormente, serão incorporados a um modelo compartilhado. As funções necessárias para o controle da versão, requerem que a ferramenta permita identificar as diferenças entre versões diferentes dos modelos e, permitir juntar partes de modelos [SELIC, 03].

Contribuição

A primeira contribuição deste trabalho é propor: uma abordagem arquitetural para incorporar em um processo de desenvolvimento de *software*, um grupo de conceitos da MDA. Esta abordagem arquitetural consiste de uma descrição da arquitetura de *software,* baseada nos conceitos de pontos de vista da MDA e os conceitos da recomendação IEEE1471.

A segunda contribuição deste trabalho é um processo de desenvolvimento de software MDA para *Web Services*, baseado na abordagem arquitetural resultante da primeira contribuição.

Capítulo 6. REFERÊNCIAS

[ALMEIDA, 03 a] ALMEIDA, João Paulo; Sinderen, Marten van; Pires, Luís Ferreira; Quartel, Dick. **The role of the service concept in model-driven applications development.** International Middleware Conference, Workshop Proceedings, Rio de Janeiro, Brazil. PUC-Rio, 16-20 Junho de 2003, p. 268-272.

[ALMEIDA, 03 b] ALMEIDA, J.P.A., van Sinderen, M.J., Ferreira Pires, L. & Wegdam, M. **Handling QoS in MDA: a discussion on availability and dynamic Reconfiguration**, Workshop on Model Driven Architecture: Foundations and Application (MDAFA 2003), University of Twente, Enschede, The Netherlands, June 26-27, 2003, Workshop Proceedings, p. 91-96.

[ALMEIDA, 04] ALMEIDA, João Paulo; et all. **On the Notion of Abstract Platform in MDA Development**. Enterprise Distributed Object Computing Conference, Eighth IEEE International, Monterey, Califórnia. (EDOC'04). Pg. 253-263.

[ASTUDILLO, 98] ASTUDILLO Hernán; Hammer ,Stuart; **Understanding the Architect's Job**, Software Architecture Workshop of OOPSLA'98, 1998.

[BASS, 03] BASS, Len; Clements, Paul; Kazman, Rick; **Software Architecture in Practice**, Second Edition. 2. ed. Boston, Addison Wesley, 2003

[BRG, 00] Business Rules Group; **Defining Business Rules ~ What Are They Really?** Disponível em:

http://www.businessrulesgroup.org/first_paper/br01c1.htm Acesso em 15/06/2006

[BUSCHMANN, 96] BUSCHMANN, Frank et all. **Pattern-Oriented** Software **Architecture**, Wiley, 1996.

[CLEMENTS, 02] CLEMENTS, Paul et all. **Documenting Software Architectures: Views and Beyond.** 1. ed. Boston, Addison Wesley, 2002

[ENDREI, 04] Endrei, Mark; **Patterns: Service-Oriented Architecture and *Web Service*ces**. IBM Redbook. Disponível em http://ibm.com/redbooks. Acesso: 23/10/2004

[ERL, 05] ERL, Tomas; **Service-Oriented Architecture: Concepts, Technology, and Design**. Prentice Hall PTR, NJ 2005.

[FERREIRA, 05] FERREIRA, André; **Utilização do MDA em conjunto com ODP em soluções distribuídas**. Dissertação (MDA em Engenharia de Software). São Paulo, 2005.

[GERVAIS, 02] GERVAIS, Marie-Pierre; **Towards an MDA-Oriented Methodology.** Proceedings of the 26th Annual International Computer Software and Applications Conference (COMPSAC'02), Oxford, Inglaterra, Agosto de 2002, pg. 265-270.

[HUHNS, 05] HUHNS, Michael; Singh, Munidar; Service-Oriented Computing: Key Concepts and Principles. **IEEE INTERNET COMPUTING** pg 75-81, volume 9, Janeiro-Fevereiro de 2005.

[IBM, 03] IBM. **Business Process Execution Language for *Web Services* Version 1.1**. Disponível em:http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/ Acesso 15/02/2005

[IEEE, 00] Institute of Electrical and Electronics Engineers, Inc. **IEEE Recommended Practice for Architectural Description of Software-Intensive Systems 1471**, 2000

[ISO, 95] International Standards Organization. **Reference Model of Open Distributed Processing, ITU-T X.901 | NA/IEC 10746-1 ODP Reference Model Part 1 – Overview**, 1995.

[JAZAYERI, 00] JAZAYERI, Mehdi; Ran, Alexander, Frank van der Linden; **Software Architecture for Product Families**, Addison Wesley, 2000.

[KLEPPE, 03] KLEPPE, Anneke; WARMER, Jos; BAST, Wim. **MDA Explained: The Model Driven Architecture™: Practice and Promise**, 1. ed. Boston, Addison Wesley, 2003

[MACIEL, 04] R.S. Pitangueira Maciel, B. Carreiro da Silva, D. Rihan, C.A. Guimarães Ferraz, N. Souto Rosa. **Processo de desenvolvimento de componentes através da MDA para a interoperabilidade entre aplicações de autoria colaborativa.** JIISIC'04 4ª Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento Madrid (España) Novembro de 2005.

[MELLOR, 02] MELLOR, Stephen; Balcer, Marc. **Executable UML**, 1. ed. Boston, Addison Wesley, 2002

[MELLOR, 04] MELLOR, STEPHEN, et all. ; **MDA Distilled: Principles of *Model-Driven Architecture*;** 1. ed. Boston, Addison Wesley, 2004

[MICROSOFT, 05 a] MICROSOFT et all. ***Web Services* Coordination (WS-Coordination)**, Disponível em: http://www-128.ibm.com/developerworks/library/specification/ws-tx/ Acesso 15/12/2005

[MICROSOFT, 05 b] MICROSOFT et all. ***Web Services*** **Atomic Transaction (WS-AtomicTransaction)** , Disponível em: http://www-128.ibm.com/developerworks/library/specification/ws-tx/ Acesso 15/12/2005

[MICROSOFT, 05 c] MICROSOFT et all. ***Web Services*** **Business Activity Framework**, Disponível em:http://www-128.ibm.com/developerworks/library/specification/ws-tx/ Acesso 15/12/2005

[OASIS, 02] OASIS - Organization for the Advancement of Structured Information Standards. **Universal Description, Discovery, and Integration – UDDI version 2.0.** Disponível em: http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm

Acesso 15/01/2005

[OASIS, 04] OASIS - Organization for the Advancement of Structured Information Standards. ***Web Services*** **Security: SOAP Message Security 1.0** Disponível em: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

[OASIS, 05] OASIS - Organization for the Advancement of Structured Information Standards. **Business Process Execution Language for *Web Services.*** Disponível em: http://www.oasis-open.org/committees/documents.php?wg_abbrev=wsbpel

Acesso 15/01/2005

[OMG, 01] OMG - Object Management Group, **Model Driven Architecture (MDA).** 01/07/2001. Disponível em: http://www.omg.org/cgi-bin/doc?ormsc/2001-07-01, Acesso 22/08/204

[OMG, 02] OMG - Object Management Group, **UML Profile for Enterprise Distributed Object Computing Specification (EDOC)**, Disponível em: http://www.omg.org. Acesso 17/08/2004

[OMG, 03 a] OMG - Object Management Group, **MDA Guide v1.01.** 12/06/2003. Disponível em: http://www.omg.org/cgi-bin/doc?omg/03-06-01, Acesso 22/08/2004

[OMG, 03 b] OMG - Object Management Group, **Common Warehouse Metamodel (CWM) Specification version 1.1**, Disponível em: http://www.omg.org. Acesso 17/08/2004

[PAPAZOGLOU, 03A] PAPAZOGLOU, Mike, Georgakopoulos, D.; Service-Oriented Computing **Communications of ACM**. Outubro. 2003. pg 25-28 vol 46 no. 10.

[PAPAZOGLOU, 03B] PAPAZOGLOU, Mike; **Service -Oriented Computing: Concepts, Characteristics and Directions**. Fourth International Conference on Web Information Systems Engineering (WISE'03).

[PELTZ, 03] PELTZ, Chris. *Web Services* Orchestration and Choreography. IEEE Computer pg. 46-52. Outubro 2003, volume36, número 10.

[PRESSMAN, 02] PRESSMAN, R.S., **Software Engineering, A practitioner's approach**, 5. ed. New York, MacGraw Hill, 2002.

[ROYCE, 70] ROYCE, WINSTON; **Managing the Development of Large Software Systems**. Proceedings of IEEE Westcon. 1970.

[SELIC, 03] SELIC, Bran. The Pragmatics of Model-Driven Development. **IEEE SOFTWARE** pg. 19-25. Setembro/Outubro 2003, volume 20, número 5.

[SHAW, 96] SHAW, Mary; Garlan, David; **Software Architecture. Perspectives on an Emerging Discipline**, Prentice Hall, 1996.

[SHENG, 03] SHENG, Zhang; Yun, He; Architecture-Based Software Process Model, **ACM SIGSOFT Software Engineering Notes**, volume 28 no 2, Março de 2003.

[STOJANOVIC, 05 a] STOJANOVIC, Zoran; Dahanayake, Ajantha; Sol, Henk. **Information Modeling Methods and Methodologies.** Idea Group Publishing, 2005. Capítulo XV.

[STOJANOVIC, 05 b] STOJANOVIC, Zoran; **A Method for Component-Based and Service-Oriented Software Systems Engineering.** Dissertação (Doutorado em Engenharia de Software) Doctoral Dissertation, Delft University of Technology. Janeiro de 2005.

[W3C, 00] W3C Consortium, **Simple Object Access Protocol (SOAP) 1.1**, Disponível em: http://www.w3.org/TR/SOAP Acesso 15/01/2005

[W3C, 01a] W3C Consortium, ***Web Services* Description Language (WSDL) 1.1**, Disponível em: http://www.w3.org/TR/wsdl Acesso 15/01/2005

[W3C, 01 b] W3C Consortium, **XML-Encyption WG**, Disponível em: http://www.w3.org/Encryption/2001/

[W3C, 02] W3C Consortium. **XML-Signature Syntax and Processing** Disponível em: http://www.w3.org/TR/xmldsig-core/

[WS-I, 04] *Web Services* Interoperability Organization. **WS-I Basic Profile Version 1.1**, Disponível em http://ws-i.org. Acesso: 23/10/4004