MATEUS CONRAD BARCELLOS DA COSTA

BASS: UM ARCABOUÇO DE SOFTWARE PARA INTEGRAÇÃO DE SISTEMAS DE INFORMAÇÃO

Belo Horizonte
07 de março de 2008

MATEUS CONRAD BARCELLOS DA COSTA
Orientador: Rodolfo Ferreira Resende

BASS: UM ARCABOUÇO DE SOFTWARE PARA INTEGRAÇÃO DE SISTEMAS DE INFORMAÇÃO

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação.

MATEUS CONRAD BARCELLOS DA COSTA

Belo Horizonte
07 de março de 2008

UNIVERSIDADE FEDERAL DE MINAS GERAIS

FOLHA DE APROVAÇÃO

BASS: Um Arcabouço de Software para Integração de Sistemas de Informação

MATEUS CONRAD BARCELLOS DA COSTA

Tese defendida e aprovada pela banca examinadora constituída por:

Ph. D. Rodolfo Ferreira Resende – Orientador
Universidade Federal de Minas Gerais

Ph. D. Clarindo Isaías Pereira da Silva e Pádua
Universidade Federal de Minas Gerais

Ph. D. Roberto da Silva Bigonha
Universidade Federal de Minas Gerais

Dr. José Palazzo Moreira de Oliveira
Universidade Federal do Rio Grande do Sul

Dr. Paulo Cesar Masiero
Universidade de São Paulo

Dr. Ricardo de Oliveira Falbo
Universidade Federal do Espírito Santo

Belo Horizonte, 07 de março de 2008

Agradecimentos

Antes e durante meu doutoramento tive a sorte de estar sempre acompanhado de pessoas formidáveis, me ensinando, auxiliando, incentivando e apontando direções e erros. Assim, inicialmente, gostaria de expressar meus sinceros agradecimentos a todas estas pessoas, professores, colegas, amigos e familiares que me apoiaram, seja de maneira direta ou, simplesmente, com os votos de sorte à essa empreitada.

Em particular, gostaria de agradecer:

Ao professor Rodolfo Ferreira Resende, excelente orientador e um grande amigo nessa jornada. Foi um grande privilégio para mim ter sido seu orientado. O professor Rodolfo soube perceber com clareza quais problemas realmente importavam e me convencer disso.

Aos professores Paulo Masiero, José Palazzo, Ricardo Falbo, Roberto Bigonha e Clarindo Pádua, por terem feito parte de minha banca e pelas valiosas contribuições que deram ao trabalho.

Ao professor Antônio Otávio Fernandes, que apoiou meu trabalho desde o início, oferecendo toda a infra-estrutura necessária para a realização do mesmo no Laboratório de Engenharia de Computadores (LECOM) do DCC-UFMG.

Aos demais professores do DCC-UFMG, pelo excelente ambiente acadêmico que lá encontrei e pelos ensinamentos obtidos nas disciplinas e nos diversos outros momentos vividos no DCC.

Aos funcionários do DCC-UFMG, pela atenção com que trataram minhas solicitações e necessidades como aluno de doutorado.

Aos colegas e amigos do LECOM pelos bons momentos vividos no Laboratório. Aos colegas da turma de mestrado e doutorado de 2002 do DCC-UFMG pelo companheirismo na realização das disciplinas. Aos colegas e amigos Marcelo Segatto e Eduardo Nakamura pela boa colaboração na produção dos artigos. Aos meus co-orientados de Projeto Orientado de Computação do DCC-UFMG, Admar e Alysson, pela dedicação e contribuições dadas ao método BASS.

Aos então diretores da Fundação São João Batista, Divaldo Crevelin e Fernando Mai, e à diretora da Faculdade de Aracruz, Dalva Cuzzuol, pela criação do programa de capacitação que apoiou os dois anos iniciais do doutorado.

Ao diretor da UnED-Serra do CEFET-ES, Ademar Stange e à coordenadora de Informática Isaura Nobre, pelo apoio dado a partir de 2006. Aos colegas professores do CEFET-ES pelo apoio e incentivo. Em particular aos professores Edilson e João, que por vezes assumiram tarefas minhas para que eu pudesse estar me dedicando ao doutorado. Ao CEFET-ES e à FUNCEFETES pelo apoio financeiro recebido para a apresentação de artigos.

À minha mãe, Edna, pelo amor e preocupação a mim dedicados desde sempre. Aos meus irmãos e ao meu finado pai, Euzébio, pelo empenho à minha educação e à formação dos valores que fizeram com que eu chegasse até aqui. Agradeço, em particular, ao meu irmão Maucrice, que sempre me ajudou, quando precisei, antes e durante o doutorado.

À minha família por parte de minha esposa, Eulália, Fabiana e Márcio, pelos incentivos ao doutorado e, principalmente, pela tranqüilidade que me proporcionaram com os cuidados

dedicados à Laura e à Luciana nas minhas muitas ausências de casa.

Iniciar e terminar essa tese só foi possível devido o amor e união entre mim e minha esposa Luciana. Não faltou por parte dela a resignação necessária para atingirmos esse propósito. Junto a mim, Luciana sustentou os momentos difíceis e compartilhou as alegrias verdadeiramente. Assim, sou mais que agradecido a ela.

Durante esse caminho, que não foi fácil, que normalmente não é fácil para ninguém, a vida nos presenteou com nossa filha Laura. Ser pai no meio de um doutorado é uma experiência, no mínimo, imprevisível. Felizmente para nós, com a presença de Laurinha, sempre tão alegre e afetuosa, tudo ficou melhor e mais fácil.

Resumo

Nesta tese apresentamos o arcabouço de software BASS: Business Application Support through Software Services. A finalidade do BASS é auxiliar no desenvolvimento de soluções de software para a integração de sistemas de informação. O desenvolvimento do BASS originou-se do estudo da relação entre requisitos e tecnologias no domínio de Integração de Sistemas de Informação. O desenvolvimento do BASS teve uma dimensão de tecnologia (o arcabouço BASS) e uma dimensão de processo de software, com o desenvolvimento de elementos de processo necessários para a utilização efetiva do arcabouço. Dentre os benefícios obtidos com o BASS está a simplificação do desenvolvimento e a diminuição de esforço em atividades de desenho e implementação.

Abstract

This thesis presents the software framework BASS: Business Application Support through Software Services. BASS aims at supporting Information Systems Integration. BASS development originated from our investigation about the interplay between requirements and technology in the Information Systems Integration domain. Such development encompassed a technological dimension (the BASS framework) and a Software Process dimension, with the definition of process elements needed for the effective use of the framework. Amongst the benefits achieved with BASS are the simplification of how to deal with complex technologies and the reduction of design and implementation efforts in the development of Information Systems Integration software solutions.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A evolução das tecnologias tem permitido o desenvolvimento de produtos de software cada vez mais avançados e que resolvem problemas de complexidade cada vez maior. No entanto, apenas essa evolução não é suficiente para garantir a qualidade e produtividade em organizações desenvolvedoras de software. Conforme o triângulo da qualidade e produtividade ilustrado na Figura 1.1 (Jalote, 2000), esses fatores dependem de três elementos: processo, tecnologia e pessoas. Diversos aspectos relacionados a esses três elementos necessitam ser tratados conjuntamente mantendo-os constantemente alinhados uns com os outros. Esse alinhamento consiste, por exemplo, em manter definidos, no processo de software, os aspectos de tecnologia que são relevantes para o desenvolvimento.

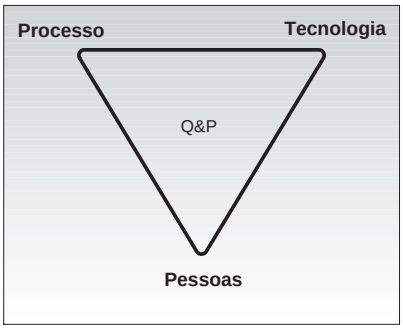


Figura 1.1: Triângulo da Qualidade e Produtividade.

Para abordar os aspectos de tecnologias, são definidas atividades de *gerenciamento de tecnologias* no processo de software. O gerenciamento de tecnologias compreende um conjunto de atividades relacionadas à seleção, desenvolvimento, organização e manutenção de tecnologias a serem utilizadas no desenvolvimento de um ou mais produtos de software. Conforme previsto em normas, padrões e modelos de maturidade (ISO/IEC, 2008, 2000; SEI, 2006), a definição dessas atividades e de seus elementos de suporte deve, idealmente, ser orientada pela natureza dos projetos a que se destina o processo de software. Processos de software deveriam, portanto, fornecer um arcabouço para o gerenciamento de tecnologias considerando o conjunto de requisitos dos projetos a que se destinam. Entretanto, diante da complexidade do atual cenário tecnológico e do aumento da complexidade dos requisitos envolvidos no desenvolvimento de software, o estabelecimento desse arcabouço tem se tornado uma tarefa de difícil execução (Xia e Lee, 2005).

Conforme discutiremos na próxima seção, iniciamos nosso trabalho de tese investigando a relação entre requisitos e tecnologias com o objetivo de estabelecer elementos de processo cuja função seria acomodar tecnologias durante as diversas fases do desenvolvimento. Em

função da complexidade desse objetivo reduzimos o escopo para a relação entre processos e tecnologia no domínio de integração de sistemas de informação. Durante a investigação desenvolvemos um arcabouço de software para integração de sistemas de informação que se tornou a nossa principal contribuição. O desenvolvimento desse arcabouço, denominado *Business Application Development through Software Services* (BASS) pôde nos mostrar que a relação entre requisitos e tecnologias, mesmo em um caso particular, é um problema bastante complexo e aqui damos nosso depoimento: certamente deve ser um problema de extrema complexidade no caso mais geral.

1.1 Objetivo do Arcabouço BASS

A integração de sistemas de informação ocorre quando há a necessidade de combinar dois ou mais sistemas de informação. Essa integração deve ocorrer, por exemplo, quando uma agência de viagens, por meio de seu sistema de informação, deseja realizar reservas em um hotel com a condição de que essas reservas sejam realizadas e registradas não só em seu sistema mas também no sistema de informação do hotel. A integração desses sistemas pode ser feita, por exemplo, por meio de tecnologias baseadas na Computação Orientada a Serviços (Papazoglou e Georgakopoulos, 2003). Nesse caso, o sistema de informação do hotel pode expor as funcionalidades necessárias para a realização de uma reserva por meio de um componente de software denominado serviço, acessível a outros sistemas interligados por meio de uma rede. Para a combinação dos dois sistemas, o sistema da agência de viagens deve, por sua vez, prover uma solução de integração baseada nas funcionalidades disponibilizadas pelo serviço do hotel.

Em uma integração envolvendo dois sistemas, o sistema que disponibiliza o serviço é denominado provedor. Já o sistema que utiliza o serviço é denominado cliente. O objetivo do arcabouço BASS é auxiliar no desenvolvimento da solução de integração de sistemas clientes de serviços. No exemplo da agência de viagens o BASS seria utilizado para auxiliar no desenvolvimento da solução de integração do sistema da agência de viagens, considerando que as funcionalidades do hotel foram disponibilizadas por meio de serviços.

Considerando o uso de tecnologias para a Computação Orientada a Serviços, diversos desafios devem ser considerados no desenvolvimento da solução de integração de sistemas clientes. Dentre esses desafios estão, por exemplo, o suporte a integrações dinâmicas nas quais parâmetros e mesmo participantes da integração são definidos em tempo de execução das aplicações de software envolvidas; o estabelecimento da integração por meio de modelos baseados em processos de negócio; o suporte aos diversos aspectos envolvendo transações de negócio e os problemas relacionados a aspectos não funcionais da integração. Com o BASS abordamos esses problemas considerando uma solução integrada para os mesmos, por meio do Arcabouço, e, consideramos aspectos de desenvolvimento dessa solução.

1.2 Motivação do Trabalho

Com a presença de novas tecnologias para a realização da integração de sistemas de informação, surgiram também novas demandas expectativas. Com a consolidação da Internet e da WWW como plataforma de desenvolvimento de aplicações, deu-se o início de um novo ciclo no Desenvolvimento de Software. Esse ciclo caracteriza-se pela necessidade de atender a novos modelos de negócio, em particular modelos de integração entre organizações (Smith e Fingar, 2006). A popularização da Computação Orientada a Serviços, por exemplo, criou novas possibilidades e expectativas para a realização de processos de negócios inter-organizacionais,

aumentando de certa forma a complexidade dos requisitos de integração (Trastour et al., 2002; Irani et al., 2003).

Nesse cenário, dois importantes fatores para o desenvolvimento de software são: a diversidade e dinamismo das tecnologias e a complexidade dos requisitos originados destes novos modelos de negócio. A complexidade dos requisitos origina-se da: (a) complexidade das funcionalidades das aplicações necessárias para suportar esses modelos de negócio como, por exemplo, suporte a transações de negócio, (b) das restrições de caráter não funcional, por exemplo, escalabilidade e segurança, e (c) de restrições de projeto como, por exemplo, tempo e custo (Datta e van Engelen, 2006).

Para suportar o desenvolvimento de software neste cenário, idealmente um processo de desenvolvimento deveria ser capaz de possuir um conjunto pré-definido de tecnologias para as quais já teria todos os aspectos relevantes especificados. Esse processo também deveria ser capaz de aceitar a definição de novas tecnologias onde todos os aspectos relevantes seriam considerados sendo, então, adequadamente especificados.

Nosso objetivo inicial foi identificar elementos da relação entre tecnologias e processo que pudessem facilitar a evolução de um conjunto de requisitos em uma solução de software. Esperávamos definir métodos e técnicas para o estabelecimento de atributos do processo, relacionados às tecnologias, em função de um conjunto de requisitos. Com o desenvolvimento do trabalho, o escopo desta investigação mostrou-se muito amplo. Focalizamos a investigação nas tecnologias de integração de sistemas de informação e na inserção dessas tecnologias em processos de software utilizando arcabouços de software.

Arcabouços de software são elementos genéricos que especificam aspectos relevantes de um domínio de aplicação e que podem ser empregados no desenvolvimento de soluções específicas com o objetivo de aumentar a qualidade do software e reduzir esforços de desenvolvimento (Fayad et al., 1999). Diferentemente de componentes de software e bibliotecas, arcabouços contribuem para a especificação arquitetural das aplicações. Arquiteturas de software, por sua vez, constituem elementos de ligação entre requisitos e modelos de implementação da solução (Shaw e Garlan, 1996). Dessa forma o uso de arcabouços pode interferir e contribuir em diversos aspectos do desenvolvimento.

Como resultado desta investigação, desenvolvemos o arcabouço BASS. No desenvolvimento do BASS focalizamos: (a) a integração de sistemas de informação dirigida por processos de negócio e (b) o emprego de tecnologias baseadas na Computação Orientada a Serviços. Neste contexto, uma importante questão é definir *quais métodos de desenvolvimento são apropriados para projetos de integração baseada em serviços.* (Lübke, 2007). Com o desenvolvimento do BASS, pudemos abordar essa questão, pois, para a sua utilização, estabelecemos aspectos de desenvolvimento considerando tecnologias baseadas na Computação Orientada a Serviços.

1.3 Escopo do Trabalho e Contribuições

Conforme já discutido, limitamos o escopo de nossa investigação ao domínio de Sistemas de Informação e, em particular, à Integração de Sistemas de Informação. Com relação ao Desenvolvimento de Software, considerando a norma ISO 12207, o desenvolvimento do arcabouço BASS compreende uma atividade relacionada a *processos de Infra-Estrutura* com o objetivo de definir artefatos e procedimentos para serem aplicados em *processos de Desenvolvimento*, em particular em atividades de *Análise, Desenho e Implementação* desses processos.

Antes de focalizarmos no desenvolvimento do arcabouço BASS, na fase inicial do trabalho, apresentamos uma organização das tecnologias de desenvolvimento para suporte a Transações de Comércio Eletrônico (Costa et al., 2004a). A principal contribuição dessa etapa foi apresentar uma organização de modelos conceituais, lógicos e físicos, que permite um melhor

mapeamento entre modelos de transações e modelos de desenho e implementação da solução de software para essas transações.

Posteriormente apresentamos um modelo arquitetural para suporte à integração de sistemas de informação baseada em Serviços Web e utilizando Aspectos (Costa et al., 2004b). A partir dessa etapa focalizamos nossa investigação no desenvolvimento de um arcabouço de software para a integração de sistemas de informação. Apresentamos o arcabouço BASS como modelo físico para a implementação da integração de sistemas de informação (Costa et al., 2007). Durante o desenvolvimento do BASS diversos exemplos foram implementados com intuito de testar a conformidade do BASS com um conjunto de necessidades de integração. Com esses exemplos, pudemos avaliar inclusive a utilização do BASS no desenvolvimento da integração envolvendo serviços comercialmente difundidos e em produção.

O arcabouço BASS visou ao atendimento de requisitos de produto e de requisitos de desenvolvimento da integração de sistemas de informação. Durante o desenvolvimento do BASS definimos quais seriam esses requisitos e os apresentamos por meio de um catálogo de requisitos de arcabouços de software para integração de sistemas de informação (Costa et al., 2008b).

Elaboramos, em conjunto com o BASS, elementos de processo que facilitassem sua utilização em processos de desenvolvimento. Considerando a norma ISO 12207, esses elementos são dedicados às atividades de Análise de Requisitos de Software, Desenho da Arquitetura de Software, Desenho Detalhado de Software e Codificação. Denominamos a aplicação desses elementos de processo e do arcabouço BASS de *método BASS*. Alguns aspectos desses elementos foram apresentados de forma preliminar (Costa et al., 2007, 2008b), sendo contudo organizados como um método único em Costa et al. (2008a).

Para avaliar e caracterizar o uso do método BASS realizamos estudos de caso e experimentos com alunos da disciplina Engenharia de Software, normalmente cursada por alunos do quarto ano do curso de graduação em Ciência da Computação, do Departamento de Ciência da Computação da Universidade Federal de Minas Gerais (DCC/UFMG). Nos experimentos pudemos comparar o desenvolvimento da solução de integração utilizando o método BASS com um método *ad hoc*, sem diretivas específicas para o domínio de Integração de Sistemas de Informação. Essa comparação permitiu verificar a efetividade da utilização do BASS com relação à redução de esforço de desenvolvimento.

1.4 Organização da Tese

Os demais capítulos desta tese estão organizados da seguinte maneira:

Capítulo 2: Integração de Sistemas de Informação (ISI). Apresentamos o referencial teórico e aspectos do domínio de aplicação e de desenvolvimento que foram necessários para a realização do trabalho. Focalizamos a integração dirigida por processos de negócio e tecnologias baseadas na Computação Orientada a Serviços. Ao final do Capítulo, discutimos aspectos do desenvolvimento de software para ISI.

Capítulo 3: Modelagem e Implementação da Integração de Sistemas de Informação. Apresentamos, inicialmente, um catálogo de requisitos para arcabouços de software para ISI. Posteriomente, apresentamos uma organização para o desenvolvimento de software para ISI. Apresentamos o arcabouço BASS e discutimos aspectos de sua concepção e utilização. Apresentamos também os estudos experimentais realizados.

Capítulo 4: Aspectos de Implementação do Arcabouço BASS. Apresentamos aspectos relevantes da implementação do arcabouço BASS.

Capítulo 5: Conclusão. Concluímos o texto da tese discutindo as contribuições e oportunidades de investigação futura.

Capítulo 2

Integração de Sistemas de Informação

Neste capítulo apresentamos o referencial teórico utilizado na Tese. Esse referencial compreende aspectos da Integração de Sistemas de Informação relacionados à integração baseada em processos de negócio, às tecnologias de integração e a aspectos de desenvolvimento de software da solução de integração. Alguns pontos apresentados são aplicados diretamente na definição do problema da Integração de Sistemas de Informação e nas soluções adotadas. Exemplos desses pontos são Processos de Negócio, Serviços Web e Arcabouços de Software. Outros pontos fazem parte da definição do problema e das soluções de maneira indireta. Exemplos desses pontos são Requisitos e Processos de Software.

2.1 Introdução

A Integração de Sistemas de Informação (ISI) consiste na integração das funcionalidades disponibilizadas por estes sistemas com o objetivo de auxiliar a realização de atividades conjuntas entre organizações possivelmente autônomas. Um exemplo simples de ISI é entre um cliente e um fornecedor, ilustrado na Figura 2.1. A ISI neste caso implica em transformar a ação de uma compra no sistema de informação cliente em uma ação de venda no sistema de informação fornecedor, seguindo rigorosamente as regras de compra do cliente e as regras de venda do fornecedor. Para tanto, as funcionalidades Compra e Venda (elipses) devem ser integradas.

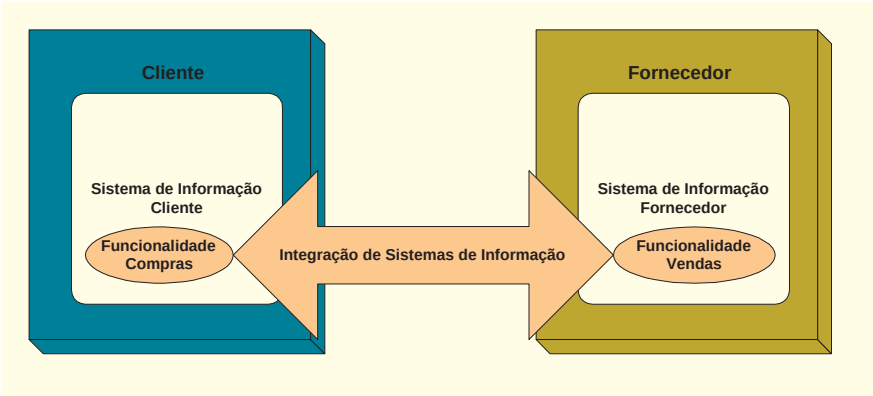


Figura 2.1: Integração de Sistemas de Informação entre Cliente e Fornecedor.

Linthicum (2000) discute os seguintes tipos de abordagem para integração entre organizações:

Orientado a Dados. Esse tipo envolve o processo de se extrair informação de um banco de

dados, processar essa informação quando necessário, e atualizá-la em outro banco de dados pertencente a outra organização. Embora esse processo possa parecer simples, a manipulação de dados necessária para estabelecer a integração pode envolver grandes conjuntos de dados em um conjunto diversificado e grande de bancos de dados. Além disso, o processo de transferência de informação freqüentemente exige a aplicação de regras de negócio. Na integração orientada a dados, aplicações que suportam os sistemas de informação não sofrem modificações e permanecem não integradas;

Orientado a Interface de Aplicações. Nesse tipo, a integração baseia-se na exposição de funções de negócio ou mecanismos de acesso a dados, por meio de interfaces de aplicações. A integração orientada a interfaces de aplicações tem sido aplicada por pacotes de aplicações comerciais como os pacotes da SAP, PeopleSoft e Baan. Diferenças do formato empregado nessas interfaces dificultam a integração dos mesmos. Para integrar tais sistemas uns com os outros, mesmo no contexto de uma única organização, é necessário usar as interfaces para obter informações, colocá-las no formato da aplicação destino e posteriormente transmiti-las (para a aplicação destino);

Orientado a Métodos. Esse tipo de integração visa ao compartilhamento das implementações de funcionalidades e não apenas das interfaces. Por exemplo, o código de um método para realizar uma compra pode ser compartilhado entre diversas aplicações, dentro ou fora da organização. Tecnologias para compartilhamento de métodos incluem, por exemplo, Objetos Distribuídos, Servidores de Aplicação e Processamento de Transações;

Orientado a Interface de Usuário. Nesse tipo, a integração ocorre pela representação do acesso a informações, pertencentes a diferentes organizações, em um mesmo local. Na Web esse tipo de integração normalmente é estabelecido por um mecanismo denominado portal. Neste tipo de integração, organizações evitam a complexidade e custo da integração de aplicações provendo a integração no nível de interface do usuário. Por exemplo, na compra de uma passagem com cartão de crédito, no portal de uma companhia aérea, a autorização de compra no cartão pode ser dirigida diretamente à operadora do cartão;

Orientado a Processos de Negócios. Esse tipo estabelece uma camada de abstração de processos no topo das aplicações e tecnologias utilizadas para a realização da ISI. Para as organizações envolvidas, essa camada provê uma visão de negócios e dos fluxos de informação de negócio da integração. No nível da integração de processos de negócio, os elementos da ISI são conceitos do domínio de aplicação da integração, por exemplo, pessoas, pedidos e empresas.

Além dos tipos de integração anteriores ao ano 2000, mencionados por Linthicum, tomou maior evidência, mais recentemente, a Orientação a Serviços ou Computação Orientada a Serviços. Semelhante à integração orientada a métodos, a integração orientado a serviços visa à exposição das funções de negócio definidas no contexto de uma organização por meio de componentes de software denominados serviços. Serviços são disponibilizados por meio de interfaces padronizadas e acessíveis a outras aplicações por meio de uma rede. A integração Orientada a Serviços estabelece arquiteturas baseada na Arquitetura Orientada a Serviços – *Service Oriented Architecture* — SOA (W3C, 2002). Nessas arquiteturas, clientes, provedores e registros de serviços visam a busca, localização, ligação e execução dinâmica de serviços.

A realização da ISI requer normalmente o desenvolvimento de soluções de software para os requisitos de integração. O desenvolvimento dessas soluções tem sido influenciado pela infra-estrutura de tecnologias de integração disponível. Exemplos de elementos dessa infra-estrutura são as redes de comunicação, protocolos e middleware de integração.

Antes do surgimento da Internet e da WWW, a infra-estrutura de tecnologias de integração caracteriza-se pelo custo alto e pela pouca flexibilidade. Um exemplo típico dessas tecnologias é o Intercâmbio Eletrônico de Documentos (Electronic Document Interchange – EDI) (Ramamurthy e Premkumar, 1995). As características dessas tecnologias foram determinantes para que a ISI neste período ficasse restrita a poucos nichos de mercado e criasse longas relações comerciais entre as organizações envolvidas. Essas características contrapunham-se ao desejo das organizações de manter relações de negócios de forma flexível, sem imposições vindas de opções tecnológicas (Trastour et al., 2002).

Com a consolidação da Internet e da WWW como plataforma alvo para o desenvolvimento de aplicações, modelos de integração mais flexíveis passaram a se tornar efetivamente implementáveis e economicamente viáveis. Nesses novos modelos, atividades como a escolha dos parceiros comerciais e a definição das regras de colaboração entre esses passaram a ser mais orientadas por interesses comerciais, tornaram-se mais flexíveis e mais independentes de aspectos tecnológicos (Irani et al., 2003).

Resumidamente, com o surgimento de tecnologias de menor custo, flexíveis e padronizadas, como, por exemplo, as tecnologias baseadas na Computação Orientada a Serviços, a Integração de Processos passou a se tornar viável. Em nosso trabalho, focalizamos a investigação no desenvolvimento da ISI considerando a Integração de Processos e o uso de tecnologias baseadas na Computação Orientada a Serviços. No restante deste capítulo discutimos mais profundamente estes elementos.

2.2 Integração de Processos

Um processo de negócio é uma especificação de um conjunto estruturado de atividades projetado para produzir um resultado específico. O fluxo de execução das atividades de um processo de negócio usualmente modifica o estado de um conjunto de dados ou de um produto (Davenport, 2000). Os processos de negócio de interesse da ISI envolvem mais de uma organização. Para discutir a ISI, considerando esses processos de negócio, Hasselbring (2000) apresenta uma decomposição vertical das organizações em três níveis: o nível da Arquitetura de Negócios, da Arquitetura de Aplicações e da Arquitetura de Tecnologias. A Figura 2.2 apresenta esta organização.

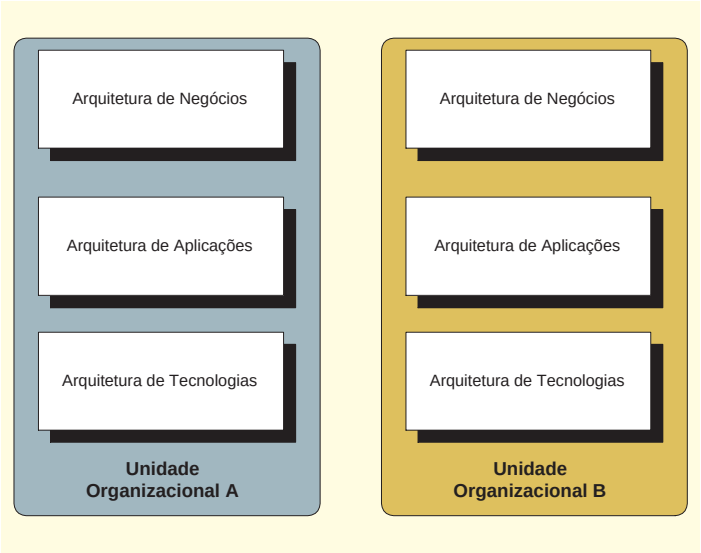


Figura 2.2: Decomposição Vertical das Organizações.

No nível da Arquitetura de Negócios define-se a estrutura organizacional, os fluxos de trabalho e regras de negócio dos processos existentes na organização. A Arquitetura de Negócios é estabelecida no nível conceitual, em termos de conceitos relevantes para o domínio da organização. No nível da Arquitetura de Aplicações define-se a implementação concreta dos elementos descritos na Arquitetura de Negócios em termos de aplicações de software. Nesse nível, o objetivo central é prover a ligação entre o domínio da aplicação descrito na arquitetura dos negócios e as soluções técnicas descritas no nível da Arquitetura de Tecnologias. O nível da Arquitetura de Tecnologias envolve a infra-estrutura de informação e comunicação da organização. Neste nível, definem-se soluções de Tecnologia de Informação capazes de atender aos requisitos definidos com base na Arquitetura de Negócio.

Muito da investigação em Sistemas de Informação é dedicada a preencher o abismo existente entre negócios e tecnologias. Na ISI esse também é um dos maiores objetivos e requer uma visão interdisciplinar dos elementos do domínio de aplicação e de Tecnologia de Informação (Hasselbring, 2000). Com a necessidade cada vez maior das organizações colaborarem entre si, por exemplo, para realização de Comércio Eletrônico entre Empresas, é necessária uma visão integrada dos processos de negócio das organizações envolvidas em colaborações. A organização de processos de negócio em termos de processos privados, públicos e colaborativos (BPMI, 2004; Schulz e Orlowska, 2004; Roser e Bauer, 2005), oferece essa visão. A figura 2.3 ilustra esta organização. Os retângulos tridimensionais contendo diagramas de atividade ilustrativos representam dois sistemas de informação. Esses digramas representam os processos de negócio internos à organização e suportados pelo seu respectivo sistema de informação. Esses processos são denominados *processos privados*.

Os retângulos abaulados, contendo também diagramas de atividade, representam processos de negócio desempenhados em colaboração. Esses processos são denominados *processos colaborativos*. Para o planejamento desses processos consideram-se as interfaces de interação do sistema com elementos externos, representadas por uma linha com um terminador circular (símbolo de interface da UML). Essas interfaces e as regras de negócio da sua utilização definem os *processos públicos* de cada sistema. Cada sistema necessita dispor de uma solução de software que dê suporte a seus processos privados e implemente seus processos públicos de maneira consistente com as colaborações a serem realizadas entre as organizações.

Temos então as seguintes definições (Roser e Bauer, 2005):

Processo Privado: Processo interno à organização. Usados para descrever atividades internas à organização por meio de fluxos de trabalho cujos elementos ativos são internos à organização.

Processo Público: Processo que descreve possíveis interações entre os processos privados de uma organização com elementos ou organizações externos.

Processo Colaborativo: Processo que descreve colaborações entre organizações. Essas colaborações são descritas utilizando os elementos definidos nos processos públicos das respectivas organizações.

A separação da ISI em termos de processos privados, públicos e colaborativos permite simplificar o desenvolvimento da arquitetura de processos. Entretanto, no nível da Arquitetura de Aplicações (Figura 2.2), a solução de software para esses processos deve ser integrada. Aplicações que suportam os processos privados devem suportar também as interfaces correspondentes aos processos públicos e as colaborações que envolvem esses processos. As ações e respostas a ações que ocorrem em uma colaboração são em geral decorrentes de uma atividade interna de uma das organizações envolvidas. Dessa forma, embora possa haver componentes

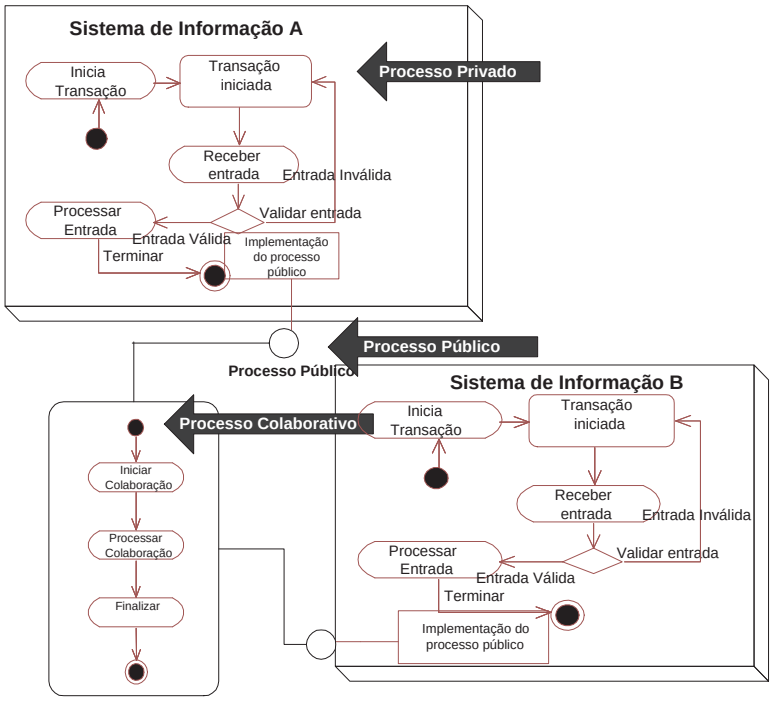


Figura 2.3: Sistemas de Informação e Processos de Negócio (Roser e Bauer, 2005).

específicos dessas arquiteturas dedicados a cada categoria de processo, esses componentes devem estar inter-relacionados. Freqüentemente, uma única aplicação é responsável por suportar os processos privados e públicos.

O desenvolvimento de software para ISI baseada na Integração de Processos apresenta ainda grandes dificuldades. Neste modelo de integração, a arquitetura das aplicações deve prover soluções de software considerando tanto aspectos funcionais, ou seja, aspectos da lógica de negócios dos processos, quanto aspectos não funcionais, por exemplo, segurança e desempenho. Dentre esses aspectos destacam-se o suporte à adaptação de processo de negócio (Zeng et al., 2002), o suporte a transações de negócio (Dalal et al., 2003), a mediação de vocabulários (Dey et al., 2002) e a qualidade de serviços (Aissi et al., 2002). Nas seções seguintes discutiremos esses aspectos.

2.2.1 Adaptação de Processos de Negócio

Estabelecer a integração no nível da Arquitetura de Processos compreende um grande desafio para as organizações. Processos de negócio legados e mudanças contínuas em regras de negócio estão entre as principais dificuldades para a obtenção de uma integração plena entre as arquiteturas de negócio de organizações autônomas (De Michelis et al., 1998). Embora haja esforços de padronização no nível da Arquitetura de Negócios, parte da solução para necessidades de adaptação entre processos de negócio internos e externos a uma organização reside no nível da Arquitetura de Aplicações.

Na Arquitetura de Aplicações, processos privados e públicos devem ser suportados de forma a permitir que a informação possa fluir entre o contexto interno e externo da organização. Ao mesmo tempo, esses processos devem manter um certo grau de isolamento, por exemplo, para garantir privacidade de elementos que só interessem ao contexto interno da organização (Bussler, 2001).

Para suportar os processos públicos e prover o comportamento esperado na especificação dos processos colaborativos, a solução de software deve ser capaz de determinar dinamicamente, por meio de seus processos privados e parâmetros externos, como interagir com outros sistemas participantes. Essa solução deve ser capaz de determinar como a execução de uma atividade deve ocorrer, por exemplo, em termos de parceiros de negócio, operações, argumentos e valores dessas operações.

Discutiremos o suporte à adaptação de processos de negócio e outros aspectos da integração de processos utilizando a adaptação de um exemplo apresentado por Bioernstad et al. (2006). Esse exemplo descreve o processo de compra de um produto, envolvendo três organizações: um cliente, uma loja e um fornecedor. A loja desempenha o papel de agente intermediário entre o cliente e o fornecedor.

Utilizaremos modelos da UML 1.4 para ilustrar os diversos aspectos desse processo. Na Figura 2.4 apresentamos as interações necessárias para a realização de um pedido de compra. Essas interações definem o processo colaborativo que deve ser suportado. Inicialmente, as interações visam determinar a disponibilidade do produto: O cliente envia um pedido à loja (1). A loja checa disponibilidade com o fornecedor e envia a resposta ao cliente (2,3,4). A loja confirma o pedido e o fornecedor informa que o pedido está confirmado (5,6). O cliente paga ou cancela o pedido e a loja confirma o envio ou cancela o pedido junto ao fornecedor (7,8). O cliente informa o recebimento do produto à loja e esta finaliza a transação junto ao fornecedor (9,10).

Para a realização das colaborações necessárias, cada participante expõe seus processos públicos. Os processos públicos são abstrações onde apenas os elementos de interação com os parceiros de negócio são apresentados. A Figura 2.5 ilustra os processos públicos de cada um dos participantes. Por meio desses processos é possível identificar as interfaces que devem ser disponibilizadas e utilizadas por cada participante.

Os processos processos privados de cada participante atuam expressivamente na realização de colaborações. Esses processos definem atividades que são internas aos sistemas participantes e suportam também as atividades descritas nos processos públicos. As Figuras 2.6, 2.7 e 2.8 representam respectivamente, as especificações dos processos privados do cliente, da loja e do fornecedor.

A solução de software para integração entre cliente, loja e fornecedor deve garantir a realização dos processos privados e colaborações de maneira transparente e dinâmica. Por exemplo, a realização do processo privado do Cliente envolve o processamento da atividade interna Especificar Compra, que produz o objeto Pedido, de acordo com as regras de negócio internas do cliente. Em seguida, a transação requer a atividade Fazer Pedido, que deve ser definida de acordo com o processo público disponibilizado pela loja. Para a realização da atividade Fazer Pedido diversos aspectos da interação devem ser definidos. Por exemplo, no caso da atividade ser realizada utilizando operações de um serviço disponibilizado pela loja, esses aspectos compreendem o endereço do serviço, as operações do serviço que devem ser executadas e os aspectos sintáticos e semânticos dessas operações. Para a execução da atividade Fazer Pedido, a solução de software deve ser capaz de determinar esses aspectos dinamicamente, em função das atividades do processo privado que está sendo executado. Idealmente a determinação desses aspectos deve ocorrer sem a necessidade de soluções específicas para cada colaboração que o cliente possa vir a realizar.

2.2.2 Transações de Negócio

Uma transação de negócio é uma mudança consistente no estado de um negócio dirigida por uma função de negócio ou objetivo (Papazoglou, 2003). Um processo de negócio é composto por uma ou mais transações de negócio. Por exemplo, o Pedido de Compra de nosso exemplo

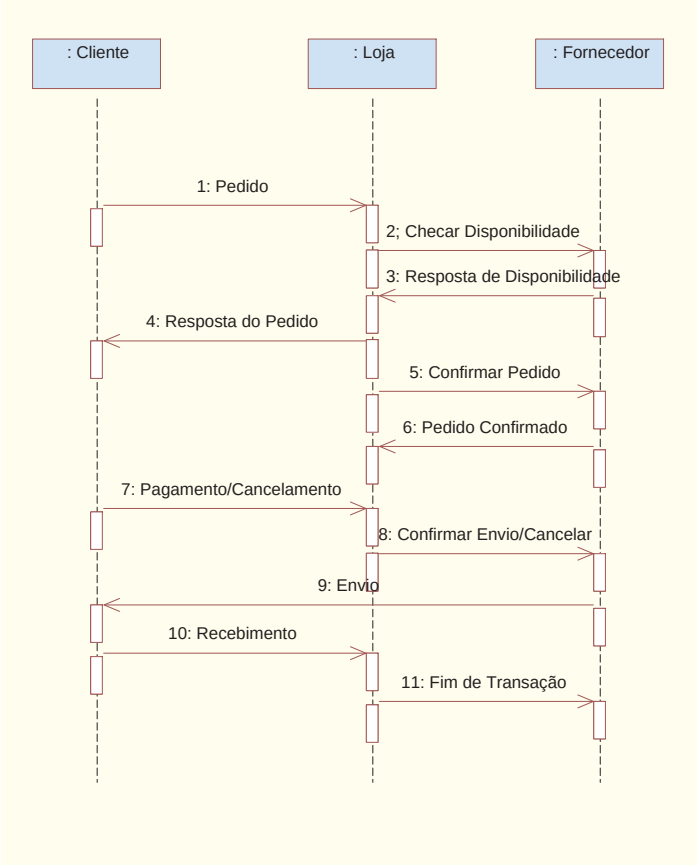


Figura 2.4: Processo Colaborativo do Pedido de Compra.

pode ser dividido em duas transações: Consulta e Compra. A Consulta envolve as atividades que vão até a atividade Receber Resposta de Disponibilidade no Cliente. A Compra compreende as atividades posteriores.

Em contextos envolvendo múltiplos participantes autônomos, por exemplo, múltiplas organizações, uma transação de negócio implica na realização de interações entre esses participantes visando atingir algum objetivo compartilhado (Papazoglou, 2003). Na transação de Compra, o objetivo é a aquisição do produto, sob o ponto de vista do cliente, e de uma venda do produto sob o ponto de vista da loja e do fornecedor.

Transações de Negócio, normalmente, não atendem aos critérios de transações ACID (Gray, 1981): Atomicidade, Consistência, Isolamento e Durabilidade (Papazoglou, 2003). Diferentemente do contexto de sistemas de bancos de dados, onde as transações possuem tempo de duração curto (que permite a reserva dos recursos manipulados, por exemplo, tabelas), transações de negócio são geralmente de longa duração. A execução dessas transações envolve sistemas autônomos e possivelmente geograficamente dispersos, com conexões de rede sujeitas a falhas. Neste contexto, diversos mecanismos devem ser considerados.

Schmit. e Dustdar (2005) apresentam os seguintes requisitos e desafios na realização dessas transações:

Suporte a caminhos de execução alternativos. Transações de negócio podem possuir diferentes caminhos de execução que apresentem diferentes resultados aceitáveis. O suporte a essas transações deve prever esses aspectos. Para tanto, devem ser oferecidos mecanismos para se escolher dinamicamente entre esses possíveis caminhos e estabelecer

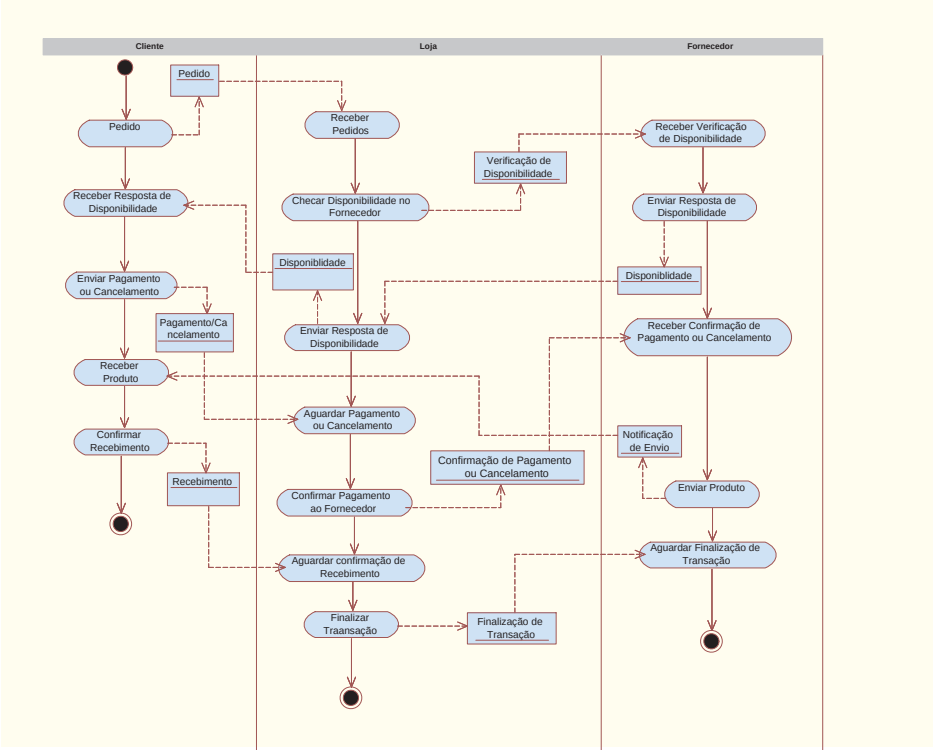


Figura 2.5: Processos Públicos do Pedido de Compra.

uma seqüência de execução aceitável. Exemplos desses mecanismos incluem desvios condicionais, estruturas de repetição que suportem a execução de um conjunto de caminhos alternativos e mecanismos para a execução desses caminhos em paralelo (Curbera et al., 2003).

Suporte a transações de múltiplas fases. Transações de Negócio complexas podem ser organizadas em múltiplas fases mais simples. Cada fase compreende uma etapa da transação, que só faz sentido no contexto da seqüência de fases estabelecida. Fases de uma transação compartilham informação entre si e, freqüentemente, o seu processamento é determinado em função de resultados das fases anteriores. O suporte a modelos de múltiplas fases permite modularizar transações complexas, aumentando as chances de sucesso dessas transações.

Modelos de múltiplas fases permitem o estabelecimento de uma fase inicial, onde busca-se antecipadamente o estabelecimento de acordos entre os participantes envolvidos. Posteriormente, outras fases dão continuidade à realização da transação por completo. Ou seja, o suporte a múltiplas fases é importante tanto para a modelagem de negócios quanto para simplificar o desenvolvimento do suporte a essas transações.

Suporte a novos modelos de acesso a recursos compartilhados. Uma importante diferença entre transações ACID e transações de negócio é que, em geral, transações ACID são rápidas e podem bloquear o acesso de outras transações a um determinado recurso. No caso de transações de negócio, isso não é possível. Dessa forma, outros modelos de acesso a recursos compartilhados, que garantam a integridade desses recursos, devem ser definidos.

Suporte a mecanismos de compensação. Um importante requisito para transações de

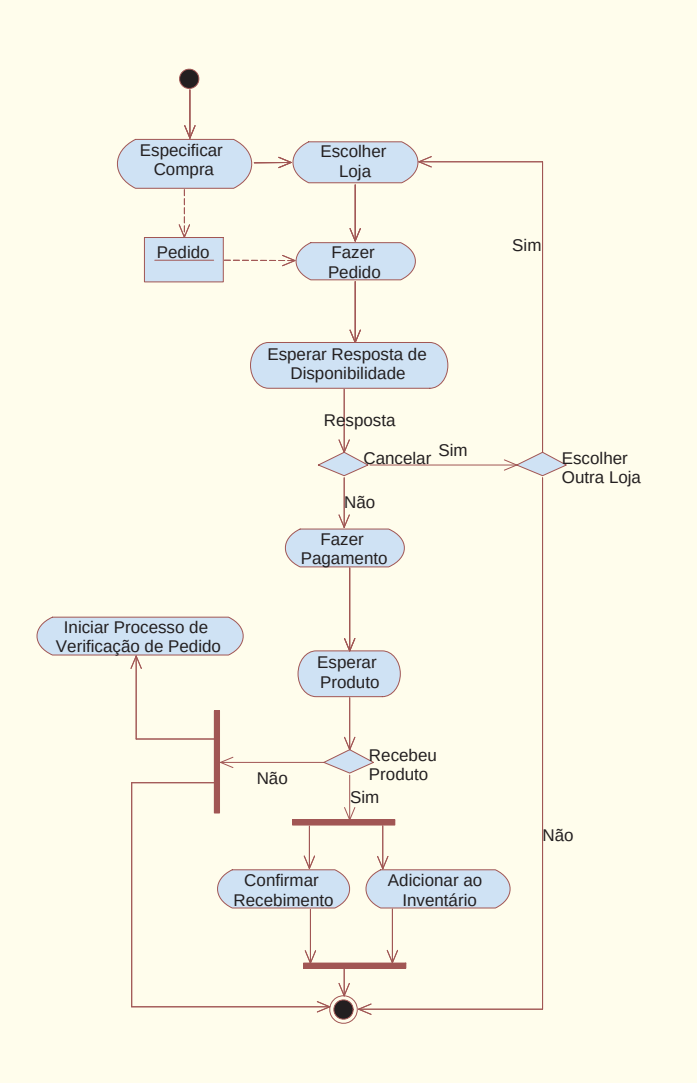


Figura 2.6: Processo Privado do Cliente.

negócio são os mecanismos de compensação, isto é, mecanismos que compensem uma ação previamente executada. Em geral a compensação ocorre quando algo decorrente de uma ação anterior sai errado. Um exemplo de compensação é a devolução de um produto que chega defeituoso às mãos do cliente, após a realização da transação de Compra. É esperado que atividades de compensação desfaçam os efeitos das atividades primárias às quais a compensação está associada (Chessell et al., 2002). Compensação também está associada ao mecanismo de restauração (*roll-back*). Por exemplo, no caso do Cancelamento do pedido junto ao Fornecedor, esse deve possuir mecanismos de compensação para liberar o produto reservado.

Suporte a temporizações. Em decorrência, principalmente da necessidade de mecanismos de Compensação, mecanismos de temporização (*timeouts*) são fundamentais no suporte a transações de negócio. No Pedido de Compra, por exemplo, em caso de devolução de produto, não seria conveniente para o fornecedor ficar esperando por devoluções por um período indefinido de tempo.

Suporte a transações hierárquicas. Transações complexas envolvendo múltiplas organi-

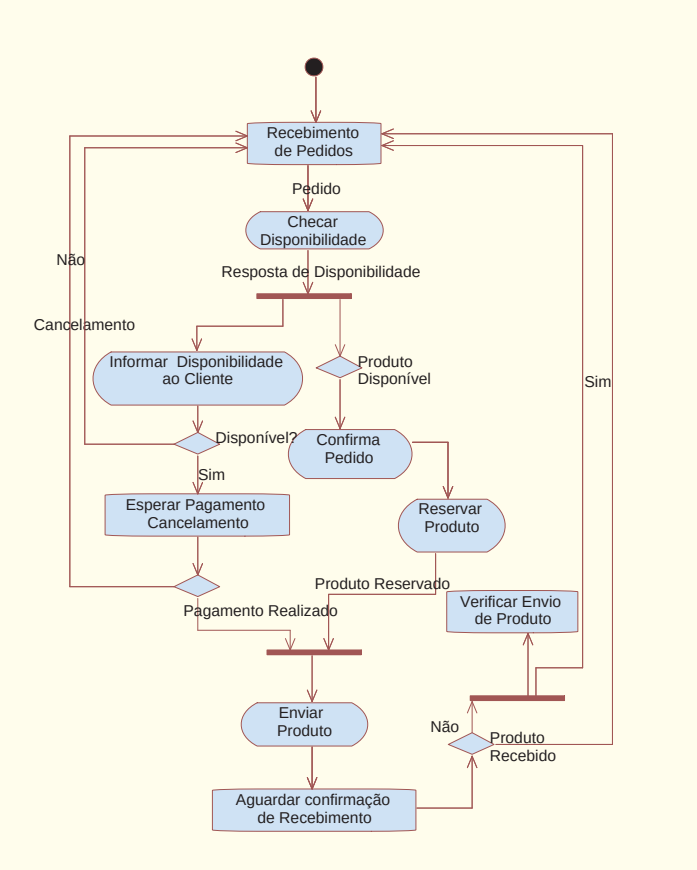


Figura 2.7: Processo Privado da Loja.

zações podem ser representadas por meio de uma organização hierárquica. Ou seja, transações formadas de sub-transações. Esse tipo de organização pode ser conveniente, por exemplo, para a substituição de uma sub-transação por outra no caso de falha. Embora a idéia de transações hierárquicas seja simples, o seu suporte é de grande complexidade.

Suporte a múltiplas transações com o mesmo objetivo. Um padrão comum em modelos de negócio é a inicialização de múltiplas transações com a mesma finalidade. Neste caso, o objetivo é escolher a transação que apresente o resultado mais satisfatório, segundo algum critério ou função objetivo. Por exemplo, na realização de uma compra, diversas transações de escolha da loja e do produto podem ser disparadas, sendo que apenas a que encontrar as melhores condições, segundo algum critério, é que será selecionada.

O controle de transações de negócio no contexto da ISI é descentralizado. O papel das aplicações que suportam essas transações deve ser dirigido pelas atribuições específicas de cada participante e pela percepção do contexto da transação com fins de determinar suas ações e tratar exceções como, por exemplo, a exclusão de um dado em virtude do abandono da transação por parte de um participante. Aplicações participam de transações de negócio como uma ferramenta por meio da qual os sistemas de informação participantes interagem realizando ações consistentes com o modelo da transação especificado. Essas aplicações devem perceber o contexto das transações de negócio e ser capazes de realizar tarefas que garantam a

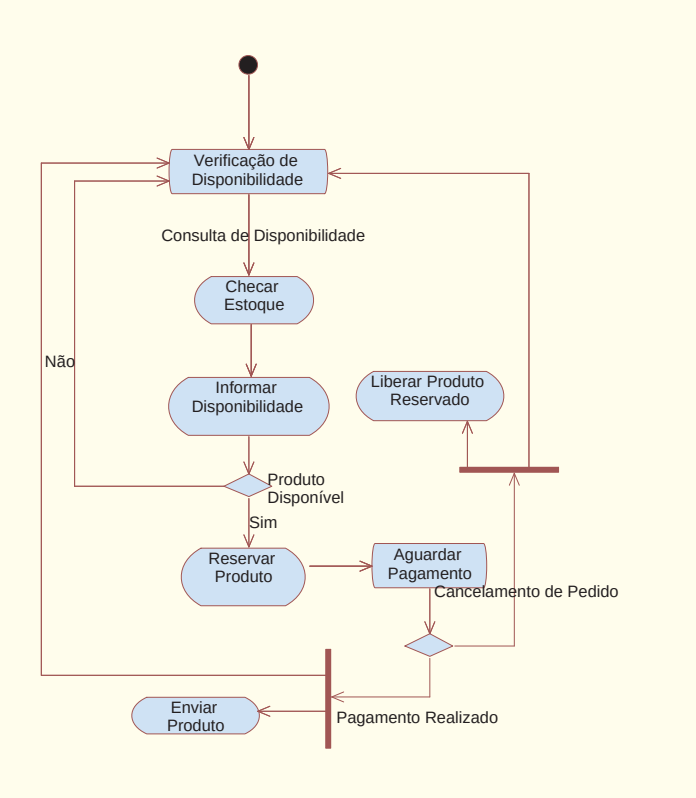


Figura 2.8: Processo Privado do Fornecedor.

mudança esperada no estado do negócio, mantendo a consistência entre o estado dos sistemas envolvidos. Exemplos dessas tarefas são o tratamento e escolha de múltiplos resultados de sucesso, o suporte a estados temporários e provisórios, suporte ao cumprimento completo ou cancelamento de transações (Dalal et al., 2003).

2.2.3 Mediação de Vocabulários

Em contextos heterogêneos, como contextos inter-organizacionais, podem ser identificados problemas de ambigüidade e incompatibilidades de vocabulários. Esses problemas ocorrem em diferentes níveis da representação da informação (Dey et al., 2002).

Os problemas relacionados a vocabulários na ISI são similares aos encontrados no inter-câmbio de documentos entre diferentes organizações. Guo (2006) enumera os seguintes casos de problemas entre múltiplas representações:

- Diferentes nomes podem ser usados para nomear dados ou operações idênticas.
- Dados e operações com o mesmo significado podem apresentar estruturas diferentes;
- Diferentes elementos de um dado ou operação podem ser utilizados para um único significado;
- Diferentes nomes podem ser utilizados para representar o mesmo valor;
- Diferentes atributos podem ser utilizados para um mesmo dado. Por exemplo, diferentes unidades de medida podem ser utilizadas para uma mesma grandeza.

Para realização da ISI, esses problemas podem ser solucionados em parte no nível da arquitetura de negócios. Iniciativas como consórcios entre organizações podem criar dicionários de termos e especificações de processo com sintaxe padronizada. Exemplos dessas iniciativas são o consórcio para o domínio de componentes eletrônicos e semicondutores *RosettaNet* (RosettaNet, 2008) e o consórcio para o domínio de aplicação de turismo *Open Travel Alliance* (Alliance, 2007). Entretanto, parte da solução para os problemas de mediação de vocabulários reside na solução de software da ISI. Portanto, devem ser previstos, para essas soluções, mecanismos que facilitem a mediação como, por exemplo mecanismos para resolução de conflitos de ambigüidade e identificação de sinônimos.

2.2.4 Semântica em Processos de Negócio

A descrição de processos públicos com a utilização de anotações semânticas tem sido uma importante inciativa para a integração de aplicações. A avaliação semântica de processos de negócio pode ser utilizada para a seleção de parceiros de negócio, por exemplo por meio da verificação de relações *todo-parte* ou relações *é um*. A análise desses aspectos torna-se mais importante à medida que anotações semânticas passam a incorporar a descrição de processos públicos, por exemplo, por meio de Serviços Web Semânticos (McIlraith et al., 2001; Sivashanmugam et al., 2003).

Anotações semânticas podem ser usadas também para definir o modelo de interação do processo público, estabelecendo as possíveis seqüências de passos dessa interação. Aplicações clientes de processos públicos que utilizam Serviços Web Semânticos devem ser capazes de determinar quais serviços invocar automaticamente e como invocar esses serviços com base nas anotações semânticas. Para reduzir a complexidade de desenvolvimento de soluções de software para ISI, ao lidar com essas anotações, idealmente, aplicações devem capturar a semântica representada e mapeá-la em formatos que facilite a utilização das informações por elementos tipicamente usados na lógica de negócios das aplicações.

2.2.5 Qualidade de Serviço

Aspectos da Qualidade de Serviço (QoS) envolvem as características qualitativas e quantitativas de soluções de software para ISI relacionadas aos requisitos especificados a partir da arquitetura de negócios da integração. Aspectos de QoS compreendem, por exemplo, disponibilidade, segurança, desempenho e eficácia, confiabilidade e acurácia (Kalepu et al., 2004). Os aspectos quantitativos são aqueles que podem ser medidos, como o tempo de execução e o custo de uma atividade. Aspectos qualitativos incluem restrições impostas à realização de interações de negócio como, padrões de confiabilidade ou índices de disponibilidade de um serviço (Cardoso et al., 2004).

Aspectos de QoS estão altamente relacionados com requisitos não funcionais da solução de software. Esses aspectos têm sido abordados na concepção de tecnologias de suporte à ISI. Um exemplo de iniciativa para suporte a aspectos de QoS na ISI é o arcabouço Web Services Policy (*WS-Policy*) (Box et al., 2003). WS-Policy compreende uma especificação que define a sintaxe para que provedores e consumidores de serviços possam estabelecer, descrever e aferir restrições e expectativas de QoS nos mais diversos domínios, por exemplo o domínio da segurança.

Segurança é um dos principais domínios de QoS para a ISI. Soluções de software para ISI devem garantir diversos requisitos de segurança tais como confiabilidade e autenticidade das comunicações. Muitas tecnologias de suporte à ISI possuem mecanismos pré-concebidos para atender a esses requisitos. Um exemplo desses mecanismos é a especificação Web Sevices Security (WSS) da OASIS (OASIS, 2006). Assim, para alguns aspectos da Segurança, é

suficiente que as soluções de software considerem o uso dessas tecnologias na especificação da arquitetura de tecnologias prevista para a ISI.

Entretanto, soluções de software devem considerar a realização da ISI entre domínios de segurança diferentes, autônomos e, muitas vezes, desconhecidos. Pode haver também a necessidade de se tomar decisões envolvendo aspectos de segurança durante a execução de processos de negócio. Uma especificação de mecanismo para essa situação é a especificação WS-Security Policy (Della-Libera et al., 2003). Com a implementação de mecanismos baseados em WS-Security soluções de software podem processar e especificar informações de segurança durante a execução dos processos de negócio.

2.3 Tecnologias de Informação e Comunicação para Integração de Processos

Soluções de software para ISI devem estabelecer as interdependências entre os sistemas de informação garantindo a privacidade entre os sistemas durante o processamento de qualquer operação necessária para a execução dos processos de negócio (Schulz e Orlowska, 2004). Esse princípio, bem como a necessidade freqüente da incorporação de sistemas legados e as diferenças de tecnologia entre as infra-estruturas tecnológicas das organizações têm levado ao desenvolvimento de soluções preferencialmente baseadas na *Integração de Aplicações* (Hohpe e Woolf, 2003; Medjahed et al., 2003).

A integração de aplicações difere de integração utilizando aplicações distribuídas, tais como aplicações cliente-servidor de múltiplas camadas (SUN, 2006). Na primeira, aplicações individuais devem permanecer autônomas e fracamente acopladas, mas podem realizar funções de forma colaborativa. Uma aplicação distribuída, por sua vez, caracteriza-se por ser fortemente acoplada, constituindo uma aplicação única, mesmo possuindo componentes fisicamente dispersos (Hohpe e Woolf, 2003).

Os seguintes modelos tecnológicos têm sido considerados na integração de aplicações (Hohpe e Woolf, 2003):

Transferência de Arquivos. Uma aplicação escreve em um arquivo que outra posteriormente lerá. Neste caso, aplicações devem estabelecer compromissos com relação ao formato, localização, tempo e controle dos arquivos.

Bancos de Dados Compartilhados. Um mesmo banco de dados é compartilhado por múltiplas aplicações. Neste caso, não há movimento de dados e o mecanismo de controle de acesso à informação é o definido no sistema de banco de dados utilizado, por exemplo transações ACID.

Chamada de Procedimento Remoto (RPC). Esse mecanismo permite implementar a integração Orientada a Métodos. Utilizando RPC, uma aplicação pode expor suas funcionalidades por meio de interfaces acessíveis a outras aplicações. As interfaces permitem que uma aplicação utilize as funcionalidades de outras aplicações executando essas funcionalidades remotamente. Um exemplo seria a Atividade Realizar Pedido na aplicação cliente (Figura 2.6), que pode ser realizada por meio da chamada de um procedimento remoto exposto pela aplicação da loja.

Mensagens. Permite estabelecer canais de comunicação assíncrona entre as aplicações. Nesses canais, aplicações postam e lêem mensagens. Para tanto, devem ser estabelecidos compromissos entre as aplicações com relação ao formato e conteúdo. Sistemas de mensagens permitem a especificação desses canais e dos diversos aspectos envolvidos nesta

especificação. Sistemas de mensagens têm sido suportados por quatro categorias de produto:

1. Sistemas operacionais. Por exemplo, a ferramenta *Microsoft Message Queuing* (MSMQ) no Windows XP;
2. Servidores de aplicação. Por exemplo, o *Java Messaging Service* (JMS) em servidores de aplicação baseados em J2EE;
3. Pacotes de desenvolvimento. Por exemplo, a ferramenta *Message Queuing* (MQ) no IBM WebSphere;
4. Pacotes de desenvolvimento que implementam especificações não proprietárias. Por exemplo, o Apache Axis que implementa as especificações de Serviços Web.

A adoção de múltiplos estilos de integração tem sido uma prática comum no desenvolvimento da solução de software para ISI. Desenvolvedores de tecnologias para suporte à integração de aplicações têm se guiado por diferentes modelos e conceitos, produzindo um cenário heterogêneo e dinâmico. Dessa forma, o desenvolvimento da solução para ISI precisa freqüentemente incorporar diferentes arquiteturas de produtos, subsistemas de transporte e protocolos de sistemas de mensagens (Vinoski, 2003).

Embora haja iniciativas no sentido de padronização dessas plataformas, a diversidade e dinamismo desse cenário impactam diretamente no desenvolvimento da solução de integração, criando novos requisitos não funcionais. Um exemplo desses requisitos é a garantia de interoperabilidade entre diferentes plataformas tecnológicas (Vinoski, 2003). Esse cenário heterogêneo e dinâmico das tecnologias também dificulta o atendimento a requisitos de desenvolvimento da solução, tais como garantia de evolução da solução com relação a mudanças tecnológicas (OMG, 2004; SEI, 2005).

As principais iniciativas para minimizar problemas de interoperabilidade têm se dado por meio da adoção de modelos e tecnologias padronizadas no contexto dessas plataformas. A adoção de tecnologias baseadas na Computação Orientada a Serviços, em particular, Serviços Web (W3C, 2002), é um dos principais exemplos dessas iniciativas. Tecnologias de Serviços Web incorporam aspectos tanto do modelo de Chamada de Procedimento Remoto quanto de Mensagens, têm se mostrado aderentes a diferentes plataformas de integração e focalizam elementos chave da integração, como localização, descrição e comunicação (Alonso et al., 2004).

A ampla aceitação de Serviços Web tem levado fornecedores de soluções de integração a incluí-los em seus produtos. Essa inclusão têm sido objetivado, principalmente, fornecer interfaces de programação de aplicações (API's) com suporte para o protocolo *Simple Object Application Protocol* (SOAP) (W3C, 2006). Não obstante, essas API's têm sido desenvolvidas sem a padronização necessária e não encapsulam completamente detalhes dos mecanismos de transporte subjacentes. Fornecedores de soluções de integração têm adaptado Serviços Web ao contexto de protocolos de transporte e arquiteturas de sistemas de mensagem proprietários (Vinoski, 2003). Ainda, por se tratar de uma tecnologia recente, mesmo entre versões de um mesmo produto existem grandes diferenças.

A Arquitetura Orientada a Serviços (W3C, 2002) e Serviços Web têm sido de fundamental importância na especificação de tecnologias para ISI. Na próxima seção discutimos esta Arquitetura e as tecnologias de Serviços Web mais profundamente.

2.3.1 Serviços Web

Um serviço Web é um componente de software acessível a outros componentes de software por meio de protocolos da Web. Serviços Web utilizam protocolos e linguagens de descrição de

interfaces baseadas em XML (Bray et al., 1998). O principal protocolo utilizado por serviços Web é o já mencionado *Simple Object Access Protocol* (SOAP). Esse protocolo serve para empacotar e transmitir as mensagens trocadas em estruturas chamadas envelope SOAP. Esses envelopes têm a função de descrever meta-dados das mensagens, por exemplo, endereço de entrega e padrões de codificação. Para transmitir os envelopes SOAP, as implementações de serviços Web utilizam protocolos de aplicação da Web, principalmente o HTTP. Esses protocolos funcionam como protocolos de transporte para serviços Web, estabelecendo conexões fim a fim entre serviços e clientes.

As mensagens que trafegam entre serviços Web e clientes são representadas utilizando as regras especificadas nas interfaces dos serviços. Geralmente essas mensagens são geradas pelas aplicações comunicantes. Em geral, elementos dessas linguagens, por exemplo, métodos, são serializados e desserializados. Isto é, para o envio de uma mensagem a um serviço Web, elementos no formato da aplicação local são transformados para o padrão XML descrito na interface do serviço. Similarmente objetos de retorno de operações do serviço chegam ao cliente no formato XML e são transformados para o formato da aplicação local.

Um dos principais objetivos da Computação Orientada a Serviços é o estabelecimento de uma ampla rede de serviços interconectados com seus potenciais clientes permitindo o estabelecimento dinâmico de colaborações *ad hoc* entre sistemas de software. A Arquitetura Orientada a Serviços (SOA) (Papazoglou e Yang, 2002; W3C, 2002) foi concebida visando a esse objetivo. A Figura 2.9 mostra os componentes dessa arquitetura e suas inter-relações.

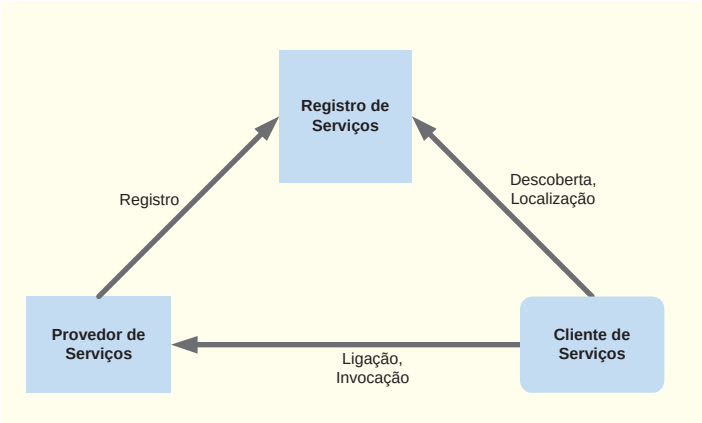


Figura 2.9: Arquitetura Orientada a Serviços (W3C, 2002).

Nesta arquitetura previu-se as seguintes atividades (W3C, 2002): *registro, descoberta, localização, ligação e invocação* de serviços. O registro de um serviço é a inserção de meta-dados desse serviço em um componente de software acessível e conhecido por potenciais clientes desse serviço. Esse componente é denominado Registro de Serviços e pode, ele mesmo, ser um serviço. A descoberta e localização se dá por meio de uma pesquisa na qual o cliente procura por serviços que atendam a algum critério. Ao descobrir o serviço desejado, o cliente deve localizá-lo. Essas operações são feitas sobre os meta-dados contidos em Registros de Serviços. Com a localização do serviço desejado, o cliente pode executar um preâmbulo da invocação do serviço denominado *ligação* (*bind*). A ligação visa estabelecer dinamicamente os elementos necessários para que o cliente possa invocar as operações disponibilizadas no serviço. Com o estabelecimento da ligação entre cliente e serviço, o serviço pode ser invocado pelo cliente, por meio de chamadas das operações disponibilizadas no serviço. Quando um cliente invoca uma operação em um serviço, essa operação é executada no contexto do provedor do serviço.

2.3.1.1 Exemplo de implementação de Integração de Processos utilizando Serviços Web

Continuaremos utilizando o exemplo do Pedido de Compra para ilustrarmos como serviços Web podem ser utilizados no desenvolvimento da solução de software para ISI. A Figura 2.10 apresenta uma visão geral da arquitetura de software, considerando o uso de Serviços Web, das aplicações envolvidas no Pedido de Compra.

As operações necessárias para a realização dos processos públicos são representadas interfaces. Essas interfaces são disponibilizadas fisicamente em servidores de aplicação com suporte para serviços Web, por exemplo, o Apache Tomcat junto com o Apache Axis (APACHE, 2006a).

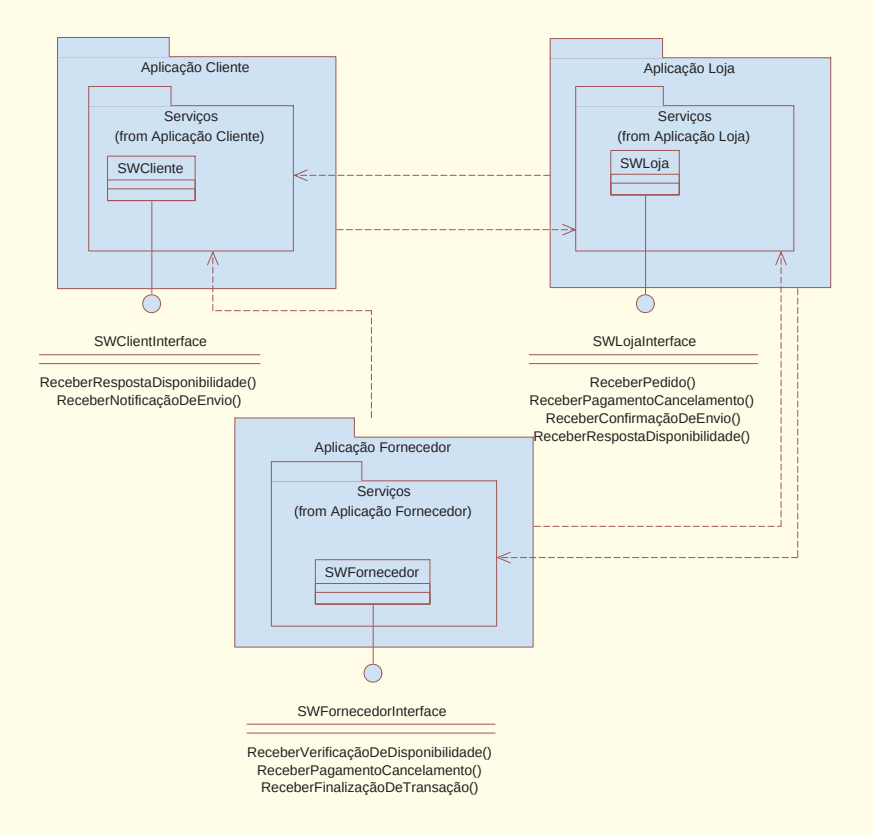


Figura 2.10: Visão Geral da Arquitetura de ISI para realização do Pedido de Compra utilizando Serviços Web.

As características de um serviço Web, tais como, suas operações, método de codificação e objetos complexos utilizados como parâmetros das operações, são descritos por meio de uma linguagem de descrição de interfaces chamada *Web Service Description Language* (WSDL) (W3C, 2001). WSDL baseia-se em XML. A Figura 2.11 apresenta fragmentos de código WSDL. Inicialmente são apresentados dados do serviço descrito. O segundo trecho mostra a definição do tipo complexo Order utilizado pelo serviço. O terceiro fragmento mostra a definição da operação purchaseOrder disponibilizada pelo serviço.

Arquivos WSDL não são para o consumo humano. Em geral, as ferramentas de suporte a Serviços Web disponibilizam mecanismos para geração automática de descrições WSDL a partir de códigos de programas descritos em linguagens de programação como C, Java e C++. O trecho de WSDL apresentado no exemplo foi gerado automaticamente no provedor

```
<wsdl:definitions
targetNamespace="http://localhost:8080/axis/services/Supplier1">
<!-- WSDL created by Apache Axis version: 1.4 Built on Apr 22,
2006 (06:55:48 PDT) --> <wsdl:types>

<schema targetNamespace="http://entity.supplier1">
<import
namespace="http://localhost:8080/axis/services/Supplier1"/>
<import namespace="urn:Supplier1"/> <import
namespace="http://schemas.xmlsoap.org/soap/encoding/"/>

 .. ..

<complexType name="Order">
    <sequence>
<element name="amount" nillable="true" type="xsd:double"/>
<element name="client" nillable="true" type="tns1:Client"/>
<element name="good" nillable="true" type="tns1:Good"/> <element
name="goodQuantity" nillable="true" type="xsd:int"/> <element
name="orderNumber" nillable="true" type="xsd:int"/> </sequence>
</complexType>

.. ..

<wsdl:operation name="purchaseOrder"> <wsdlsoap:operation
soapAction=""/>
    <wsdl:input name="purchaseOrderRequest">
<wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://services.supplier1" use="encoded"/>
</wsdl:input>
```

Figura 2.11: Excerto de um Arquivo WSDL.

do serviço pelo Apache Axis a partir de uma classe equivalente escrita na linguagem Java.

Clientes de serviços Web também não precisam lidar diretamente com WSDL. Ambientes e pacotes de desenvolvimento de Serviços Web disponibilizam ferramentas para a geração de *stubs*[1] a partir de WSDL. Um stub é uma representação local do serviço remoto por meio do qual a aplicação cliente consegue acesso indireto ao serviço. Geralmente espera-se que o *stub* seja gerado na linguagem de programação nativa do cliente. A Figura 2.12 mostra um fragmento de código de um *stub* em Java gerado a partir da descrição WSDL de um serviço.

A utilização dos *stubs* por uma aplicação cliente estabelece uma organização arquitetural para a mesma. A Figura 2.13 ilustra esta arquitetura por meio de um diagrama de componentes da UML. Nesta arquitetura, serviços Web dependem de componentes que implementam a lógica de negócio da aplicação. Por outro lado, os componentes que implementam a lógica de negócios da aplicação utilizam os *stubs* para acessarem os serviços Web dos parceiros de negócio e realizarem colaborações.

Diversas características de Serviços Web tornam essa tecnologia apropriada para o desenvolvimento de soluções de software para ISI. Curbera et al. (2001) enumeram as seguintes características fundamentais de serviços Web:

[1]Por falta de uma tradução para o Português amplamente aceita para o termo *stub*, manteremos o termo em inglês no decorrer do texto.

```
    public java.lang.Integer purchaseOrder(teste.integration.supplier1.Order order)
    throws java.rmi.RemoteException {
        if (super.cachedEndpoint == null) {
            throw new org.apache.axis.NoEndPointException();
        }
        org.apache.axis.client.Call _call = createCall();
        _call.setOperation(_operations[7]);
        _call.setUseSOAPAction(true);
        _call.setSOAPActionURI("");
        _call.setSOAPVersion(org.apache.axis.soap.SOAPConstants.SOAP11_CONSTANTS);
        _call.setOperationName(new javax.xml.namespace.QName("http://services.supplier1",
        "purchaseOrder"));

        setRequestHeaders(_call);
        setAttachments(_call);
 try {        java.lang.Object _resp = _call.invoke(new java.lang.Object[] {order});

        if (_resp instanceof java.rmi.RemoteException) {
            throw (java.rmi.RemoteException)_resp;
        }
        else {
            extractAttachments(_call);
            try {
                return (java.lang.Integer) _resp;
            } catch (java.lang.Exception _exception) {
                return (java.lang.Integer)
                org.apache.axis.utils.JavaUtils.convert(_resp, java.lang.Integer.class);
            }
        }
  } catch (org.apache.axis.AxisFault axisFaultException) {
  throw axisFaultException;
}
    }
```

Figura 2.12: Fragmento de um *stub* Gerado a partir de WSDL.

1. **Interoperabilidade no nível de troca de informação.** O nível mais básico de integração entre aplicações utilizando Serviços Web envolve a habilidade de trocar dados XML, por meio de um protocolo simples e padronizado, tal como o protocolo SOAP, que possa ser conduzido sobre os principais protocolos de comunicação da Web. Dessa forma, Serviços Web garantem um nível básico de interoperabilidade entre aplicações.

2. **Representação uniforme de sistemas heterogêneos.** As interfaces desses serviços são descritas utilizando WSDL. As funcionalidades das aplicações são descritas em termos de Mensagens XML que podem ser trocadas entre as aplicações. O uso de representações baseadas em XML provê independência de plataforma. Em adição, para o suporte a diferentes implementações com mesma funcionalidade, as interfaces de serviços Web estabelecem uma descrição funcional abstrata separada de possíveis implementações diferentes.

3. **Integração dinâmica de aplicações.** A especificação do conjunto de serviços com os quais um cliente de serviços irá interagir pode ser definida em tempo de execução. Dessa forma, é possível estabelecer a integração entre aplicações em tempo de execução

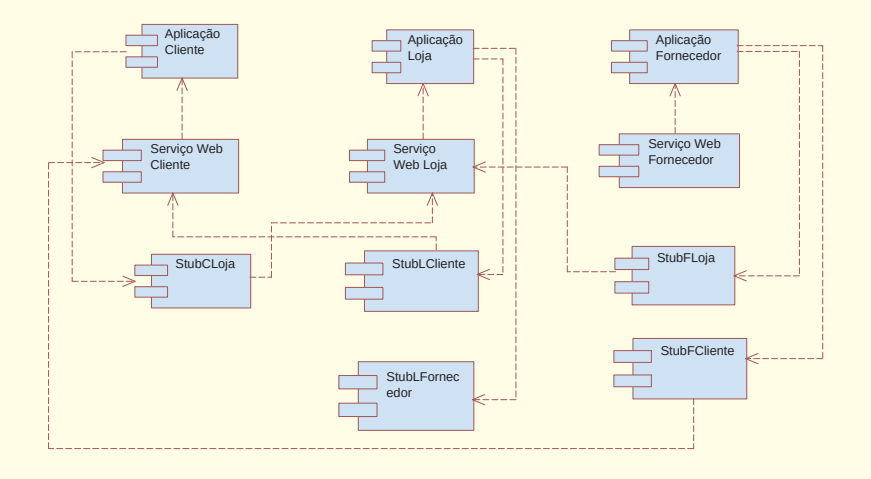


Figura 2.13: Arquitetura da ISI para Realização do Pedido de Compra com a Utilização de *stubs*.

e, como conseqüência, a integração entre aplicações pode se dar dinamicamente. A integração em tempo de execução requer, tipicamente, mecanismos de ligação tardia e a existência de registros de serviços por meio dos quais as aplicações possam selecionar esses serviços baseando-se tanto em aspectos funcionais e não funcionais da integração;

4. **Modelo de desenvolvimento da integração baseado em componentes.** O modelo de comunicação de Serviços Web baseia-se em mensagens. Entretanto, o uso de interfaces bem definidas e padronizadas para envio dessas mensagens (operações) permite que serviços Web sejam tratados como componentes de software e a adoção do modelo de desenvolvimento baseado em componentes (Szyperski, 2002). Neste modelo, serviços podem ser considerados como componentes cuja interface é composta de um conjunto de operações abstratas descritas em seus documentos WSDL.

2.3.1.2 Evolução de ferramentas de suporte a Serviços Web

Atualmente é possível identificar três gerações de ferramentas de auxílio ao desenvolvimento de Serviços Web. Apesar de terem sido concebidas em um espaço relativamente curto de tempo, existem grande diferenças entre essas gerações.

A primeira geração de ferramentas não tentava abranger a utilização comercial de Serviços Web. O motivo principal para a criação das ferramentas da primeira geração foi provar que o conceito de Serviços Web é válido. Sendo assim, tais ferramentas não abordaram vários aspectos importantes para a utilização de Serviços Web no mundo real, como desempenho e suporte a diferentes cenários de trocas de mensagens. Um representante da primeira geração é o Apache SOAP.

A segunda geração possibilitou a aplicação de Serviços Web em aplicações reais. Foi adicionado suporte a alguns cenários de troca de mensagens, como troca de mensagens síncronas e assíncronas com resposta rápida. Cenários envolvendo assincronismo com respostas demoradas ou troca de mensagens de grandes tamanhos continuaram sem suporte completo. Um representante da segunda geração é o Apache Axis até a Versão 1.4.

A terceira geração busca possibilitar a abrangência maior de domínio. As principais características dessa geração são o melhor desempenho, melhor suporte a operações assíncronas e facilidade para utilização de várias extensões da especificação de Serviços Web. As no-

vas extensões definem padrões para a implementação de Serviços Web que atendem a vários aspectos não suportados pela especificação de Serviços Web original. Um representante da terceira geração é o Apache Axis Versão 2.0.

Observamos algumas diferenças entre a segunda e terceira gerações. Destacamos essas diferenças considerando a arquitetura Axis 2 e Axis 1:

- O Axis 2 não suporta apenas interações do tipo pedido-resposta, como o Axis 1. A arquitetura na qual o Axis 2 é baseado permite que qualquer padrão de troca de mensagens seja modelado.
- No Axis 2 a troca de mensagens assíncronas é suportada tanto na API quanto na camada de transporte. O comportamento da camada de transporte pode ser configurado através da API para utilizar uma via (apenas ida) ou duas vias (ida e volta). Ao utilizar o transporte de duas vias é possível contornar limitações da camada de transporte, como por exemplo *timeout's* que ocorrem ao se aguardar por longos períodos de tempo pela resposta de um pedido.
- O processamento de mensagens SOAP é realizado através da API StAX (Axis1 utiliza a API SAX), utilizando menos memória e obtendo uma vantagem de desempenho sobre o Axis 1.
- A arquitetura do Axis 2 é altamente modularizada facilitando o suporte a extensões das especificações de Serviços Web. Para adicionar suporte a uma nova extensão basta instalar um módulo que implementa o suporte à nova extensão.
- O modelo de implantação de serviços é baseado em pacotes de arquivos. Para realizar a implantação de um serviço no Axis 2, basta copiar o pacote para a localização determinada na configuração que o serviço é automaticamente instalado e iniciado.
- O mapeamento de tipos de dados trocados foi remodelado. No modelo do Axis 2 cada mensagem ou objeto do documento SOAP é representado localmente por um objeto.

2.3.2 Sistemas de Gerenciamento de Processos de Negócio

Sistemas de Gerenciamento de Processos de Negócio (*Business Process Management Systems* - BPMS) compreendem uma importante alternativa tecnológica para o suporte a processos de negócio envolvidos na ISI. Considerando esta abordagem, as colaborações entre organizações são modeladas como fluxos de trabalho suportadas diretamente por um sistema de software. Nessa seção a discussão dos aspectos desses sistemas e definições empregadas são baseadas principalmente no trabalho de van der Aalst et al. (2003).

Um BPMS consiste de um sistema de software genérico cujas atividades são dirigidas explicitamente por modelos de processo e que permitem o gerenciamento de atividades que compreendem todo o ciclo de vida de processos de negócio operacionais. Esse ciclo de vida inclui: desenho, instanciação, gerenciamento e análise de processos de negócio.

Existe uma discussão no contexto da Gerência de Processos de Negócio quanto à organização física de BPMSs. Segundo van der Aalst et al. (2003), um BPMS pode ser tanto um sistema de software separado, semelhante a sistemas como SGBDs ou pode ser formado por um conjunto integrado de componentes de aplicações de domínio específico independentes. No domínio da Gerência de Processos de Negócio existem muitas expectativas, tanto de mercado quanto na comunidade acadêmica, pelo desenvolvimento de BPMSs como sistemas separados ou ainda, como módulos independentes em grandes pacotes corporativos como, por exemplo, nos pacotes SAP e Baan. Segundo van der Aalst et al. (2003), a idéia de definir um BPMS em um único sistema é consistente com três objetivos:

1. Evitar a codificação explícita de aspectos específicos de processos de negócio. Por meio desses sistemas espera-se reduzir a tarefa de codificação de software, transferindo-a para a montagem de sistemas *ad hoc* a partir de módulos de processamento reutilizáveis.

2. Suporte a desenho e redesenho orientado a processos. Esses sistemas permitiriam mudanças no comportamento de uma execução por meio da modificação de diagramas de fluxo de trabalho que representam os processos;

3. Aderência a modelos de computação para processos de negócio baseados em Serviços Web, por exemplo, o modelo adotado na linguagem *Business Process Execution Language for Web Services* (BPEL) (IBM, 2003). A linguagem BPEL permite a especificação de processos de negócio em termos das interfaces de serviço exibidas em WSDL pelos provedores. Uma máquina BPEL permite executar processos especificados nessa linguagem. Um BPMS, por sua vez, poderia ser construído no topo de uma máquina BPEL.

Iniciativas para o desenvolvimento de BPMS's têm ocorrido tanto na indústria relacionada à Tecnologia de Informação quanto junto a comunidade acadêmica. Não obstante, a complexidade do gerenciamento de processos de negócio, em parte discutida na Seção 2.2, tem tornado a concepção desses sistemas uma atividade de grande complexidade. Na comunidade acadêmica, essa complexidade tem levado a adoção de diferentes abordagens. Uma delas é o uso de modelos formais, como Redes de Petri (Reisig, 1985).

Sistemas de Gerenciamento de Processos de Negócio podem ser obtidos integrando-se aplicações de domínio específico e distribuindo-se, entre essas aplicações, as responsabilidades do gerenciamento dos processos de negócio. Essa alternativa corresponde ao modelo para ISI de Integração de Processos, discutido anteriormente. Na próxima seção focalizaremos nos aspectos de desenvolvimento de software voltado para essas aplicações e sua integração.

2.4 Aspectos de Desenvolvimento de Software para ISI

Nesta seção discutimos aspectos do desenvolvimento de soluções de software para ISI. Inicialmente, apresentamos uma revisão de conceitos de Desenvolvimento de Software relevantes para esta discussão. Posteriormente, focalizamos no desenvolvimento de Arcabouços de Software (*Software Frameworks*), como elemento de suporte ao desenvolvimento de soluções para ISI.

2.4.1 Revisão de Conceitos de Desenvolvimento de Software

2.4.1.1 Requisitos

Os requisitos de um sistema de software determinam o que esse sistema deve fazer para resolver um problema ou atingir um objetivo. As representações iniciais de requisitos descrevem o que o sistema deve fazer, mas não fornecem informações suficientes para o esboço de um sistema de software específico. Essas representações definem, não o comportamento de um sistema específico, mas o de uma classe de sistemas capaz de satisfazê-los.

Em abordagens de desenvolvimento de software típicas, representações iniciais de requisitos são posteriormente modeladas com vistas à obtenção de uma representação mais específica, denominada *especificação de requisitos*. A especificação de requisitos difere dos requisitos por estabelecer um comportamento (exterior) específico para o sistema de software idealizado (Davis, 1993). A especificação deve conter uma descrição completa do que o sistema de software deve fazer, ainda sem se ater a detalhes de como construir o sistema. Uma especificação de requisitos define os requisitos do software e dá condições para o início de sua construção.

Davis (1993) discute os conceitos acima e a delimitação da fase de requisitos no ciclo de vida de um software por meio do dilema "*o que* versus *como*". Segundo esse dilema, existe uma imprecisão para se determinar onde termina a atividade de determinação de *o que deve ser feito* e iniciam-se atividades para se determinar *como o sistema deve ser feito*. Esse dilema pode perdurar até especificações do sistema em níveis de abstração mais baixos, como a especificação modular do software, em um processo de refinamentos sucessivos. Entretanto, Davis considera que a fase de requisitos é delimitada pelo nível em que se constrói a especificação de requisitos.

Requisitos podem ser divididos em duas categorias: requisitos funcionais e requisitos não funcionais. Requisitos funcionais descrevem o comportamento do sistema em termos de suas funcionalidades. Um exemplo de requisito funcional no exemplo de Pedido de Compra é a funcionalidade Fazer Pedido. Requisitos não funcionais são restrições impostas ao sistema, que determinam em que condições o sistema deve funcionar (Leffingwell e Widrig, 2000; Mylopoulos et al., 1992). Exemplos de requisitos não funcionais são restrições de desempenho, robustez, confiabilidade e segurança.

Especificações de requisitos podem ser descritas de maneira informal, por meio de linguagem natural ou modelos, ou utilizando linguagens formais como, por exemplo, a linguagem *Requirements Modeling Language* (RML) (Greenspan et al., 1994). Independentemente da forma de representação de uma especificação de requisitos, é nela que se baseia a construção do sistema de software. Cabe a abordagem de desenvolvimento adotada dar condições para que um sistema de software possa ser desenvolvido a partir dessa especificação, seja ela informal ou formal (Fraser et al., 1991).

Uma técnica amplamente utilizada para a especificação de requisitos funcionais é a modelagem baseada em **Casos de Uso** (Jacobson et al., 1992). Um caso de uso é usado para representar e descrever uma funcionalidade completa de um sistema. Casos de uso são representados de forma gráfica e textual. A forma gráfica leva a construção de diagramas de caso de uso que fornecem uma representação conjunta das funcionalidades que o sistema deve possuir. Modelos para a construção de diagramas de caso de uso fazem parte da UML. Casos de uso são também utilizados na modelagem de negócios. Para estes casos, em processos de desenvolvimento é comum o uso da categorização *casos de uso de negócio*, indicando que o caso de uso em questão não representa uma funcionalidade de software.

A modelagem baseada em Casos de Uso é uma técnica efetivamente útil para a descrição de comportamento. Aspectos como, por exemplo, especificações de interfaces com o usuário podem sobrecarregar de detalhes os modelos de caso de uso. Essas e outras informações, por exemplo, regras de negócio e descrição de dados são mais bem descritas por meio de outros modelos, tais como, tabelas, formulações, máquinas de estado e diagramas de classes (Cockburn, 2001).

Requisitos não funcionais - RNF, desempenham papel importante na definição das características do sistema e tomada de decisões em seu desenvolvimento, servindo como critérios de seleção entre alternativas para esse desenvolvimento (Mylopoulos et al., 1992). Estes requisitos são fundamentais, particularmente, na escolha de tecnologias de desenvolvimento. Essas escolhas são difíceis de serem feitas, em parte, devido ao fato de que uma tecnologia que atenda um requisito possa conflitar com outros requisitos e, devido à conseqüente criação de interdependências entre as escolhas (Mylopoulos et al., 1992).

Requisitos também podem ser classificados em *Requisitos de Negócio, Requisitos de Produto* e *Requisitos de Desenvolvimento*. Requisitos de negócio descrevem em termos de conceitos de negócio, o que deve ser atingido para prover um benefício esperado. Requisitos de produto descrevem um sistema ou produto, o qual é uma das possíveis formas de se atender a requisitos de negócio. Requisitos de produto compreendem os requisitos funcionais e os requisitos não funcionais. Requisitos de desenvolvimento ou requisitos de processo descrevem aspectos do processo de desenvolvimento que uma organização deve seguir e as restrições que

se devem obedecer para garantir o benefício esperado (Davis, 1993).

O conjunto de técnicas utilizadas para levantar, detalhar, documentar e validar requisitos denomina-se Engenharia de Requisitos (van Lamsweerde, 2000). Processos de Engenharia de Requisitos envolvem atividades que se estendem da análise do domínio de aplicação à modelagem e validação de especificações de requisitos. Os aspectos de evolução e mudança de requisitos também são tratados pela Engenharia de Requisitos.

A compreensão correta de conceitos do domínio do problema que são relevantes para o desenvolvimento de uma aplicação de software é um dos objetivos da Engenharia de Requisitos. Requisitos de software são construídos com base nesses conceitos e nas suas relações. Para a construção de definições precisas desses conceitos e de suas relações pode-se, no contexto da Engenharia de Requisitos, utilizar a **Modelagem Conceitual** (Gulla, 1996).

Modelos conceituais permitem a descrição de requisitos de aplicação em termos dos conceitos do domínio do problema. Tais modelos podem oferecer diversas vantagens para o desenvolvimento, tais como, facilitar a comunicação entre partes interessadas e simplificar a descrição de elementos complexos do domínio (Lindland et al., 1994).

Modelos conceituais podem ser descritos utilizando elementos de modelo genéricos, como, por exemplo, classes, na modelagem orientada a objetos. Uma alternativa que pode complementar esses elementos de modelo em termos de expressividade é o uso de meta-conceitos. Meta-conceitos permitem categorizar conceitos do domínio do problema e pré-estabelecer regras e restrições para as associações entre conceitos. Um exemplo do uso de meta-conceitos é a categorização *entidade, controle e fronteira* apresentada por Jacobson et al. (1992) e amplamente utilizada na análise orientada a objetos. Um conjunto de meta-conceitos e suas associações constitue um meta-modelo (Guizzardi et al., 2004).

2.4.1.2 Processos de Software

Processos de software são construídos com base na noção de ciclo de vida de um software, organizando e disciplinando o uso dos métodos e ferramentas de desenvolvimento com o intuito de auxiliar na execução das fases desse ciclo (Fuggetta, 2000).

Há diferentes definições de processo de software (Fuggetta, 2000; Humphrey, 1995; Jacobson et al., 1998; Pressman, 2005). Porém, todas elas se baseiam na noção elementar de que um processo é uma abordagem sistemática para a criação de um produto ou para realização de uma tarefa (Osterweil, 1987). Humphrey (1995), por exemplo, estabelece que um processo de software é a seqüência de passos necessária para desenvolver ou manter sistemas de software e que a *definição do processo* é a descrição dessa seqüência e desses passos.

Feiler e Humphrey (1993) complementam a definição de processo com relação à sua estruturação. Segundo os autores, processos complexos de software podem ser vistos como um conjunto aninhado de abstrações. Cada processo é composto de sub-processos, os quais, por sua vez, podem ser subdivididos em elementos ainda menores. Não existe um limite técnico para delimitar o número de refinamentos que podem ser aplicados a um processo. Entretanto, devem ser considerados aspectos práticos como o tamanho específico do projeto e o grau de particionamento do trabalho dentro do projeto.

Uma definição mais detalhada de processo de software é apresentada por Fuggetta (2000):

> *Um processo de software é um conjunto coerente de políticas, estruturas organizacionais, tecnologias, procedimentos e artefatos que são necessários para conceber, desenvolver, implantar e manter um produto de software.*

Considerando essa definição, processos de software devem levar em conta as tecnologias, os métodos e técnicas de desenvolvimento e os aspectos organizacionais, de recursos humanos, mercadológicos e econômicos que afetem projetos de software (Fuggetta, 2000).

Processos de software são, portanto, entidades complexas que envolvem, inclusive, elementos fora do escopo da Computação.

2.4.1.3 Incorporação de Tecnologias em Processos de Software

Diferentes projetos de software possuem diferentes necessidades, mas também compartilham semelhanças. Para a definição de processos de software levando em conta semelhanças e diferenças entre projetos, uma solução é a criação de elementos de processo de software de propósito geral, reutilizáveis (Feiler e Humphrey, 1993). A partir desses elementos genéricos são então construídos elementos mais específicos. Essa abordagem leva à criação de níveis de abstração na definição de processos de software contendo processos mais ou menos abstratos e processos concretos.

Quando um processo é posto em prática em um projeto, dizemos que o mesmo é instanciado (*enacted*). Um processo instanciável é um processo que inclui todos os elementos requeridos para sua instanciação: a definição do processo, as entradas requeridas para o processo, seus agentes e recursos devidamente associados (Humphrey, 1995). Exemplos desses elementos são: modelos de ciclo de vida, fases e atividades desse ciclo, descrição de fluxos de atividades, modelos e tecnologias de desenvolvimento, pessoas e ferramentas.

Modelos de processo abstratos não podem ser instanciados diretamente. Entretanto, alguns elementos e decisões de propósito geral podem ser introduzidos nesses processos. Quando não se adapta totalmente às necessidades de um projeto, o processo necessita ser personalizado. A personalização (*Tailoring*) é o ato de adaptar o processo e suas definições para suportar a sua instanciação para um propósito particular (Humphrey, 1995). Esse processo pode ser feito através de três operações básicas: exclusão, inclusão e modificação de elementos do processo.

As tecnologias, como os demais elementos necessários no ciclo de vida de software, devem ser consideradas na definição e personalização de processos. Desde que o processo instanciado tenha todos os elementos necessários para a execução do projeto, não existem restrições sobre em que nível de abstração do processo as tecnologias devem ser consideradas.

Um processo pode ser definido com um mínimo de aspectos tecnológicos, deixando para a sua personalização a inclusão da maioria desses aspectos. Ou então, um processo pode definir todos os parâmetros tecnológicos do projeto, como a linguagem de programação, detalhes de codificação e arcabouços específicos que devem ser usados. Nesse caso, o processo de personalização pode ser mínimo para a classe de projetos a que o processo se destina. Na definição de processos deve-se considerar as relações de custo e benefício dessas decisões.

Os processos de software têm variado quanto às considerações a respeito da adoção de tecnologias. No *Rational Unified Process* - RUP, por exemplo, a decisão de usar componentes como elementos de modelo de implementação vêm de seu meta-processo (*Unified Process - UP*) (Jacobson et al., 1998). Um outro exemplo é a definição do uso de uma camada de persistência no processo Praxis (Paula-Filho, 2003). Neste caso, a acomodação de outros modelos de acesso a bancos de dados relacionais deve ser considerada na personalização do processo.

A definição de processos de software tem sido orientada por práticas e recomendações organizadas por meio de modelos e normas. Uma dessas normas é a **ISO 12207** (ISO/IEC, 2008). Essa norma estabelece uma estrutura comum para os processos de ciclo de vida de software, com terminologia bem definida. Essa estrutura contém processos, atividades e tarefas que servem para serem aplicadas durante a aquisição de um sistema que contém software, de um produto de software independente ou de um serviço de software, e durante o fornecimento, desenvolvimento, operação e manutenção de produtos de software. A ISO 12207 também provê um processo que pode ser utilizado para definir, controlar e melhorar os

processos.

Um dos processos organizacionais da ISO 12207 é o processo de Gerenciamento de Infra-Estrutura. Esse processo define práticas para o estabelecimento e manutenção da infra-estrutura necessária para qualquer outro processo. A infra-estrutura pode incluir hardware, software, ferramentas, técnicas, padrões e recursos para o desenvolvimento, operação ou manutenção. As práticas definidas nesse processo são organizadas em três atividades básicas: Implementação do Processo, Estabelecimento da Infra-Estrutura e Manutenção da Infra-Estrutura.

É por meio de atividades apresentadas em processos de Infra-Estrutura (ou equivalentes) que melhorias e inovações relacionadas a tecnologias de desenvolvimento são realizadas e incorporadas aos processos de desenvolvimento propriamente ditos.

Em nosso trabalho freqüentemente fizemos uso do Processo Unificado para apresentação de soluções e decisões tomadas no desenvolvimento. O Processo Unificado é um modelo de processo que não possui todos os elementos necessários para que possa ser instanciado. Não são especificados neste modelo, por exemplo, as linguagens de programação a serem adotadas e a arquitetura das aplicações a serem desenvolvidas. Entretanto, aspectos essenciais do processo definido já se apresentam especificados. O Processo Unificado prescreve, por exemplo, o uso de modelagem e desenho orientado a objetos, o uso de notação UML e o desenvolvimento baseado em casos de uso.

O Processo Unificado considera um modelo de desenvolvimento no qual são empregados três modelos principais: a) o Modelo de Análise, representado por diagramas que incluem elementos do domínio do problema; b) o Modelo de Desenho, com diagramas representando, em alto nível, os elemento do domínio da solução de acordo com as tecnologias definidas; e c) o Modelo de Implementação que detalha a solução descrita pelo modelo de desenho explicitando todos os aspectos relevantes do sistema físico que está sendo desenvolvido.

2.4.1.4 Arquiteturas de Software

O domínio de investigação da Arquitetura de Software compreende o estudo da estrutura geral dos sistemas de software, especialmente as relações entre seus componentes (Clements, 1996). Em processos de desenvolvimento de software, arquiteturas têm sido empregadas para a construção de modelos lógicos intermediários entre especificações de requisitos software e a solução para esses requisitos.

Define-se uma arquitetura de um sistema de software especificando-se o conjunto de componentes que irão compor o sistema , o comportamento desses componentes e os mecanismos de interação entre os mesmos (conectores). Os componentes representam elementos computacionais e de armazenamento de dados em um sistema. Exemplos de componentes são servidores, clientes, objetos e bancos de dados. Conectores representam a interação entre os componentes. Os conectores estabelecem relacionamentos entre componentes. Exemplos de elementos que podem ser modelados como conectores compreendem uma chamada de procedimento, um protocolo de comunicação e uma cláusula SQL ligando uma base de dados e uma aplicação.

Diversos processos de software, por exemplo, aqueles construídos a partir do Processo Unificado (Jacobson et al., 1998), se apóiam em arquiteturas de software como mecanismo para organização da solução de software. Um exemplo dessa adoção é a organização de aplicações em uma arquitetura em camadas. A arquitetura em camadas é um dos estilos arquiteturais discutidos por Shaw e Garlan (1996). Nesse estilo, os componentes de uma aplicação ou sistema são organizados em camadas hierárquicas. Componentes de uma camada só podem possuir acesso ou visibilidade sobre componentes localizados em camadas hierarquicamente inferiores.

Em processos baseados no Processo Unificado é comum a adoção de arquiteturas de ca-

madas baseadas na categorização *entidade, controle e fronteira* (veja Seção 2.4.1.1). A Figura 2.14 ilustra essa arquitetura por meio de um diagrama da UML. Cada pacote representa uma camada. A camada mais alta hierarquicamente é a camada de Fronteira, que contém componentes de interação do sistema, tais como interfaces com o usuário. A camada intermediária é a camada de Controle, que contém componentes que implementam a lógica de negócios da aplicação. A camada de Entidade fica na base da hierarquia e possui componentes relacionados a entidades do domínio, geralmente mapeadas em mecanismos de persistência de dados. Segundo a hierarquia dessa arquitetura, componentes da camada de fronteira visualizam componentes das camadas de Controle e Entidade. Componentes da camada de Controle só visualizam componentes da camada de Entidade e componentes da camada de Entidade não visualizam os componentes das outras camadas.

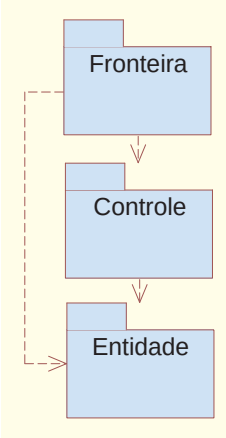


Figura 2.14: Exemplo de Arquitetura em Camadas em Processos de Software.

A **Arquitetura Dirigida por Modelos** (Model Driven Architecture - **MDA**) (OMG, 2004) é uma organização para o desenvolvimento de software no qual a especificação das funcionalidades do sistema é separada dos detalhes de sua implementação (Mashkoor, 2004). Ou seja, espera-se que a arquitetura de software da solução seja definida em função dos modelos que especificam o sistema a ser construído. MDA visa ao mesmo tempo o desenvolvimento de software em um alto nível de abstração e a construção de especificações de software flexíveis frente às constantes mudanças tecnológicas. Utilizando MDA, o desenvolvimento de um sistema deve ser organizado com base em três modelos: Modelo Independente de Computação (CIM), Modelo Independente de Plataforma (PIM) e Modelo Específico de Plataforma (PSM). MDA também prevê a construção de mecanismos de transformação entre PIMs e PSMs.

Um dos principais objetivos da MDA é que não seja necessária uma nova especificação de um sistema, para a adoção de uma nova tecnologia. A adoção de elementos de modelos semanticamente precisos e completos para a especificação da estrutura e do comportamento de sistemas de software, bem como a separação da lógica da aplicação das tecnologias subjacentes são elementos chave para essa arquitetura (Köhler et al., 2000).

Por proverem descrições abstratas dos sistemas, uma das principais atribuições de arquiteturas de software é servir como ponte entre requisitos e implementação (Garlan e Shaw, 1993). As tecnologias possuem modelos ou estilos arquiteturais como, por exemplo, modelos de comunicação ponto a ponto, modelo de objetos e modelos de arquiteturas em camadas. Quando uma tecnologia é adotada, esses modelos determinam parcial ou integralmente a organização do sistema a ser construído. Por exemplo, a utilização de uma linguagem orientada a objetos irá condicionar a organização dos programas fonte a um conjunto de classes

(Shaw e Garlan, 1996). Dessa forma, modelos arquiteturais desempenham papel fundamental na acomodação de tecnologias em processos de software. Dentre outras características, esses modelos permitem introduzir aspectos tecnológicos sem perda de generalidade para o processo.

2.5 Arcabouços de Software

Um arcabouço de software (arcabouço) é um elemento personalizável e extensível que pode ser utilizado para o desenvolvimento de uma aplicação ou de parte de uma aplicação. Sua concepção baseia-se em prover a solução para um conjunto de necessidades relevantes do domínio de aplicação. Arcabouços interferem diretamente na arquitetura, no modelo lógico e na implementação de aplicações (Johnson, 1997). Dessa forma, a influência de um arcabouço é fatorada entre os diferentes modelos e atividades do desenvolvimento de software.

O termo arcabouço (*software framework*) foi introduzido no contexto da programação orientada a objetos, sendo então definido como um conjunto de classes que cooperam entre si e que permitem o reúso de desenho para uma determinada classe de aplicações (Johnson e Foote, 1988; Deutsch, 1989; Johnson, 1997). A aplicação de um arcabouço pode variar quanto ao domínio do problema de diferentes maneiras. Por exemplo, um arcabouço pode auxiliar o desenvolvimento de editores gráficos de diferentes domínios, como edição de imagens, edição de músicas ou ferramentas CAD. Por outro lado arcabouços podem ser projetados visando a um domínio único, por exemplo, o suporte a sistemas financeiros.

Arcabouços definem a arquitetura de aplicações, ou parte da arquitetura, interferindo na definição da estrutura geral da aplicação, dos componentes necessários e na maneira como esses componentes interagem entre si. Enquanto o arcabouço define esses parâmetros de desenho, desenvolvedores da aplicação final podem se concentrar apenas em aspectos específicos da aplicação. Arcabouços enfatizam reúso de desenho ao invés de código, por meio de elementos como interfaces e classes abstratas. Entretanto é também comum a inclusão de subclasses concretas em arcabouços (Gamma et al., 1995).

O reúso de desenho leva ao princípio de inversão de controle entre aplicação e arcabouço. Normalmente, uma aplicação utiliza outro elemento de software realizando chamadas de operações definidas no elemento. Com arcabouços, é comum a necessidade de construção de classes concretas cujos métodos são utilizados pelo arcabouço. As funcionalidades de arcabouços personalizáveis por meio da construção de classes concretas específicas são chamadas do pontos de extensão (*hot-spots*). Pontos de extensão funcionam como mecanismos de personalização por meio dos quais aspectos específicos da aplicação são introduzidos no comportamento do arcabouço (Fayad et al., 1999).

O acoplamento fraco entre arcabouço e aplicação é um aspecto fundamental para garantir que arcabouços possam evoluir sem forçar constantes modificações das aplicações que o utilizam. Arcabouços de software permitem o desenvolvimento mais rápido de aplicações e fornecem estrutura similar a uma classe de aplicações, tornando-as mais fáceis de manter e mais consistentes para os usuários. O custo desses benefícios é a perda de flexibilidade do desenho, o qual em parte já é definido pelo arcabouço (Gamma et al., 1995).

Na Seção 2.2 discutimos aspectos do domínio da ISI. Esses aspectos são determinantes para a definição de requisitos de produto de soluções de software para ISI. Além dos requisitos de produto, o projeto dessas soluções deve também atender a requisitos de desenvolvimento. Esses requisitos interferem nas características desejáveis, pelas partes interessadas, relacionadas a prazo e custo do desenvolvimento e da manutenção de produtos de software (Agarwal e Rathod, 2005). Exemplos desses requisitos são produtividade no desenvolvimento, curva de aprendizado de tecnologias, testabilidade e manutenibilidade das soluções (SEI,

2006).

Na próxima seção discutimos aspectos de desenvolvimento que devem ser considerados no desenvolvimento de arcabouços de software para ISI.

2.5.1 Aspectos de Desenvolvimento de Arcabouços de Software

Para contribuir para o atendimento a requisitos de desenvolvimento de software, o projeto de arcabouços de software deve considerar importantes diretivas (Fayad et al., 1999). A seguir discutimos essas diretivas no contexto particular da Integração de Sistemas de Informação.

2.5.1.1 Integração entre Tecnologias

O desenvolvimento de software tem exigido cada vez mais a integração entre diversos elementos, tais como arcabouços, componentes, bibliotecas de interface gráfica e de comunicação e sistemas legados. Deve-se prever que um arcabouço será utilizado em conjunto com outros elementos. Sua arquitetura deve refletir esta necessidade considerando tanto aspectos fundamentais de modularização, por exemplo o acoplamento mínimo com os demais componentes da aplicação (Liskov, 1987), quanto o contexto de tecnologias do domínio de aplicação. No caso do domínio de ISI, o projeto de arcabouços deve levar em conta a necessidade de integração com elementos de software comuns ao desenvolvimento de sistemas de informação baseados na Web. Exemplos destes elementos são arcabouços para desenvolvimento de aplicações Web (APACHE, 2006b) e mecanismos de persistência (Hibernate, 2006).

A conformidade com padrões e especificações também é um aspecto fundamental para *integrabilidade*. No domínio da ISI, a integração com outros elementos de software padronizados pode ser simplificada pela utilização de interfaces e especificações de processos de negócio padronizadas, por exemplo, as especificações de Serviços Web, como WSDL e BPEL IBM (2003).

2.5.1.2 Facilidade de Desenvolvimento

O sucesso na utilização de arcabouços depende da capacitação dos desenvolvedores. Um importante critério para a adoção ou rejeição de um arcabouço é a relação entre o esforço necessário para sua adoção e os benefícios trazidos pelo seu uso. No desenvolvimento de arcabouços, a melhoria dessa relação implica em enfatizar a facilidade de uso como decisão de projeto (Fayad et al., 1999). No domínio da ISI, existe uma grande diversidade de tecnologias, produtos e alternativas de desenvolvimento disponíveis. Esse aspecto torna difícil a adoção de "mais uma tecnologia" para auxiliar o desenvolvimento. Dessa forma, para produzir arcabouços efetivos, deve-se dar garantias de que os custos de adoção serão pequenos.

Simplificar ou facilitar o desenvolvimento pode parecer um objetivo fundamental de qualquer ferramenta de desenvolvimento. Entretanto, o benefício de uma ferramenta pode estar em reduzir o custo de resolução de problemas de alta complexidade, e não necessariamente na facilidade de seu uso. Por exemplo, se um arcabouço atender expressivamente ao conjunto de requisitos de produto de soluções de software para a ISI, sua aplicação pode ser compensadora mesmo que o seu uso seja complexo. Entretanto, em um contexto de crescente diversidade de tecnologias, ferramentas difíceis de serem utilizadas tendem a ser descartadas.

2.5.1.3 Suporte a Mudanças

Sistemas de software podem sofrer modificações tanto durante o desenvolvimento, por exemplo, quando os requisitos mudam, quanto depois da entrega, por exemplo, para atendimento a novos requisitos ou correção de erros (von Knethen, 2002). No domínio da ISI, mudanças

que levam a novos requisitos têm se tornado tão freqüentes que a solução de software para ISI permanece sempre sob constante desenvolvimento (Hohpe e Woolf, 2003).

Modelos de desenvolvimento de software para ISI devem idealmente prover suporte a mudanças como uma funcionalidade básica e não mais encarar mudanças como atividades ou tarefas excepcionais. Esse suporte deve buscar estabelecer uma infra-estrutura relacionada tanto a mudanças relativas a aspectos tecnológicos quanto de requisitos de produto da solução de software.

2.5.1.4 Manutenção, Validação e Testes

A presença de componentes de software genéricos, a inversão de controle de execução e a perda do controle explícito do fluxo de execução provocadas pelo emprego de arcabouços tornam a validação e a remoção de defeitos de aplicações uma atividade complexa. Portanto, arcabouços devem incluir mecanismos que facilitem a localização de defeitos, tais como a produção de logs e suporte para a realização de testes. A verificação e validação da ISI compreendem também aspectos de verificação e validação dinâmica de sistemas distribuídos, tais como, a detecção de bloqueios perpétuos (*deadlocks*) (Foster et al., 2006).

Um exemplo de abordagem para a verificação de integração de aplicações utilizando Serviços Web é o uso de plataformas de ensaio *(testbeds)* (Pierce et al., 2002). Uma plataforma de ensaio é um conjunto de elementos computacionais definidos especificamente para realização de execuções controladas de um sistema (Ionescu et al., 2005). A utilização de plataformas de ensaio implica em configurar a aplicação especificamente para a realização de ensaios, considerando os elementos da plataforma. Essa configuração é complexa e pode envolver um alto custo. Arcabouços de software podem facilitar esta tarefa provendo mecanismos de reduzam o esforço dessa configuração.

* * * * *

Neste capítulo discutimos e apresentamos aspectos da Integração de Sistemas de Informação envolvendo a integração baseada em Processos de Negócio, as tecnologias de integração e aspectos de desenvolvimento de software. Com relação às tecnologias de integração focalizamos, particularmente, em Serviços Web. Com relação aos aspectos de desenvolvimento de software, focalizamos o desenvolvimento de Arcabouços de Software para ISI. Abordamos também aspectos como requisitos e processos de software que fazem parte, mesmo que indiretamente, da solução que apresentamos para o ISI.

No próximo capítulo, apresentamos nossas contribuições para o desenvolvimento da solução de software para ISI. Os diversos problemas e aspectos da ISI levaram a definição de um catálogo de requisitos para arcabouços de software para ISI. Com base nesses requisitos apresentamos o arcabouço BASS, destinado a auxiliar o desenvolvimento de aplicações clientes de Serviços Web. O método BASS, que considera o uso do arcabouço, provê uma solução de desenvolvimento para auxiliar no atendimento dos requisitos apresentados no catálogo.

Capítulo 3

Modelagem e Implementação da Integração de Sistemas de Informação

Neste capítulo apresentamos inicialmente um catálogo de requisitos para Arcabouços de Software para Integração de Sistemas de Informação (Costa et al., 2008b). Esses requisitos refletem os aspectos de ISI e de desenvolvimento de soluções para ISI discutidos no Capítulo 2. Posteriormente apresentamos os modelos, técnicas e mecanismos que, junto com o arcabouço BASS, fazem parte da solução que propusemos para auxiliar no desenvolvimento de soluções para ISI. Denominamos essa solução de *Método BASS*. Os elementos propostos foram desenvolvidos visando a sua adoção nas seguintes atividades do processo de desenvolvimento (de acordo com a norma ISO 12207): Análise de Requisitos de Software, Desenho Arquitetural de Software, Desenho Detalhado de Software, Codificação e Testes de Software.

O método BASS tem como objetivo o estabelecimento de um conjunto de técnicas e procedimentos para a utilização adequada do arcabouço BASS. Como o arcabouço BASS foi concebido com base nos aspectos de desenvolvimento organizados no método, entendemos que a exposição do método deve vir antes do arcabouço.

O restante desse capítulo está organizado da seguinte maneira: na Seção 3.1 apresentamos o catálogo de requisitos de Arcabouços de Software para Integração de Sistemas de Informação. Na Seção 3.2 apresentamos o método BASS. Na Seção 3.3 apresentamos o arcabouço BASS. Na Seção 3.4 apresentamos os experimentos que realizamos com o intuito de avaliar a efetividade do método BASS no desenvolvimento de soluções para ISI.

3.1 Requisitos de Arcabouços de Software para ISI

O uso de arcabouços de software pode contribuir tanto para o atendimento a requisitos de produto quanto no atendimento a requisitos de desenvolvimento. Nesta seção apresentamos um catálogo de requisitos para arcabouços de software cuja finalidade é suportar o desenvolvimento da solução de software para ISI. Agrupamos nesse catálogo os requisitos de produto e os requisitos de desenvolvimento.

Para apresentar o catálogo, utilizamos uma estrutura similar à utilizada por Hoffmann e outros (Hoffmann et al., 2004) na descrição de requisitos para ferramentas de gerenciamento de requisitos. Essa estrutura baseia-se na estrutura hierárquica utilizada no padrão ISO/IEC para avaliação de Produtos de Software (ISO/IEC, 1991). Os requisitos foram classificados em termos de sua prioridade como "alta", "média" e "baixa". Utilizamos os símbolos "+++", "++" e "+", respectivamente, para denotar essas prioridades. Estruturamos esses requisitos em duas seções: Requisitos de produto e requisitos de desenvolvimento. Os aspectos relacionados a requisitos de produto foram discutidos na Seção 2.2 e os aspectos relacionados a requisitos de

desenvolvimento foram discutidos na Seção 2.5.1. Na apresentação de cada requisito indicamos a subseção do tópico correspondente.

3.1.1 Requisitos de Produto

Requisito 1 (+++): Adaptação de Processos de Negócio (sec. 2.2.1) – Um arcabouço para ISI deve suportar a integração entre os processos privados e os processos públicos dos potenciais parceiros de negócio.

Requisito 2: Transações de Negócio (sec. 2.2.2) – (+++) Um arcabouço para ISI deve facilitar o suporte a modelos de Transações de Negócio.

Requisito 3 (+++): Mediação de Vocabulário (sec. 2.2.3) – Um arcabouço para ISI deve prover mediação de disparidades entre diferentes vocabulários utilizados pelos potenciais parceiros de negócio.

Requisito 4 (++): Anotações Semânticas (sec. 2.2.4) – Um arcabouço para ISI deve facilitar a utilização de anotações semânticas dos processos públicos dos potenciais parceiros de negócio.

Requisito 5 (+++): Qualidade de Serviços (sec. 2.2.5) – Um arcabouço para ISI deve prover suporte transparente a mecanismos de definição e avaliação de aspectos de Qualidade de Serviço.

3.1.2 Requisitos de Desenvolvimento

Requisito 6 (+++): Integração entre Tecnologias (sec. 2.5.1.1) – Um arcabouço para ISI deve ser de fácil integração com elementos de software comuns ao desenvolvimento de sistemas de informação baseados na Web.

Requisito 7 (+++): Simplificar o Desenvolvimento (sec. 2.5.1.2) – Um arcabouço para ISI deve ser de fácil utilização.

Requisito 8 (++): Suportar Mudanças (sec. 2.5.1.3) – Um arcabouço para ISI deve facilitar o mapeamento de mudanças nos requisitos de integração em modificações necessárias na solução de software.

Requisito 9 (++): Suporte à Manutenção, Validação e Testes (sec. 2.5.1.4) – Um arcabouço para ISI deve facilitar a realização de atividades e definição de mecanismos para suporte à manutenção, validação e testes da integração.

3.2 O Método BASS

O Objetivo do método BASS é auxiliar o desenvolvimento de soluções para ISI que utilizam tecnologias baseadas na Computação Orientada a Serviços. Nesse contexto, o método BASS é aplicável ao desenvolvimento de aplicações clientes de serviços. Organizamos o método BASS em função dos níveis de abstração de seus elementos. Foram considerados três níveis de abstração (Costa et al., 2004a):

1. Nível Conceitual - Neste nível são apresentados elementos que permitem a descrição da ISI em termos de elementos do domínio de aplicação;

2. Nível Lógico - Neste nível são apresentados uma arquitetura e outros mecanismos que permitem a descrição de soluções para ISI sem a especificação de detalhes das tecnologias envolvidas.

3. Nível Físico - Neste nível são apresentados os aspectos dos elementos físicos necessários para implementação concreta de soluções para ISI.

3.2.1 Nível Conceitual

Para permitir a descrição de requisitos da ISI no nível conceitual, desenvolvemos um meta-modelo com elementos de modelo aderentes ao domínio de Sistemas de Informação. O objetivo foi prover uma maneira simples para a representação dos requisitos da integração.

O meta-modelo proposto visa suportar a modelagem de requisitos da ISI considerando o ponto de vista de uma organização ou unidade de negócio específica. Ou seja, a solução de software em questão está sendo construída para uma unidade de negócio específica (unidade de negócio cliente) e visa integrar o sistema de informação dessa unidade com outros sistemas. Consideramos esses outros sistemas como dados. Ou seja, não são feitas suposições de que esses sistemas sofrerão modificações com a integração.

Para representar os conceitos do domínio da ISI, definimos os seguintes elementos de modelo:

- **Negócio**: Provê informação sobre unidades de negócio envolvidas na integração. Definimos uma Categoria de Negócio e uma Instância de Negócio da seguinte forma:

  1. uma categoria de Negócio é um tipo de unidade de negócio envolvido na integração. Exemplos de categorias de Negócio são Banco, Fornecedor, Loja, e Transportador.
  2. Uma instância de Negócio representa uma unidade de negócio específica. Cada instância de negócio deve ser de uma categoria representada no modelo. Exemplos de instâncias de Negócio são Banco X e Fornecedor Y.

- **Entidade de Negócio**: Provê informações sobre conceitos relevantes para a ISI utilizados pelas unidades de negócio. Definimos uma categoria de Entidade de Negócio e uma instância de Entidade de Negócio da seguinte forma:

  1. Uma categoria de Entidade de Negócio é um tipo de entidade de negócio. Exemplos de categoria de Entidade de Negócio são Pedido e Empréstimo.
  2. Uma instância de Entidade de Negócio representa uma Entidade de Negócio específica. Exemplos de instâncias de Entidade de Negócio são Pedido 38A e Empréstimo 8170.

O relacionamento entre conceitos derivados a partir dos meta-conceitos Negócio e Entidade de Negócio são governados pelas seguintes regras:

1. Uma categoria de Negócio pode ser associada com n categorias de Entidade de Negócio, $n \in N^*$;

2. Uma categoria de Entidade de Negócio pode ser associada com n categorias de Negócio, $n \in N^*$;

3. Uma instância de uma Entidade de Negócio deve ser associada a uma única instância de Negócio.

Diferentes instâncias de uma mesma Entidade de Negócio podem ser associadas a diferentes instâncias de diferentes categorias de Negócio. Por exemplo, uma instância de Pedido pode estar associada a uma instância de Transportador, enquanto outra instância de Pedido pode estar associada a uma instância de Fornecedor.

Modelos conceituais construídos utilizando os meta-conceitos Negócio e Entidade de Negócio abstraem diferenças de desenho e implementação dos parceiros de negócio envolvidos em uma colaboração. Por exemplo, uma categoria de Entidade de Negócio C define um único tipo de entidade para representar todas as possíveis especificações de C nas diferentes unidades de negócio envolvidas em uma colaboração. A aplicação dos meta-conceitos Negócio e Entidade de Negócio pode ocorrer no desenvolvimento de diagramas para a construção de uma visão conceitual dos requisitos, por exemplo no modelo de Análise do Processo Unificado.

Ainda não foi elaborado um *perfil da UML* (OMG, 2007) em que os conceitos envolvidos seriam Negócio, Entidade de Negócio, suas instâncias e os relacionamentos entre os mesmos. Conforme apresentamos na Seção 5.2 esse é um trabalho futuro que pretendemos realizar. Parte da definição de um perfil, envolve representar os meta-conceitos por meio de *estereótipos*. Em nossas atividades de desenvolvimento criamos, de maneira ainda informal, os estereótipos $\ll business\ unit \gg$ e $\ll entity\ proxy \gg$ para representar respectivamente os meta-conceitos Negócio e Entidade de Negócio. Os nomes dos estereótipos foram escolhidos a fim de evitar conflitos com outros nomes de estereótipos já existentes e foram mantidos em inglês para manter a uniformidade com os demais elementos da notação.

A Figura 3.1 apresenta um diagrama de classes para a modelagem conceitual da aplicação de suporte ao Cliente no exemplo do Pedido de Compra apresentado no Capítulo 2. As unidades de negócio envolvidas são a Loja e o Fornecedor, modelados como Negócio (estereotipados como $\ll business\ unit \gg$). Os conceitos envolvidos na integração foram modelados como Entidade de Negócio (estereotipados como $\ll entity\ proxy \gg$). Caso novas lojas e novos fornecedores fossem envolvidos na integração esse modelo permaneceria o mesmo.

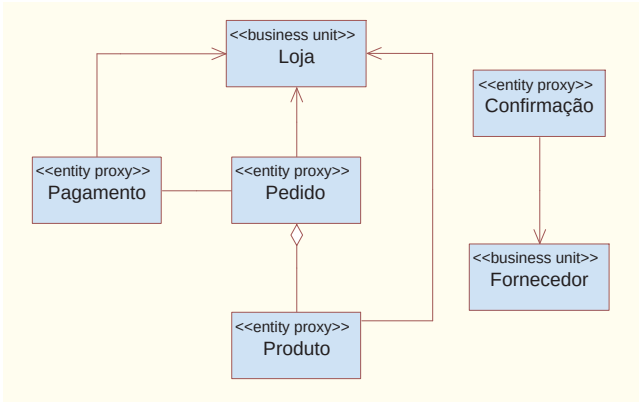


Figura 3.1: Excerto do Modelo Conceitual do Pedido de Compra.

3.2.2 Nível Lógico

Para suportar a descrição da solução de integração no nível lógico, pré-definimos mecanismos lógicos e uma arquitetura para a aplicação considerando o uso de tecnologias baseadas na Computação Orientada a Serviços na implementação física da integração. A idéia central dos elementos propostos foi facilitar o mapeamento entre modelos conceituais e as tecnologias de desenvolvimento.

3.2.2.1 Modelo Arquitetural

Derivamos o modelo arquitetural a partir do meta-modelo para análise orientada a objetos especificado no Processo Unificado. Esse meta-modelo especifica classes categorizadas em termos de três meta-conceitos: *Entidade*, *Controle* e *Fronteira*. Conforme discutido na seção 2.4.1.4, essa categorização pode subsidiar o estabelecimento de arquiteturas em camadas em processos de software derivados do Processo Unificado. Definimos a arquitetura para soluções de software para ISI considerando essa categorização e introduzindo elementos lógicos para o suporte ao meta-modelo conceitual de integração proposto na Seção 3.2.1.

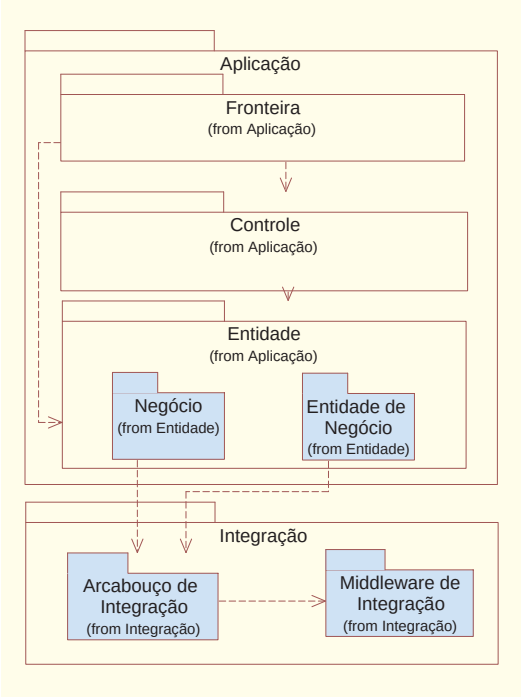


Figura 3.2: Arquitetura para Solução de Software para ISI.

A Figura 3.2 ilustra a arquitetura proposta. As camadas e subcamadas da arquitetura são representadas pelo elemento de modelo pacote da UML. A arquitetura está organizada em duas camadas principais: a camada de Aplicação e a camada de Integração. As relações de dependência entre as camadas denotam possíveis dependências entre os componentes das camadas e definem a hierarquia. Essa arquitetura possui os seguintes aspectos principais:

1. A camada de Aplicação engloba toda a lógica de negócio da aplicação, inclusive a lógica de negócios relacionada à integração. Essa camada também engloba componentes de interface com o usuário, entidades persistentes e entidades associadas à integração.

2. A camada de Integração provê os componentes de implementação que suportam os aspectos específicos de tecnologia da integração.

3. Os meta-conceitos Negócio e Entidade de Negócio são suportados por categorias de classes específicas no nível lógico. Classes baseadas nestas categorias são agrupadas nas sub camadas *Negócio e Entidade de Negócio*.

4. A lógica de negócio da integração, definida na camada de Aplicação, deve ser desenvolvida em termos de negócios e entidades de negócio. Ou seja, qualquer referência

a elementos externos à aplicação, na definição da lógica de negócio, deve ser feita em termos de conceitos derivados dos meta-conceitos Negócio e Entidade de Negócio.

5. Os aspectos físicos da integração necessários para suportar conceitos derivados dos meta-conceitos Negócio e Entidade de Negócio são especificados na camada de Integração e são transparentes para os desenvolvedores da camada de Aplicação.

3.2.2.2 Modelo de Abstração de Operações de Serviços

Uma das principais propriedades da arquitetura proposta é a separação da lógica de negócio dos aspectos de implementação da integração. Para satisfação dessa propriedade, negócios e entidades de negócio não podem considerar aspectos de implementação das aplicações e serviços dos parceiros de negócio envolvidos na Integração. Portanto, esses aspectos devem ser suportados de forma transparente, do ponto de vista de desenvolvedores da camada de Aplicação.

Uma instância de uma entidade de negócio é, na verdade, uma representação local de uma entidade externa mantida pela sua unidade de negócio associada. O acesso concreto a essa entidade se dá por meio do mecanismo de acesso disponibilizado pela unidade de negócio associada, como, por exemplo, um serviço Web. Considerando o uso de Serviços Web, o estado das entidades de negócio pode ser modificado simplesmente por meio da invocação das operações disponíveis no serviço disponibilizado pela unidade de negócio à qual pertence a entidade de negócio. Para evitar esse acesso diretamente da camada de Aplicação, preservando a separação dessa dos mecanismos de implementação, definimos um conjunto abstrato de operações para a manipulação de entidades de negócio denominado **operações RLSDU**. RLSDU é uma sigla formada pelos termos em inglês ***R**etrieve, **L**ist, **S**ave, **D**elete, e **U**pdate*.

Operações RLSDU são um conjunto de operações aderentes ao domínio de Sistemas de Informação. Esse conjunto é similar ao conjunto conhecido como CRUD (*Create, Retrieve, Update, e Delete*) (Kilov, 1990). Operações RLSDU agem sobre entidades de negócio considerando o estado local e externo dessas entidades. O estado externo corresponde à representação de uma entidade de negócio junto à unidade negócio associada a mesma. Essas operações possuem a seguinte semântica:

- **Retrieve** – Recupera o estado externo de uma instância de entidade de negócio;
- **List** – Recupera o estado externo de um conjunto de instâncias de entidade de negócio;
- **Save** – Salva o estado de uma instância de entidade de negócio, transferindo seu estado interno para o seu estado externo de acordo com a representação externa provida pela unidade de negócio a que pertence;
- **Delete** – Remove a representação local e externa de uma instância de entidade de negócio;
- **Update** – Atualiza o estado de uma instância de entidade de negócio transferindo seu estado interno para o seu estado externo de acordo com a representação externa provida pela unidade de negócio a que pertence.

3.2.2.3 Camada de Integração

O objetivo principal da camada de Integração é suportar a implementação da lógica de negócios da integração, na camada de Aplicação, em termos de negócios e entidades de negócio. Consideramos o uso de Serviços Web como o principal mecanismo para o desenho e implementação da camada de Integração. Em adição, estendemos a infra-estrutura provida por serviços Web por meio de um arcabouço de software para ISI (arcabouço de integração).

A Figura 3.2 apresenta o arcabouço de integração como uma sub-camada da camada de Integração. Esse arcabouço deve prover soluções personalizáveis para os requisitos de integração, considerando o modelo lógico da camada de Aplicação e estendendo as funcionalidades providas pelo *middleware* de suporte a serviços e por outras tecnologias que venham a ser incorporadas no desenvolvimento. O arcabouço de integração provê isolamento entre a camada de Aplicação e os componentes de *middleware*. Com essa concepção da camada de Integração, mecanismos e operações associados a serviços, por exemplo, ligação, execução e composição de serviços, não devem afetar o desenvolvimento da lógica de negócios da integração especificada na camada de Aplicação. Para satisfazer esses aspectos, o arcabouço de integração deve prover os seguintes mecanismos lógicos (Costa et al., 2008b):

- Suporte a operações RLSDU: O suporte a RLSDU consiste em prover mapeamento dinâmico de cada par (entidade de negócio, operação RLSDU) em operações de serviço, em função da lógica de negócios executada a partir de elementos da camada de Aplicação. Com o suporte às operações RLSDU, estabelecemos um mecanismo implícito de adaptação entre os processos de negócio privados e públicos;

- Suporte à integração com unidades de negócio: O arcabouço deve encapsular mecanismos de ligação com serviços disponibilizados pelas unidades de negócio para fornecer ligações transparentes entre instâncias de Negócio e as respectivas unidades de negócio;

- Suporte a Transações de Negócio: O arcabouço de integração deve suportar os diversos aspectos de transações de negócio e a implementação dessas transações em termos de negócios, entidades de negócio e operações RLSDU na camada de Aplicação;

- Suporte à Mediação de Vocabulários: Mediação de vocabulários pode ser suportada por diferentes tipos de mecanismos com diferentes graus de complexidade. Idealmente, o arcabouço de integração deve prover um mecanismo de mediação personalizável.

- Qualidade de Serviço: O Arcabouço de integração deve encapsular mecanismos de suporte a QoS, tornando aspectos de QoS gerenciáveis e acessíveis por meio de mecanismos abstratos que possam ser utilizados no desenvolvimento da camada de Aplicação.

3.2.3 Nível Físico

A representação da solução de software para ISI no nível físico consiste no produto ou parte do produto de software destinado à integração. Os elementos necessários para o seu desenvolvimento compreendem elementos de software concretos, tais como componentes, bibliotecas e arcabouços. Apresentamos nesta seção os elementos que auxiliam a implementação desse produto de acordo com a arquitetura proposta no modelo lógico.

Os elementos de modelagem física que propusemos são disponibilizados por meio de uma interface de programação de aplicações (API). Essa API é suportada pelo arcabouço BASS. Para o desenvolvimento da camada de Aplicação, essa API oferece os seguintes elementos fundamentais:

Session – Session é uma classe concreta. O objetivo principal dessa classe é prover as operações RLSDU.

Business – Business é uma classe abstrata. Por meio de especializações dessa classe são definidas categorias de Negócio;

EntityProxy – EntityProxy é uma classe abstrata. Por meio de especializações dessa classe são definidas categorias de Entidade de Negócio;

OclExpression – OCLExpression é uma classe concreta. Por meio dessa classe são descritas regras de negócio da integração. Essas regras de negócio são traduzidas pelo arcabouço em mapeamentos e execuções de operações definidas pelos processos públicos dos negócios envolvidos na integração;

ResponseHandler – ResponseHandler é uma classe abstrata. Por meio de especializações dessa classe são definidos métodos específicos de tratamento de respostas em interações assíncronas.

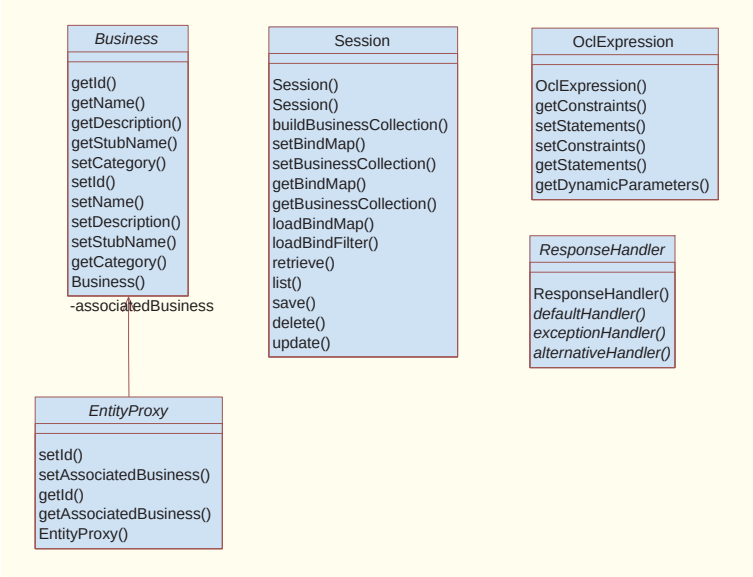


Figura 3.3: Interface de Programação de Aplicações do BASS.

A Figura 3.3 apresenta essas classes e suas operações públicas. Na próxima seção apresentamos o arcabouço BASS e como se dá o suporte a essa API. Antes, contudo, aproveitamos para ilustrar o método BASS com um exemplo. O exemplo usado envolve a implementação da funcionalidade Gerenciamento de Pedido de Compra, similar ao exemplo utilizado no Capítulo 2. Essa funcionalidade possui a seguinte descrição: *O gerenciamento de Pedido de Compra deve prover a funcionalidade que permita uma unidade de negócio cliente realizar pedidos de produtos em várias unidades de negócio fornecedoras.* Essa funcionalidade deve incluir as seguintes operações:

- **Fazer Pedido.** O cliente faz um pedido seguindo o seguinte fluxo:
  1. Selecionar o fornecedor;
  2. Especificar os itens do pedido, escolhendo os produtos a partir da lista de produtos do fornecedor escolhido;
  3. Criar um pedido com o itens definidos;
  4. Submeter o pedido ao Sistema do Fornecedor.

- **Verificar Pedido.** O cliente verifica um pedido seguindo o seguinte fluxo:
  1. Selecionar o Fornecedor;
  2. Recuperar o Pedido no sistema do fornecedor, dado um identificador do pedido;
  3. Apresentar o pedido.
- **Cancelar o Pedido.** O cliente cancela um pedido seguindo o seguinte fluxo:
  1. Selecionar o Fornecedor;
  2. Excluir o pedido no sistema do fornecedor dado um identificador do pedido;
- **Atualizar um Pedido.** O Cliente atualiza um pedido seguindo o seguinte fluxo:
  1. Selecionar o fornecedor;
  2. Recuperar o pedido a partir do sistema do Fornecedor, dado um identificador do Pedido;
  3. Atualizar os dados do pedido recuperado no sistema do fornecedor.
- **Listar um conjunto de Pedidos.** O cliente lista seus pedidos seguindo o seguinte fluxo:
  1. Selecionar o fornecedor;
  2. Recuperar uma lista de pedido do sistema do fornecedor de acordo com um critério de seleção;
  3. Apresentar a lista de pedidos;

Para discutirmos o desenvolvimento do Pedido de Compra, consideramos o uso do Processo Unificado. Modelamos o Gerenciamento de Pedido como um caso de uso. O meta-modelo conceitual proposto é empregado no desenvolvimento do modelo de análise. A arquitetura de integração e os mecanismos de desenho adotados são parte do modelo de desenho. A API do BASS fornece os elementos para a implementação dos aspectos de integração na camada de Aplicação e o BASS, junto com o *middleware* de serviços, suporta a implementação da camada de Integração.

A Figura 3.4 apresenta um excerto de um possível modelo de análise para o caso de uso Gerenciamento de Pedido. Os elementos de modelo Negócio e Entidade de Negócio foram modelados como os estereótipos $\ll business\ unit \gg$ e $\ll entity\ proxy \gg$, respectivamente. A classe Fornecedor é, então, estereotipada como um $\ll business\ unit \gg$ e representa todos os potenciais fornecedores do cliente. As demais classes do modelo são todas estereotipadas como $\ll entity\ proxy \gg$. Neste nível de representação classes estereotipadas como $\ll entity\ proxy \gg$ são representações genéricas de uma mesma categoria de entidade. A definição de atributos e operações dessas classes deve se basear na modelagem de negócios da integração.

A Figura 3.5.a apresenta a arquitetura de integração entre um cliente e três fornecedores. Três elementos compõem a camada de Integração: o arcabouço BASS, os pacotes de código fonte local (*stubs*) dos serviços utilizados e as interfaces de programação de aplicações (API's) correspondentes às diferentes opções de *middleware* necessários para a integração com cada fornecedor distinto. Conforme discutido na Seção 2.3.1 do Capítulo 2, *stubs* são representações locais dos serviços que descrevem suas operações e os tipos de dados utilizados pelos serviços. Embora *stubs* forneçam representações mais abstratas do que outros mecanismos, por exemplo, chamada de procedimento remoto, o desenho desses elementos pode diferir dependendo da API de serviços utilizada.

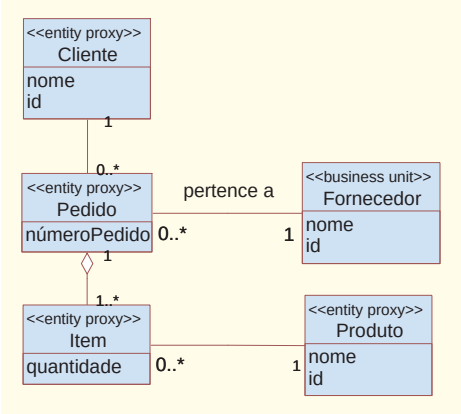


Figura 3.4: Excerto do Modelo de Análise do Gerenciamento de Pedidos.

No cenário apresentado na Figura 3.5.a, cada fornecedor é acessado por meio de seus *stubs* particulares com o suporte de uma API específica. As API's de serviços são os elementos de software que implementam o suporte à comunicação com os serviços, por exemplo, o Apache AXIS (APACHE, 2006a). As API's podem ser as mesmas para diferentes serviços, podem ser a mesma mas com diferenças de versão ou podem ser distintas. Para a utilização do BASS, a única restrição com relação a essas API's é a adoção do mecanismo de *stubs*. A versão corrente do BASS suporta as versões 1.4 do Apache Axis e a versão 2.0, na qual o mecanismo de *stubs* foi completamente remodelado.

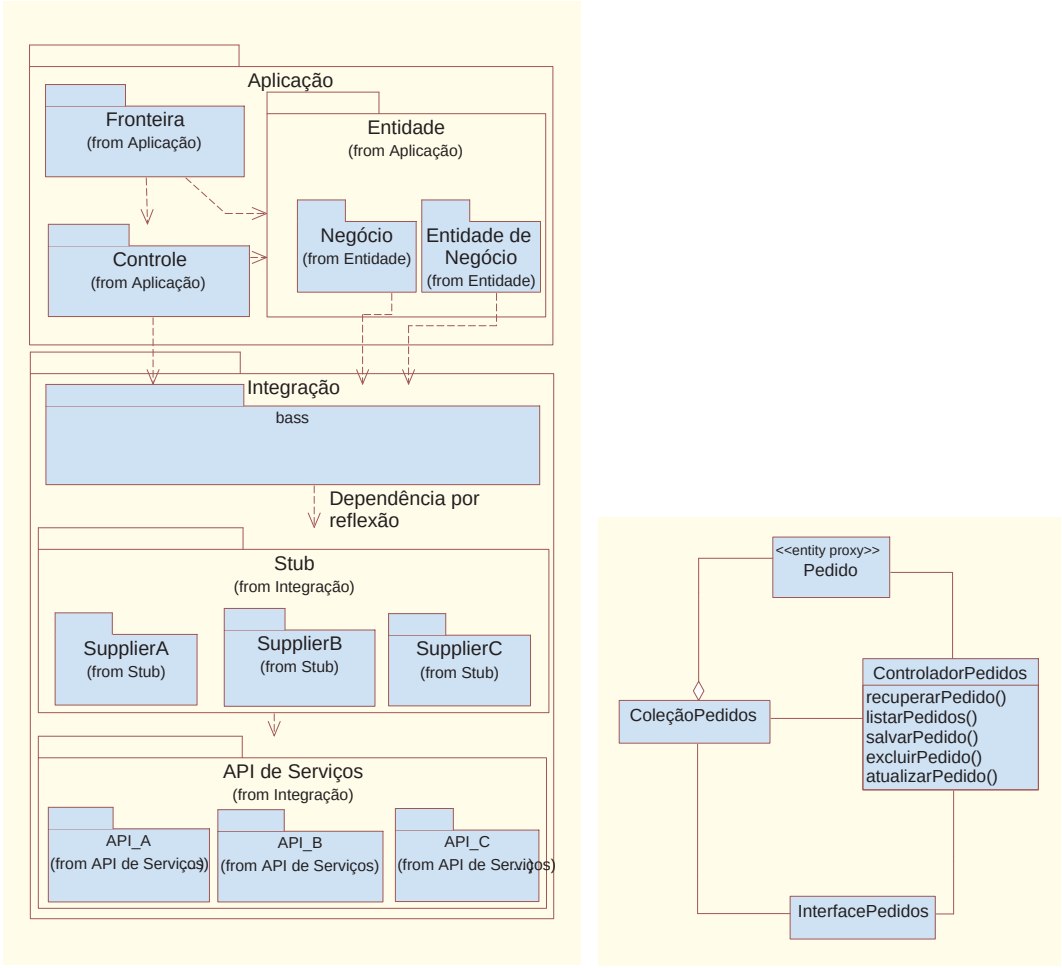
A Figura 3.5.b apresenta um excerto de um diagrama de classes do modelo de desenho do Gerenciamento de Pedidos. Para implementar as operações necessárias ao Gerenciamento de Pedidos, a classe ControladorPedidos necessita implementar as operações recuperarPedido(IdPedido id), listarPedidos(Criterio criterio), salvarPedido(Pedido pedido), excluirPedido(IdPedido id), e atualizarPedido(Pedido pedido). Esse conjunto de operações pode ser implementado em termos de operações RLSDU, as quais são suportadas pelo arcabouço BASS.

3.3 O Arcabouço BASS

A Figura 3.6 apresenta os principais relacionamentos entre a camada de aplicação e o arcabouço BASS. A classe Pedido e todas as outras classes estereotipadas como Entity Proxy são especializações da classe EntityProxy do BASS. Classes estereotipadas como Business Unit são especializações da classe Business. As operações RLSDU podem ser executadas sobre qualquer classe que herda da classe EntityProxy. O mapeamento das operações RLSDU em operações de serviços é determinado em função da lógica de negócio expressa utilizando a linguagem *Object Constraint Language* (OCL) (OMG, 2003). A classe OCLExpression provê um mecanismo para a definição de expressões OCL relacionadas com a integração.

Toda operação RLSDU é não bloqueante. Esse aspecto provê um mecanismo inerente ao BASS para suporte a transações assíncronas e de longa duração. Entretanto, o suporte a operações assíncronas depende do *middleware* subjacente. Ou seja, o *middleware* de serviços deve prover algum mecanismo para suportar a execução de operações assíncronas. A versão atual do BASS suporta operações assíncronas quando o *middleware* utilizado provê o mecanismo de *call back*, tal como o AXIS 2.0.

Para o desenvolvimento da camada de Aplicação, aspectos de implementação, como o estilo do mecanismo de operações assíncronas, é transparente. Por exemplo, quando uma operação Retrieve é executada sobre um Pedido, a resposta esperada é uma instância de Pedido. Se o



a)Arquitetura de Integração para o Gerenciamento de Pedidos. b) Elementos do modelo lógico.

Figura 3.5: Modelo Lógico da Integração.

mapeamento definido para essa operação especificar uma operação assíncrona, a instância de Pedido não é imediatamente retornada. Entretanto, quando a resposta chega, o BASS trata automaticamente a instância de Pedido retornada.

A personalização do BASS envolve duas tarefas principais:

Especificação dos mapeamentos de cada classe EntityProxy: Os mapeamentos de classes EntityProxy são feitos por meio de ligações. Uma ligação é a implementação de um tipo abstrato de dados que encapsula uma associação entre uma *EntitiyProxy* e um serviço de software e os detalhes dessa associação. Esses detalhes compreendem a sintaxe das operações dos serviços, seus parâmetros e valores de retorno.

Especificação do vocabulário da Aplicação: O vocabulário da aplicação inclui a definição de um conjunto de termos e relações morfológicas, por exemplo, sinonímia, entre esses termos.

Cada par *{EntityProxy, operação RLSDU}* possui um conjunto de ligações. Por exemplo, a operação *Save* para a **EntityProxy** Pedido possuirá uma ligação com cada um dos três fornecedores. No BASS, cada conjunto de ligações (*bindings*) de um *EntityProxy* é representada por um arquivo XML que denominamos arquivo de ligação. Cada arquivo de ligação possui uma entrada para cada operação RLSDU. O quadro 3.7 apresenta o conjunto de ligações correspondentes ao par *{Pedido, save}*.

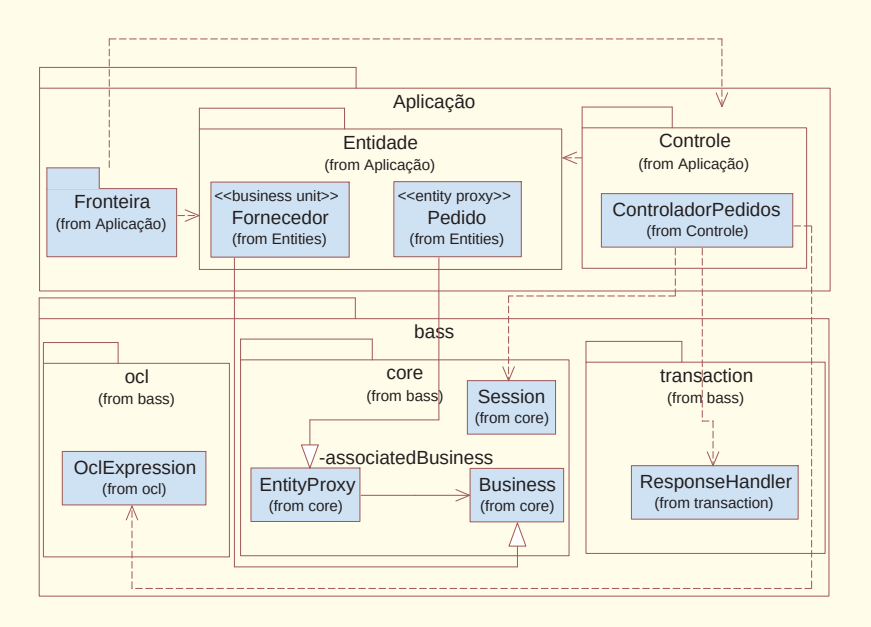


Figura 3.6: Dependências entre a Camada de Aplicação e o Arcabouço BASS.

O mecanismo de especificação de um vocabulário usado pela aplicação é um ponto de extensão do arcabouço BASS. Esse ponto de extensão permite a incorporação de diferentes mecanismos para análise e mediação de vocabulários considerados na integração. Como mecanismo de mediação padrão o BASS baseia-se em listas compostas por: a) um termo local, usado na modelagem de conceitos da aplicação local e b) um conjunto de sinônimos obtidos a partir de conceitos utilizados na definição de serviços. A Figura 3.8 ilustra os termos usados no exemplo do Gerenciamento de Pedidos.

3.3.1 Relacionamento do Método BASS com outras Abordagens para ISI

No contexto da Computação Orientada a Serviços, a investigação acadêmica tem se concentrado principalmente em aspectos envolvendo composição, descoberta e localização dinâmica e automática de serviços, por meio de mecanismos para representação de semântica ou modelos propostos pela Gerência de Processos de Negócio (van der Aalst et al., 2003). Por outro lado, os aspectos de desenvolvimento de software têm sido pouco investigados e a definição de métodos e modelos de desenvolvimento de software que possam ser aplicados com sucesso em projetos envolvendo arquiteturas orientadas a serviços permanecem ainda em aberto (Lübke, 2007).

Uma das principais abordagens consideradas no desenvolvimento de aplicações orientadas a serviços é o uso de modelos de processos de negócio baseados em *Workflow*. Recentemente, Cardoso (2006) discutiu a falta de métodos de análise e desenho para o desenvolvimento de aplicações baseadas em *Workflow* que utilizem Serviços Web. Em seu trabalho o autor propõe um arcabouço chamado Poseidon. Poseidon é um arcabouço conceitual constituído de passos, procedimentos e algoritmos para guiar o desenho de soluções de softwares baseados em tecnologias da Web e *Workflow*. O Poseidon organiza o desenvolvimento baseando-se em casos de negócio (business cases) e considerando o emprego de mecanismos de desenho de *Workflow* (*Workflow* patterns) (van der Aalst et al., 2000) na especificação da lógica de negócios da aplicação.

Dois aspectos fundamentais diferenciam a abordagem do Poseidon do método BASS. O

```
<!-- SAVE BindEntryCollection -->
<save>
  <bindEntry className= "SupplierABindingStub">
    <operations>
      <operation name="makeInvoice"
        returnType="" MappedToField="">
        <parameters>
          <parameter name="invoice"
            paramType="Invoice" category="javabean" >
          </parameter>
        </parameters>
      </operation>
    </operations>
  </bindEntry>
  <bindEntry  className="SupplierBBindingStub">
  <!--same structure of the bind entries above-->
  </bindEntry>
  <bindEntry className="SupplierCBindingStub">
  <!--same structure of the bind entry above-->
  </bindEntry>
</save>
```

Figura 3.7: Excerto de um Arquivo de Ligação Associado a *EntityProxy* Pedido Mostrando a Ligação {Pedido,save}.

```
pedido order invoice request
numero number id orderId
descrição description specification
preço price amount value
data date orderDate
empregado employee
```

Figura 3.8: Vocabulário do Pedido de Compra.

primeiro é que o Poseidon é um arcabouço conceitual e não define uma tecnologia de integração específica. Já o método BASS pressupõe a existência de um arcabouço de software. O segundo aspecto é que o Poseidon parte do princípio que a integração vai ser definida por meio de modelos de processos de negócio baseados em *Workflow*. No BASS, o objetivo foi tornar modelos de processos de negócio, por exemplo, envolvendo composição de serviços Web e mecanismos de desenho de *Workflow*, transparentes para o desenvolvedor da camada de aplicação (conforme Figura 3.2).

O desenvolvimento de software baseado em *Workflow* e Processos de Negócio tem sido amplamente considerado na Computação Orientada a Serviços. Em um trabalho inicial, Papazoglou e Yang (2002) propõem princípios de desenho para Serviços Web e Processos de Negócio. Nesse trabalho é discutida a separação entre serviços e a especificação de processos de negócio. Dois princípios discutidos são *a separação de interfaces de serviços de sua implementação* e a *utilização de linguagens de definição de processos de negócio baseadas em serviços*. Na integração entre clientes e serviços, a aplicação cliente deve descrever a implementação da lógica de negócios em termos de processos de negócio, por exemplo, por meio de definições de processo utilizando Web Services Flow Language (WSFL) (IBM, 2001). Tais

definições mapeiam os processos de negócio dos clientes em interfaces de serviços.

A adoção de descrições de processos de negócio baseadas em serviços tem sido adotada por outros modelos e tecnologias, por exemplo, pela linguagem BPEL. Uma das vantagens desses modelos é possibilitar a utilização de sistemas de gerenciamento de processos de negócio para a execução dos processos. Contudo, a utilização desses sistemas não elimina a necessidade de se desenvolver aplicações de domínio específico que devem ser integradas aos processos suportados. Estabelecendo um paralelo entre linguagens de execução de Serviços Web e o método BASS, temos que o essas tecnologias podem vir a ser complementares, por exemplo com o desenvolvimento de extensões no BASS que permitam o mapeamento de operações RLSDU em composições descritas em BPEL. Nesse caso, a vantagem de utilizar o BASS é, novamente, tornar transparente, para o desenvolvedor da camada de aplicação (conforme Figura 3.2), a complexidade da definição de processos de negócio usando linguagens de execução de serviços Web.

Quando modelos de processos de negócio baseados em Serviços Web são codificados diretamente em aplicações de domínio específico, essa codificação implica em se fazer referências a interfaces de serviços diretamente no código que implementa a lógica de negócios da integração. Nesses casos, computações que utilizam serviços Web têm sido explicitamente programadas nos módulos da aplicação, por exemplo, utilizando *stubs*. Esse aspecto tem impactado em atividades de implementação e manutenção da solução de integração. Alguns dos problemas resultantes dessa abordagem têm sido observados e algumas soluções têm sido apresentadas na literatura.

Eberhart (2002) propõe o arcabouço conceitual *Web Service Description Framework* (WSDF). Esse arcabouço trata o aspecto de dinamismo de aplicações orientadas a serviços baseando-se em modelos da Web Semântica e Ontologias. WSDF provê mecanismos tanto para a descrição quanto para a ligação e invocação dinâmica de serviços. O principal propósito de WSDF é permitir que um cliente possa acessar um serviço sem o conhecimento prévio de sua descrição. WSDF não é um arcabouço de software, mas uma especificação conceitual, arquitetural e de desenho de sistemas. WSDF requer a utilização do padrão de desenho *Observer* (Gamma et al., 1995) tanto no desenho de clientes quanto de serviços. O objetivo do *Observer* é descrever um mecanismo que estabelece uma relação um-para-muitos entre objetos, de forma que, quando um objeto muda o seu estado, todos os seus objetos dependentes são notificados. WSDF define o uso de meta-informações sobre serviços tanto no lado cliente quanto nos serviços, por meio de anotações semânticas utilizando RDF (*Resource Description Framework*) (Lassila e Swick, 2002). O *Observer* deve ser utilizado no desenho de serviços e clientes para notificar mudanças nos serviços mantendo essas anotações atualizadas.

Com relação ao arcabouço WSDF é importante observar que o mesmo requer o estabelecimento de regras de desenho e implementação que devem ser adotadas tanto nos cliente quanto nos serviços. Atualmente, a definição e interpretação de aspectos semânticos dos processos de negócio no BASS são delegados ao desenvolvedor. Em princípio, no método BASS procuramos não estabelecer nenhum tipo de restrição quanto à definição dos serviços. No entanto, tais padronizações podem vir a ser consideradas, por exemplo, para o atendimento de requisitos relacionados à semântica de processos de negócio por mecanismos automáticos.

B. Verheecke e Jonckers (2003) abordam o problema da perda de flexibilidade causado pela codificação explícita do acesso a serviços. Os autores propõem uma camada intermediária entre aplicações e serviços chamada de *Web Services Management Layer* (WSML). A camada WSML permite o uso de Serviços Web sem explicitar diretamente a invocação desses serviços na lógica de negócios da aplicação. A principal tecnologia utilizada pela WSML é a programação Orientada a Aspectos. Por meio da programação de aspectos, são definidos elementos chamados *Service Types*. A lógica de negócios da integração é implementada por meio desses services types. Quando esses elementos são utilizados, a programação de aspectos

permite que a invocação de serviços seja realizada apenas na camada WSML.

Baligand e Monfort (2004) apresentam uma abordagem similar, também utilizando Aspectos. Entretanto, o seu foco é a utilização de políticas de *QoS* e mecanismos, considerando essas políticas, para determinar a invocação de serviços. Basicamente, os autores propõem um modelo de implementação para programar a as políticas de segurança, explicitadas usando *WS-Security* (Della-Libera et al., 2003), de maneira transparente. Com a programação de aspectos os elementos de implementação necessários para a utilização dessas políticas são removidos da lógica de negócio da aplicação e invocados dinamicamente.

Tanto a camada WSML quanto a proposta para *WS-Security* apresenta por Baligand e Monfort (2004) apresentam soluções semelhantes a proposta no método BASS. Nessas propostas os autores também procuram encapsular aspectos tecnológicos e disponibilizar elementos de programação mais apropriados para o desenvolvimento da solução de Integração. Também experimentamos, em nossa investigação inicial, o uso de Aspectos para o suporte à seleção e execução dinâmica de serviços Web (Costa et al., 2004b). Entretanto, nossa solução baseada em Aspectos se mostrou com alto custo de implementação, introduzindo elementos não intuitivos na definição da lógica de negócios.

Quanto às propostas de B. Verheecke e Jonckers (2003), Baligand e Monfort (2004), observamos que as mesmas mantém a noção de serviço na definição da lógica de negócios da integração. Acreditamos, baseados nos experimentos que realizamos, que o modelo conceitual proposto no BASS e as operações RLSDU, são abstrações mais favoráveis ao desenvolvimento da lógica de negócios da integração. Entretanto, não foi possível estabelecer comparações diretas entre essas propostas e o método BASS. Os autores também não divulgaram nenhuma evidência, de caráter experimental, dos benefícios que podem ser obtidos com suas propostas.

Uma importante iniciativa para a acomodação de tecnologias baseadas em serviços é o Enterprise Service Bus (ESB) (Schmidt et al., 2005). O objetivo de um ESB é prover mecanismos que facilitem a definição de elementos operacionais necessários para a integração entre clientes e serviços. Um ESB é definido por um conjunto de meta-dados que descrevem clientes e serviços, mecanismos de mediação e outras operações que devem ocorrer na integração. Um ESB também visa descrever mecanismos de roteamento, descoberta e pareamento (*matchmaking*) necessários em arquiteturas orientadas a serviços. Resumidamente, um ESB provê uma camada de conectividade entre clientes e serviços.

ESBs não estabelecem nenhum tipo de mecanismo de suporte à descrição ou implementação da lógica de negócios de aplicações por meio de serviços. Assim, ESBs e o método BASS podem ser vistos como complementares, dado que o arcabouço BASS não estabelece mecanismos de busca e descoberta de serviços.

Uma iniciativa importante no desenvolvimento de software, da qual o BASS se aproxima é a Arquitetura Dirigida por Modelos, discutida na Seção 2.4.1.4. De maneira similar a modelos de desenvolvimento baseados em MDA, o método BASS organiza o desenvolvimento em um nível conceitual, independente de aspectos computacionais, um nível lógico, independente de tecnologias específicas e em um nível físico no qual tecnologias são especificadas e introduzidas. Os níveis Conceitual, Lógico e Físico do BASS correspondem, respectivamente, ao Modelo Independente de Computação (CIM), ao Modelo Independente de Plataforma (PIM) , e ao Modelo Específico de Plataforma (PSM) da MDA.

Algumas diferenças entre MDA e o método BASS podem ser observadas. A abordagem MDA visa à definição de soluções genéricas de desacoplamento entre problema e solução. Já o método BASS focaliza o domínio específico de Integração de Sistemas de Informação. Um outro aspecto é que, no método BASS, aspectos tecnológicos foram considerados em seus três níveis de abstração. Conforme já discutido, o método BASS foi concebido para a utilização do arcabouço. Por outro lado, o arcabouço foi concebido visando atender os requisitos do domínio de aplicação. Dessa forma, a tecnologia (o arcabouço BASS) e o modelo de desen-

volvimento foram fortemente influenciados um pelo outro. Em modelos baseados em MDA, em geral, admitem-se todos os aspectos da tecnologia e especifica-se um PSM para cada uma das tecnologias admitidas. Especifica-se também um PIM apropriado para que sejam realizadas *transformações* automáticas entre o PIM e os PSM. No BASS, optamos por modificar a tecnologia e proporcionar modelos de desenho mais apropriados para o mapeamento dos modelos conceituais. As transformações previstas no MDA, no método BASS são realizadas por atividades de desenho e implementação e por atividades de personalização do Arcabouço.

3.4 Experimentos

Realizamos estudos experimentais com o objetivo de caracterizar a utilização do arcabouço BASS e das técnicas necessárias à sua aplicação. Os estudos foram realizados com alunos da disciplina de Engenharia de Software ofertada pelo DCC/UFMG. Para a definição e realização desses estudos nos baseamos no método de experimentação apresentado por Wohlin et al. (2000).

Nosso principal objetivo foi verificar se o uso do BASS pode reduzir o esforço de desenvolvimento da integração. Para realizar essa verificação projetamos experimentos de comparação aos pares, do tipo *um fator e dois tratamentos*. O fator observado foi o *método de desenvolvimento da integração*. Os dois tratamentos foram: (1) um método de desenvolvimento baseado no Processo Unificado, que denominamos de *método ad hoc* e (2) o *método BASS*.

Para o projeto dos experimentos formulamos as seguintes hipóteses:

- $d_i = e_{1i} - e_{2i}$ é a distância (diferença) entre cada medida de um par de medidas de esforço ($e_{1i} - e_{2i}$). μ_d a média dessas distâncias. Então,

    1. $H_0 : \mu_d = 0$. H_0 é a hipótese Nula , que deve ser rejeitada. Definimos H_0 como a possibilidade de que o esforço médio usando o método BASS e o método *ad hoc* sejam o mesmo.
    2. $H_1 : \mu_d \neq 0,\ \mu_d < 0$ *or* $\mu_d > 0$. H_1 é a hipótese alternativa, a qual pode ser aceita pela rejeição de H_0. H_1 é a possibilidade do esforço médio usando o método BASS ser menor ou maior que o esforço médio usando o método *ad hoc*.

A medida de esforço (variável dependente) foi o tempo gasto em atividades de desenho e implementação para o obtenção da solução de software para integração. No estudo, realizamos três experimentos com essas formulações. Os experimentos diferem um do outro com relação a aspectos de instrumentação, conforme Tabela 3.4. Todos os experimentos foram realizados com alunos da disciplina Engenharia de Software cursada normalmente por alunos do sétimo período do Bacharelado em Ciência da Computação da UFMG. Os estudantes participantes (sujeitos) possuem um nível de conhecimento e experiência prática que variando de intermediário a alto com relação à linguagem Java e Serviços Web.

O experimento I foi realizado no semestre 2007-1. Os experimentos II e III foram realizados em 2007-2. Os objetos de estudo dos experimentos foram os seguintes:

- Caso de uso Gerenciamento de Pedidos: Desenvolvimento da funcionalidade de gerenciamento de Pedidos para uma aplicação que permite a um cliente ser integrado a três fornecedores. Os seguintes fluxos desse caso de uso foram requisitados:

    1. Procurar por produtos em uma lista de fornecedores;
    2. Fazer um pedido em um fornecedor específico;
    3. Fazer mudanças em um pedido;

4. Verificar um pedido;
5. Excluir um pedido.

- Caso de uso Gerenciamento de Reservas: Desenvolver a funcionalidade de Reserva em hotéis para uma aplicação de uma Agência de Viagens. A funcionalidade deve integrar a agência com um conjunto de hotéis, permitindo a realização de reservas. Foi requisitada a integração com dois hotéis. Os seguintes fluxos foram requisitados:

  1. Procurar por acomodações em uma lista de hotéis;
  2. Fazer a reserva de uma acomodação preferida;
  3. Consultar uma reserva;
  4. Fazer mudanças em uma reserva;
  5. Cancelar uma reserva;
  6. Entrar em uma lista de espera de um hotel.

Os sistemas dos fornecedores e dos hotéis foram previamente implementados e seus processos públicos disponibilizados utilizando serviços Web. Serviços Web foram implementados utilizando o servidor de aplicações Apache Tomcat com o suporte do Apache Axis 1.4 e 2.0. Em ambos os tratamentos os sujeitos utilizaram a API Axis para clientes de serviços Web. A linguagem de programação utilizada foi Java.

Foram fornecidos os seguintes artefatos para os participantes:

- Tutorial de desenvolvimento do *método ad hoc*;
- Tutorial de desenvolvimento do método BASS;
- Especificação de requisitos dos casos de uso;
- Pacote de projeto com interfaces de usuário pré-implementadas para os casos de uso. O pacote continha também as especificações dos processos públicos dos fornecedores e hotéis em termos de *stubs*. Esses componentes foram os mesmos para os dois tratamentos;
- Um guia de participação que incluiu o método de apreensão de tempo, prazos e formato das entregas.

Os pacotes de projeto fornecidos seguiram a arquitetura de software apresentada na Figura 3.2, para os dois tratamentos. A camada de Interface foi fornecida já completamente implementada. Os mecanismos de integração com os serviços (*stubs*) também foram fornecidos. Coube aos desenvolvedores implementarem a camada de Controle e a camada de Entidade, para os dois tratamentos, e personalizar o arcabouço BASS para o tratamento envolvendo o método BASS.

É importante lembrarmos que o estudo proposto não teve a intenção de realizar medidas relativas à curva de aprendizado de cada tratamento. Dessa forma, foram também conduzidos treinamentos cobrindo todos os aspectos necessários para a realização dos experimentos. Para verificar se os sujeitos estavam habilitados para iniciar os experimentos foram requisitados exercícios de compreensão envolvendo aspectos dos dois tratamentos.

Para os experimentos envolvendo o caso de uso Gerenciamento de Pedidos, foi solicitada a medida do tempo gasto para a finalização da integração com cada um dos fornecedores. Dessa forma, os dados coletados formaram três conjuntos de pares de medidas. Para os experimentos envolvendo o caso de uso Gerenciamento de Reserva foi medido o tempo gasto para finalizar

Tabela 3.1: Definição dos Experimentos.

	Experimento I	Experimento II	Experimento III
Desenho do experimento	Comparação de Pares	Comparação de Pares	Comparação de Pares
Objetos	Gerenciamento de Pedidos	Gerenciamento de Pedidos	Gerenciamento de Reservas
Número de sujeitos	cinco	oito	sete
Medidas	Tempo gasto para integração de cada fornecedor. Resultado: três conjuntos de cinco pares de medidas	Tempo gasto na integração de cada fornecedor. Resultado: três conjuntos de oito pares de medidas	Tempo gasto na integração com cada hotel. Resultado: dois conjuntos de cinco pares de medidas.
Entregas	Código fonte da Integração desenvolvida usando os dois tratamentos. Tabela contendo as medidas de tempo em horas.	Código fonte da Integração desenvolvida usando os dois tratamentos. Tabela contendo as medidas de tempo em horas.	Código fonte da Integração desenvolvida usando os dois tratamentos. Tabela contendo as medidas de tempo em horas.
Treinamento	Quatro horas de treinamento coletivo	Quatro horas de treinamento coletivo	Quatro horas de treinamento coletivo
Teste Estatístico	Teste T de Student	Teste T de Student	Teste T de Student

a integração com cada um dos hotéis. Foram coletados dois conjuntos de pares de medidas. Os aspectos de desenho e instrumentação dos experimentos foram resumidos na Tabela 3.4.

No experimento I fornecemos primeiramente a especificação do fornecedor 1 e do fornecedor 2. Somente após a conclusão da integração desses dois fornecedores é que fornecemos aos sujeitos, a especificação do fornecedor 3. Para evitar interferências da ordem de uso dos tratamentos, em todos os experimentos um grupo de sujeitos iniciou o desenvolvimento com um método BASS e o outro iniciou com o método *ad hoc*. Posteriormente os grupos trocavam de tratamento.

3.4.1 Resultados

A Figura 3.9 apresenta o tempo de desenvolvimento gasto por cada sujeito, usando os dois tratamentos. A Figura 3.10.a apresenta as diferenças de esforço entre os dois tratamentos. Esse gráfico pode ser visto também como um gráfico de espalhamento. Conforme podemos visualizar, fora um ponto, os dados estão concentrados entre 20% e 80% de redução de esforço. Essa redução se deu em atividades de desenho e implementação da solução, com a utilização do método BASS. O ponto fora da área de concentração, embora pudesse ser removido, foi mantido para ilustrar um problema decorrente da curva de aprendizado específica do sujeito. A Figura 3.10.b apresenta as reduções médias de esforço em cada experimento. Conforme pode ser visto, essa redução média foi de 52,23%, 35,78% e 50,18% nos experimentos I,II e III, respectivamente. As tabelas 3.2, 3.4.1 e 3.4 sumarizam os tempos medidos nos três experimentos. Esses tempos detalham o quanto foi gasto na implementação da integração com cada serviço. Por exemplo, no caso da Gerenciamento de Pedidos, foram medidos os tempos para integração com os Fornecedores um, dois e três separadamente. Essas tabelas também mostram a diferença de esforço $d_i = e_{1i} - e_{2i}$ em valores absolutos medidos em horas

e o valor percentual da redução utilizando o método BASS.

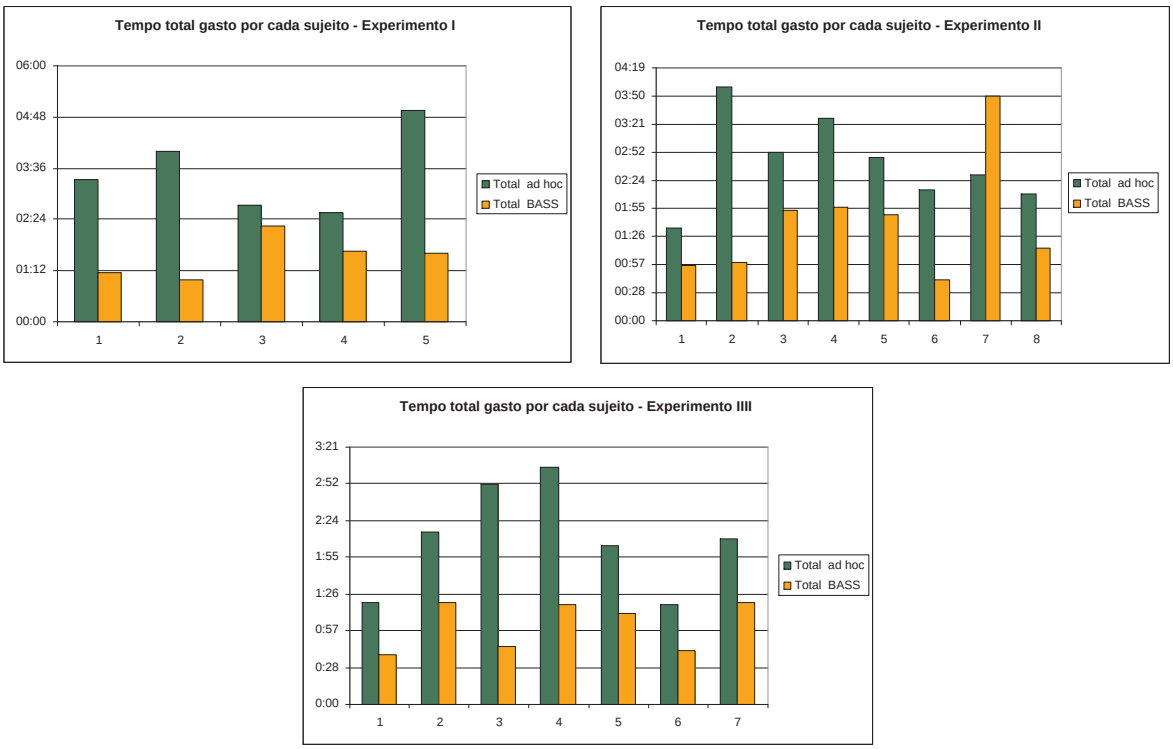
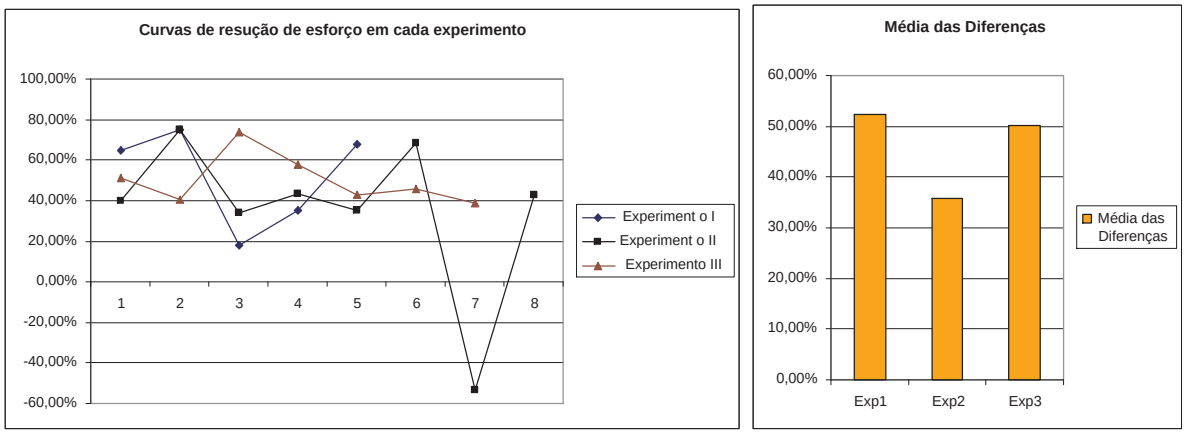


Figura 3.9: Tempo de Desenvolvimento Medido nos Experimentos I, II e III.



a) Diferença dos Esforços. b) Redução Média.

Figura 3.10: Medidas de Redução de Esforço.

Para avaliar a validade dos resultados obtidos nos experimentos realizamos um teste estatístico de medida de confiança denominado *Teste t*, na variação proposta por Student (Montgomery, 2006) para avaliação de conjuntos de pares de medidas. O teste t de Student é considerado um teste paramétrico rigoroso para avaliar a rejeição de hipóteses em comparações de medidas aos pares (Wohlin et al., 2000). A Tabela 3.5 sumariza os resultados do teste realizado. O *t stat* (calculado) não pode ser menor que o *t crítico* (dado por tabelas estatísticas) para um grau de liberdade definido. Aplicando-se o teste nos resultados dos três

Tabela 3.2: Tempos Gastos no Desenvolvimento da Integração com cada Fornecedor - Experimento I.

		Sujeitos				
		um	dois	três	quatro	cinco
Fornecedor 1	ad hoc	2:10	2:10	1:30	1:03	3:20
	BASS	0:45	0:25	1:10	0:55	0:53
Fornecedor 2	ad hoc	00:40	01:10	00:35	00:52	01:07
	BASS	00:15	00:15	00:40	00:21	00:27
Fornecedor 3	ad hoc	00:30	00:40	00:40	00:38	00:30
	BASS	00:10	00:20	00:25	00:23	00:16
Total	ad hoc	03:20	04:00	02:45	02:33	04:57
	BASS	01:10	01:00	02:15	01:39	01:36
Diferenças (d)	Absoluto	02:10	03:00	00:30	00:54	03:21
	%	65,00	75,00	18,18	35,29	67,68

Tabela 3.3: Tempos Gastos no Desenvolvimento da Integração com cada Fornecedor - Experimento II.

		Sujeitos							
		um	dois	três	quatro	cinco	seis	sete	oito
Fornecedor 1	ad hoc	0:39	2:10	1:16	2:33	2:07	1:15	1:10	1:08
	BASS	0:29	0:25	1:16	1:15	1:10	0:18	2:10	0:35
Fornecedor 2	ad hoc	0:29	1:10	0:51	0:26	0:22	0:33	0:40	0:38
	BASS	0:16	0:15	0:25	0:20	0:21	0:10	1:00	0:24
Fornecedor 3	ad hoc	0:27	0:40	0:45	0:29	0:18	0:26	0:40	0:24
	BASS	0:12	0:20	0:13	0:22	0:17	0:14	0:40	0:15
Total	ad hoc	1:35	4:00	2:52	3:28	2:47	2:14	2:30	2:10
	BASS	0:57	1:00	1:54	1:57	1:48	0:42	3:50	1:14
Diferenças	Absoluto	0:38	3:00	0:58	1:31	0:59	1:32	1:20	0:56
	%	40,0	75,0	33,72	43,75	35,33	68,66	-53,33	43,08

experimentos, verificamos que a hipótese*Nula* (H_0) pode ser rejeitada. Isto significa que a probabilidade de H_0 é menor que 5%.

Os resultados eram esperados, principalmente pela redução do esforço de programação necessário em implementações utilizando o método BASS. Usando o método BASS, por exemplo, quando um novo fornecedor é adicionado, é necessário apenas adicionar as ligações correspondentes ao fornecedor em arquivos XML.

Os experimentos mostraram a efetividade do método e do arcabouço BASS em atividades de desenho e implementação da solução de integração. Entretanto, acreditamos que os ganhos obtidos com o BASS possam se estender à atividade de análise de requisitos. Futuramente, esperamos realizar mais experimentos com medidas que possam também quantificar esforços nessa atividade.

Tabela 3.4: Tempos Gastos no Desenvolvimento da Integração com cada Hotel - Experimento III.

		Sujeitos						
		um	dois	três	quatro	cinco	seis	sete
Hotel 1	ad hoc	0:49	1:30	2:01	2:46	1:34	0:55	1:30
	BASS	0:21	0:50	0:28	0:57	0:45	0:25	1:00
Hotel 2	ad hoc	00:31	00:45	00:51	00:20	00:30	00:23	00:40
	BASS	00:18	00:30	00:17	00:21	00:26	00:17	00:20
Total	ad hoc	1:20	2:15	2:52	3:06	2:04	1:18	2:10
	BASS	0:39	1:20	0:45	1:18	1:11	0:42	1:20
Diferenças	Absoluto	0:41	0:55	2:07	1:48	0:53	0:36	0:50
	%	51,25	40,74	73,84	58,06	42,74	46,15	38,46

Tabela 3.5: *Teste t de Student* para os Experimentos I, II e III. $\alpha = 0,05$.

	Exp. I	Exp. II	Exp. III
Média das diferenças	52,23%	35,78%	50,18%
Grau de liberdade	4	7	6
t Stat	3,531	2,420	5,009
t Crítico	2,776	2,365	2,447
$P(H_0)$	0,024	0,046	0,002

Capítulo 4

Aspectos de Implementação

O esforço de desenvolvimento do BASS teve a intenção de fornecer uma prova de que é possível construir um arcabouço que atenda a requisitos da ISI, com o viés de facilitar o mapeamento entre requisitos e tecnologias que discutimos ao longo do trabalho. Neste capítulo apresentamos algumas decisões de desenho e implementação que foram importantes no desenvolvimento do BASS. Discutimos também alguns aspectos de uma ferramenta para a personalização do BASS que desenvolvemos, chamada *BASS Tuner*.

4.1 Objetos *Session*

O arcabouço BASS foi desenvolvido utilizando a linguagem Java. A Figura 4.1 apresenta a organização de pacotes do BASS. Os elementos da API do BASS são implementados nos Pacotes core, ocl e transaction. Informações estáticas da integração, tais como potenciais unidades de negócio associadas, são mantidos por meio de objetos da classe core.Session. Vários objetos core.Session podem co-existir, podendo estabelecer diferentes contextos de integração para uma mesma aplicação. Objetos core.Session mantêm informações relativas à integração que permitem o mapeamento e invocação de serviços. Objetos core.Session devem também executar as operações RLSDU.

Para a utilização de objetos core.Session, após a sua criação esses objetos devem ser inicializados. A inicialização compreende a execução do fluxo de atividades descrito na Figura 4.2. Inicialmente, todos as ligações definidas para a Sessão são lidos a partir dos arquivos de ligação. Posteriormente, também é lido o vocabulário da Sessão.

Para a execução de uma operação RLSDU específica, devem ser executadas as operações de serviço definidas em seu mapeamento. Cada operação de um serviço é executada com o auxílio de classes operation.Operation. A Figura 4.3 ilustra o fluxo de execução de uma operação. Para execução da operação, deve ocorrer uma associação entre a operação a ser executada e um método correspondente no *stub* do serviço mapeado. Quando o método é determinado, sua lista de parâmetros é definida e os valores desses parâmetros são passados. Quando o método é executado, o seu valor de retorno é recuperado para que seja retornado à aplicação.

Para a execução das operações RLSDU, uma ou mais operações podem ser mapeadas. O mapeamento destas operações é feito em função do conteúdo de expressões OCL ou de ligações pré-estabelecidas. No primeiro caso, a aplicação ainda não determinou quais as ligações (*bindings*) da entidade de negócio em questão. Esse é o caso das operações retrieve, list e save. No segundo, o elemento considerado na operação RLSDU é um objeto já existente. É o caso da execução de operações delete e update que devem ocorrer sobre entidades de negócio já existentes.

Para a determinação das ligações, a principal fonte de informação é a expressão OCL

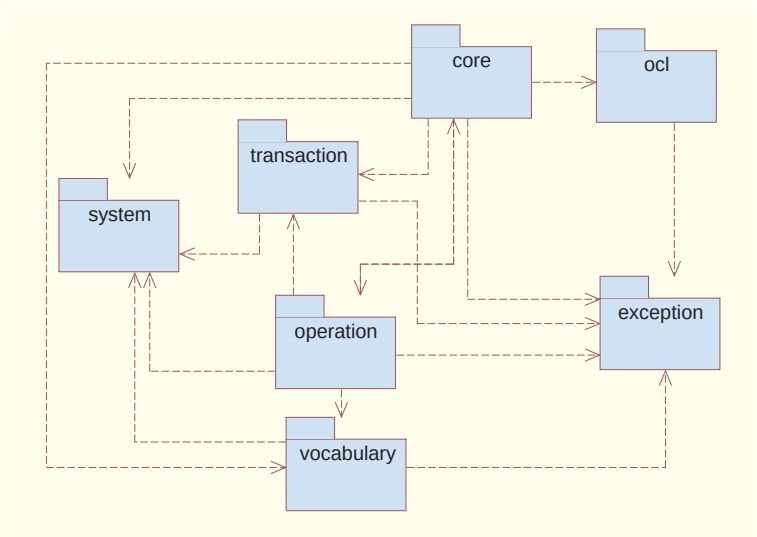


Figura 4.1: Pacotes do BASS.

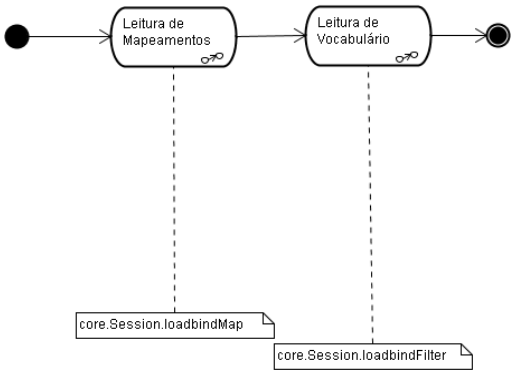


Figura 4.2: Inicialização do BASS.

passada como parâmetro. É utilizando essa expressão que uma das possíveis ligações relativas ao par {Operação RLSDU, EntityProxy} é escolhida. As Figuras 4.4, 4.5,4.6,4.7 e 4.8 apresentam o fluxo de atividades de cada uma das operações RLSDU.

As expressões OCL também fornecem valores de parâmetros para serem utilizados nas chamadas de operações. Para que esses valores sejam utilizados pelas operações, é necessário ocorrer um pareamento entre eles e os argumentos dos métodos correspondentes às operações. Esse pareamento é feito com o auxílio de classes mediadoras que implementam a interface *vocabulary.Mediator*. Um mediador tem a função de, dadas duas coleções de parâmetros, definidas por meio da classe core.ParameterCollection, mapeá-las corretamente entre si. No caso da execução de uma operação, uma coleção de parâmetros é fornecida pela expressão OCL e a outra coleção é definida por meio da descrição dos argumentos de uma operação de um serviço Web. Mediadores também realizam a tarefa reversa, de traduzir valores de retorno de operações de serviço em instâncias da classe EntityProxy associada.

Quando o elemento usado em uma operação RLSDU é um objeto EntityProxy, nas operações delete e update não ocorre determinação da ligação (*binding*). Isso porque toda vez que um objeto é retornado, ele é associado a um objeto do tipo core.Business e suas ligações são mantidas as mesmas durante a sua existência. Um objeto core.Business mantém as informações

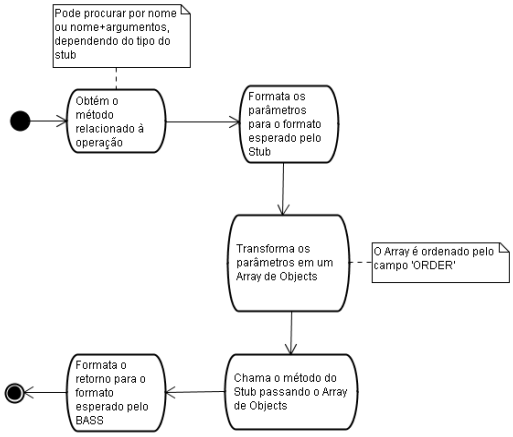


Figura 4.3: Execução de uma Operação.

de uma unidade de Negócio específica, ou seja uma instância de negócio.

O BASS prevê dois mecanismos para o gerenciamento de respostas assíncronas na camada de Aplicação: o Response Object e o Response Handler. O Response Object é o mecanismo padrão. Nesse caso, a chamada de uma operação RLSDU utiliza uma referência de objeto como argumento. Quando o objeto chega, o seu conteúdo é atribuído ao objeto referenciado. O Response Handler provê um mecanismo para personalizar o tratamento de respostas em interações assíncronas. Quando a resposta chega, essa é tratada por métodos definidos no Response Handler.

4.2 Personalização do BASS

Conforme mencionado na seção anterior, a inicialização de objetos core.Session consiste em dois passos principais: a leitura dos arquivos de ligação e a leitura do vocabulário. A leitura de arquivos de ligação é feita com auxílio da classe core.BindParser, no método core.Session.loadBindMap. Cada arquivo de ligação inclui todas as ligações de um EntityProxy com suas potenciais representações externas. Essas ligações são organizadas em termos das operações RLSDU. Ou seja, para cada par {*EnttiyProxy, operação RLSDU* } temos um conjunto de possíveis representações da operação RLSDU sobre a classe EntityProxy em questão, em termos de operações de serviços. As ligações mantêm toda a informação necessária para executar uma operação RLSDU sobre uma entidade de negócio em uma unidade de negócio específica. Essas informações compreendem:

1. O *stub* correspondente ao serviço;
2. A lista de operações no *stub* que devem ser executadas;
3. A lista de parâmetros de cada operação;
4. O mapeamento do valor de retorno das operações;

Os arquivos de ligação são arquivos definidos em XML (veja Figura 3.7). A construção desses arquivos deve refletir decisões da engenharia de negócios. É neste arquivo que são especificados os possíveis comportamentos de uma operação RLSDU sobre uma entidade de negócio específica. Mudanças nos requisitos da integração também irão refletir em mudanças nestes arquivos.

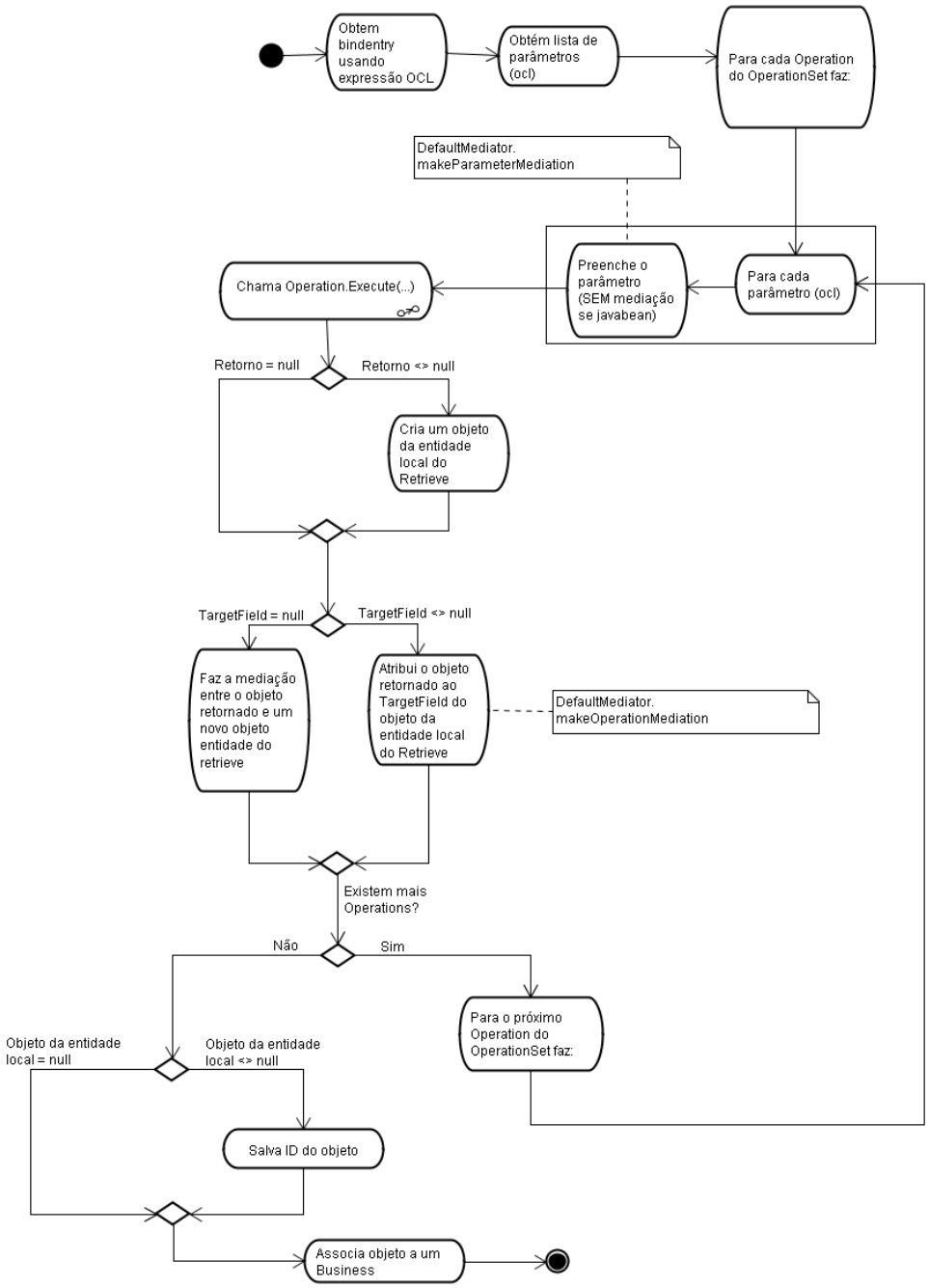


Figura 4.4: Operação Retrieve.

Esses arquivos são muito importantes, mas mesmo assim verificamos que deveria ser possível a definição desses arquivos por pessoas sem conhecimento de programação. Visando a esse objetivo, desenvolvemos no contexto de um Projeto Orientado de Curso no DCC-UFMG, a ferramenta *BASS Tuner* (Andrade et al., 2007). O objetivo do *BASS Tuner* é dar suporte para a geração semi-automática do código fonte de acesso aos serviços (*stubs*), arquivos de ligação e vocabulários envolvidos da integração. Por meio de interfaces gráficas, o BASS Tuner permite ao usuário:

1. Ter acesso a um registro de serviços e gerar o código fonte para acesso aos serviços

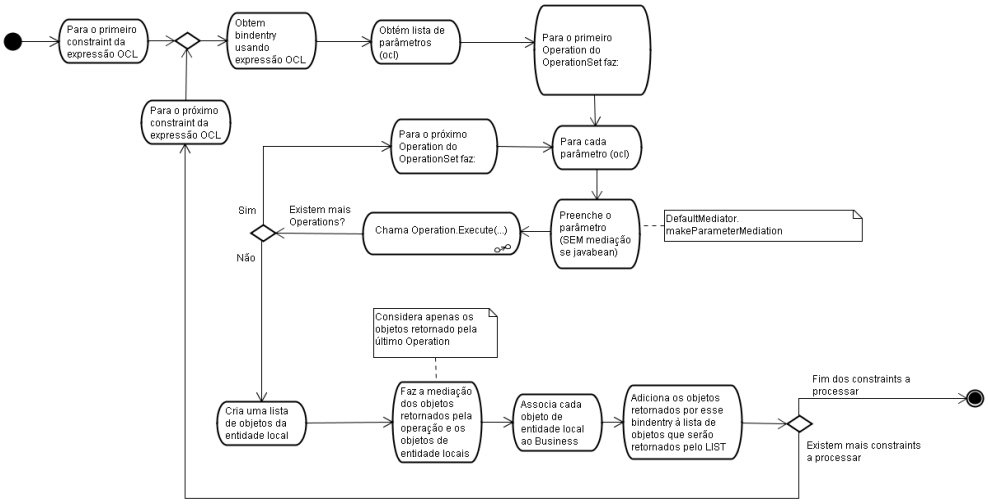


Figura 4.5: Operação List.

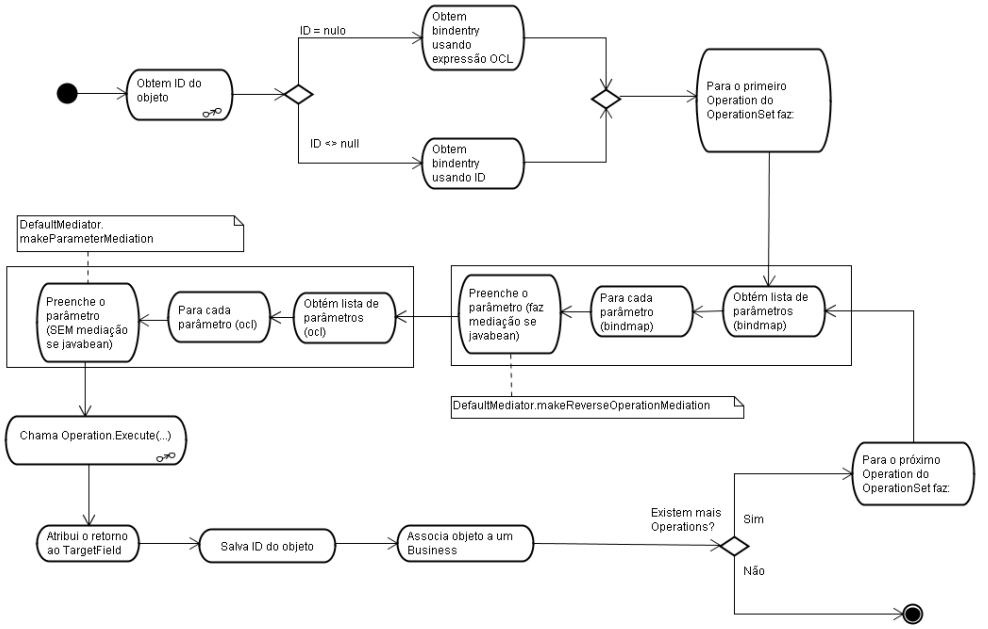


Figura 4.6: Operação Save.

desejados. A Figura 4.9 mostra a tela de geração des código.

2. Ter acesso ao conjunto de entidades de negócio definidas na aplicação e ao código gerado;

3. Definir as operações RLSDU de cada entidade de negócio, mapeando as operações nas operações dos serviços;

4. Definir o mapeamento dos valores de retorno das operações dos serviços nas entidades de negócio;

5. Gerar os arquivos de ligação a partir dos mapeamentos definidos de forma automática.

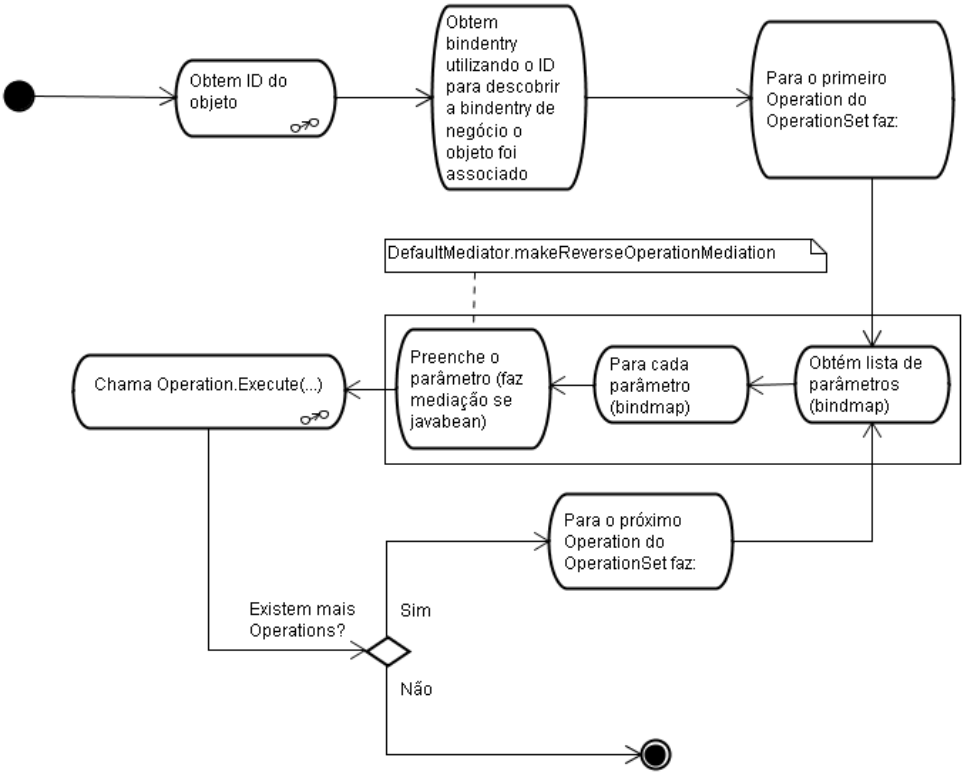


Figura 4.7: Operação Delete.

Para permitir ao *BASS Tuner* reunir informação das entidades de Negócio (EntityProxy) e *stubs*, é necessário uma pré-configuração dessas entidades e dos *stubs* no código fonte dos arquivos do projeto da aplicação. O esquema de pré-configuração é feito por meio de etiquetas (*tags*), de maneira semelhante à definida pela ferramenta para Programação Orientada a Atributos *XDoclet* (Walls et al., 2003). As entidades de negócio devem possuir uma etiqueta @bass.entity e as classes *stub* devem ter uma etiqueta @bass.soap. A Figura 4.10 mostra a tela de geração de arquivos de ligação.

Um outro aspecto do *BASS Tuner* é a noção de *projeto* de Integração. A idéia desse projeto é que sejam mantidas informações da integração em um contexto independente do projeto de software, mas ao mesmo tempo integrado com mesmo. Dessa forma, poderia haver um processo de desenvolvimento onde as atividades de personalização do BASS e de desenvolvimento da solução de software completa pudessem ocorrer em paralelo. No BASS Tuner temos o esboço dessa idéia implementado. Nele, um projeto armazena informações sobre a integração, como localização do projeto de software e localização dos arquivos de ligação.

Além da definição dos arquivos de ligação, a personalização do BASS requer a definição do vocabulário da integração. O vocabulário permite que a aplicação conviva com vários termos equivalentes. Espera-se que a definição do vocabulário da integração seja uma atividade prevista na modelagem dos processos de negócio da integração. No BASS, um vocabulário pode ser definido por meio de um arquivo texto simples, contendo um conjunto de listas de termos. Por exemplo:

$Termolocal_1 \;\; sinnimo_{11} \;\; sinnimo_{12} \;\; \ldots \;\; sinnimo_{1N}$
$Termolocal_2 \;\; sinonimo_{21} \;\; sinnimo_{22} \;\; \ldots \;\; sinnimo_{2N}$

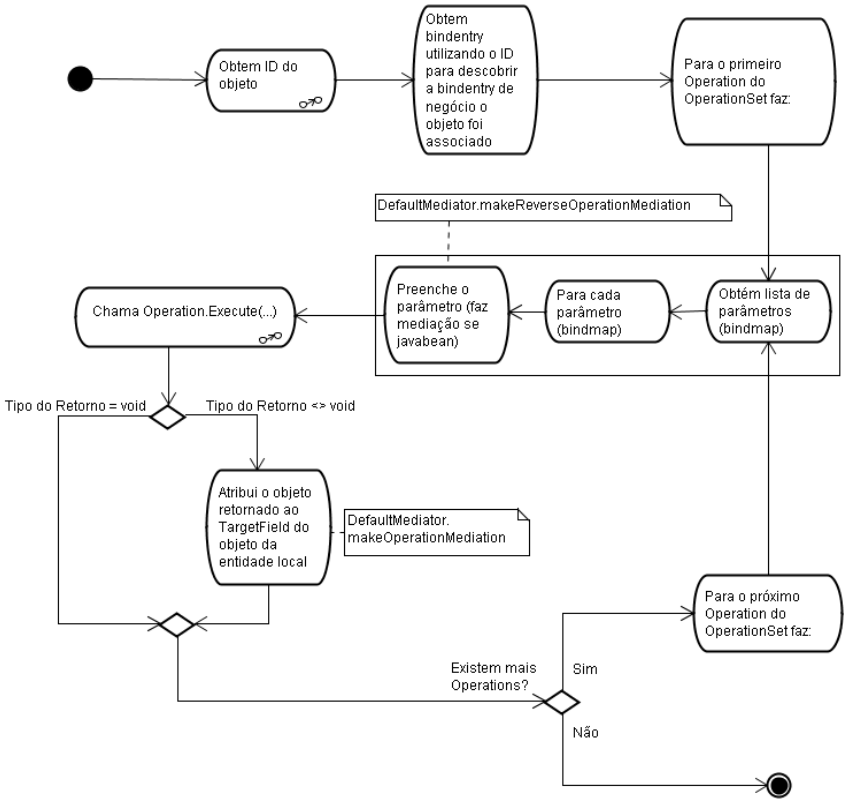


Figura 4.8: Operação Update.

⋮

$Termolocal_M$ $sinnimo_{M1}$ $sinnimo_{M2}$ ... $sinnimo_{MN}$

Para determinarmos o vocabulário podemos inferir os candidatos a termos locais aplicando as seguintes regras: (a) todo nome de membro de EntityProxy e (b) todo nome de variável de uma ExpressaoOCL. Já os candidatos a sinônimo consistem nos nomes de parâmetros das operações definidas nos serviços.

4.3 Invocação Dinâmica de Operações

O principal mecanismo que utilizamos no BASS para possibilitar a invocação dinâmica de operações foi Reflexão Computacional (Maes, 1987), por meio das classes disponibilizadas no pacote java.lang.reflect e também com auxílio da API JavaAssist (Chiba, 2000). Por meio de mecanismos de reflexão foi possível instanciar dinamicamente os *stubs*, identificar e executar os métodos correspondentes às operações definidas nas ligações de cada EntityProxy. As classes responsáveis por executar uma operação de um stub, por meio de reflexão, devem implementar a interface *operation.Operation*. Na versão atual, existem duas classes que implementam *operation.Operation*: operation.Axis1Operation, que suporta o acesso e execução de operações em *stubs* gerados com o Apache Axis 1.4 e operation.Axis2Operation que suporta o acesso e execução de operações em stubs gerados com o Apache Axis 2.0.

Novas tecnologias de integração podem ser facilmente adicionadas ao BASS por meio do desenvolvimento de novas implementações para *operation.Operation*. De fato, basta que a tecnologia utilize o conceito de stubs para que possa servir de middleware no método BASS.

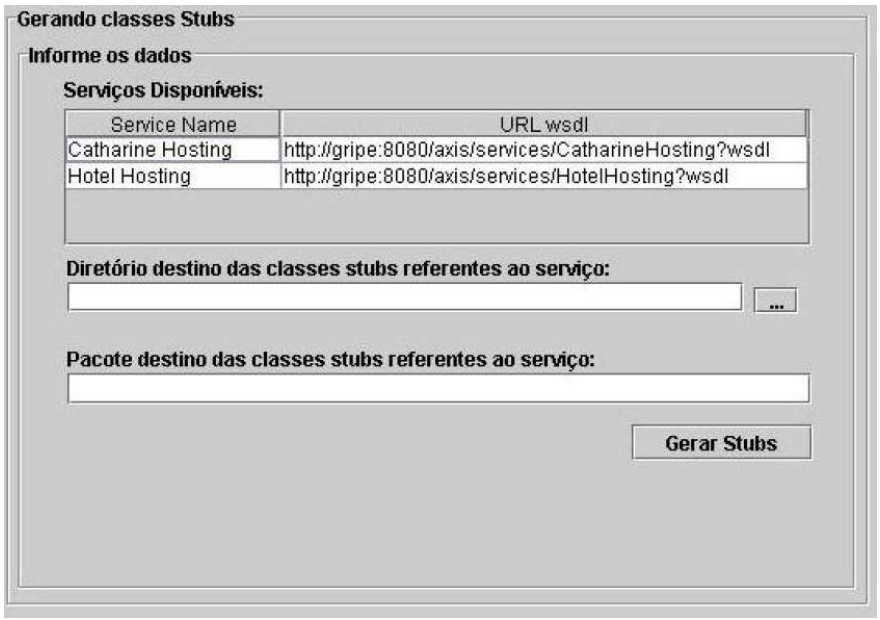


Figura 4.9: Tela de Geração de *stubs*.

A única restrição é a utilização de *stubs*. A interface *operation.Operation* não faz parte da API do BASS, sendo o seu uso, e o de suas implementações, um aspecto interno do arcabouço.

4.3.1 Mapeamento de Objetos

Na execução de operações é necessária a realização de mapeamentos entre objetos, tanto para a passagem de parâmetros quanto para o recebimento de valores de retorno. Conforme já discutido, esse mapeamento é feito por classes Mediadoras. Os algoritmos que realizam esses mapeamentos, na classe mediadora padrão (vocabulary.DefaultMediator), realizam inspeções nos membros dos objetos envolvidos verificando e detectando os membros correspondentes dos objetos em questão e fazendo a transferência de seus valores. Quando os objetos definidos pelos serviços envolvidos são bem estruturados em um modelo orientado a objetos que represente bem os conceitos do domínio da integração, essa operação é facilitada. Entretanto, nem sempre os tipos de dados são adequadamente modelados.

Durante o desenvolvimento do BASS, um dos exemplos que mais caracterizam essa situação é um pequeno estudo de caso que desenvolvemos envolvendo o serviço Web da Amazon.com (Amazon, 2006). O objetivo do estudo de caso foi avaliar o uso do BASS em um cenário que envolvesse serviços Web disponibilizados por terceiros e em operação. Assim, criamos uma aplicação de Livraria cujo objetivo é possibilitar a intermediação entre compradores e a Amazon para a venda de livros.

A Figura 4.11 ilustra as classes EntityProxy que definimos para poder realizar operações de consulta à entidade de negócio Livro. Foi necessária a definição das classes adicionais Solicitação e ItemResultado. A classe solicitação é usada no preâmbulo de uma consulta qualquer à Amazon. Já a classe ItemResultado é utilizada para ser mapeada no elemento de busca definido pela Amazon, chamado ItemSearchResponse.

Como o sistema da Amazon não categoriza ou classifica os itens, por exemplo, por meio das classes Livro, CD, DVD e Roupa, etc., toda consulta retorna sempre um elemento genérico. Só através de ItemSearchResponse é que conseguimos obter um elemento Livro. Assim, mapeamos o elemento ItemResultado nas operações que retornam o ItemSearchResponse. Finalmente, o

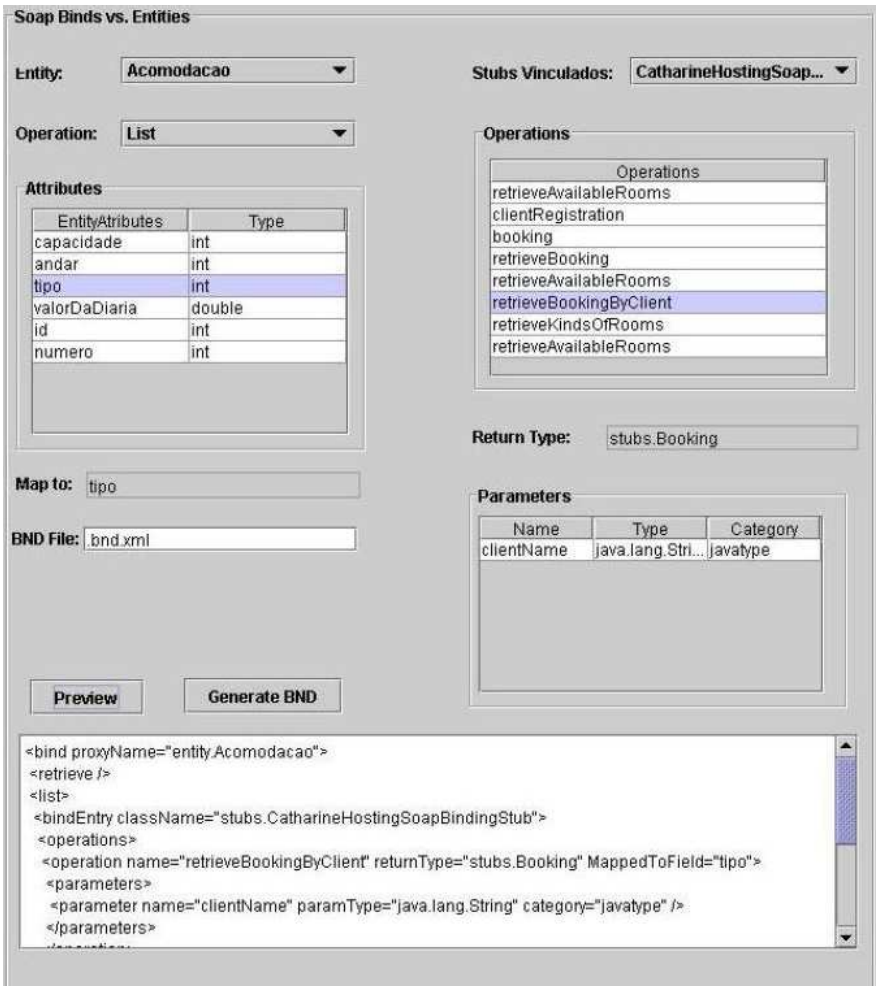


Figura 4.10: Interface do *Bass Tuner*: Tela de Geração de arquivos de ligação.

BASS monta o objeto Livro ou o conjunto de objetos Livro como uma coleção pertencente à ItemResultado.

A Figura 4.12 mostra os tipos de dados da Amazon envolvidos em uma consulta. As informações sobre um livro estão contidas nos objetos do tipo ItemAttributes. ItemAttributes é uma classe com centenas de membros que pretende descrever todo e qualquer produto comercializado pela Amazon. Quando um ItemResultado é requisitado, o BASS varre todos os objetos que constituem o ItemSearchResponse e monta a lista de livros associada ao ItemResposta. Embora o mapeamento com os tipos de dados da Amazon tenha sido complexo, o mesmo foi possível de ser realizado, permitindo a organização da camada de Aplicação organizada em termos de elementos de domínio sem grande interferência das especificidades do serviço Web da Amazon.

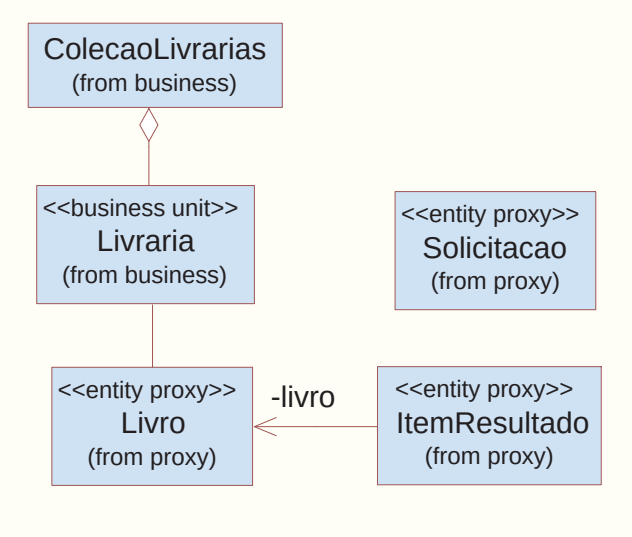


Figura 4.11: Classes *EntityProxy* Utilizadas na Consulta a Livros.

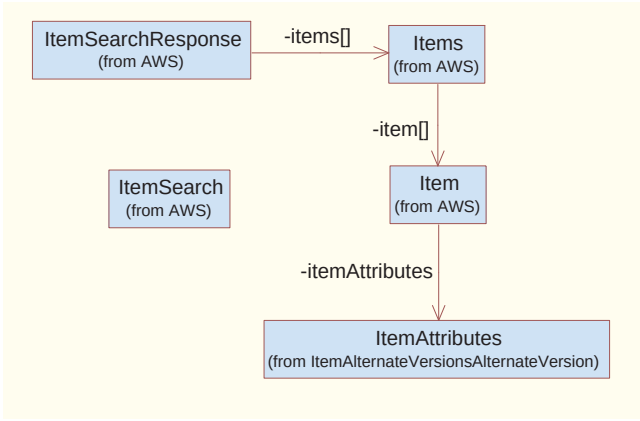


Figura 4.12: Tipos de Dados Definidos pelo Serviço Web da *Amazon*.

Capítulo 5

Conclusão

Um dos objetivos centrais da Engenharia de Software é conduzir o desenvolvimento de software, enquanto atividade econômica, a uma condição mais previsível, estável e segura. Entretanto, mesmo com a evolução dessa atividade quanto à sua capacidade de realização, desenvolver software continua sendo um atividade complexa e esse parece ser um objetivo difícil de ser alcançado (Randell, 1979; Gibbs, 1994; Jackson, 2003). Neste trabalho apresentamos nossa contribuição a esse objetivo com o desenvolvimento do arcabouço BASS. Neste capítulo apresentamos as principais contribuições e algumas propostas de trabalhos futuros.

5.1 Contribuições

Na etapa inicial do trabalho focalizamos a investigação na definição de aspectos relacionados às tecnologias de desenvolvimento no contexto de processos de software. Realizamos um estudo sobre transações de negócio e seu suporte por meio de tecnologias da Web. Procuramos investigar quais as melhores alternativas para o desenvolvimento de soluções de software para suportar modelos de integração baseados nessas transações.

A partir desse estudo propusemos uma organização para o desenvolvimento de soluções de software, considerando modelos apresentados na literatura (Costa et al., 2004a). Os principais objetivos dessa organização foram: (a) flexibilizar o desenvolvimento facilitando o mapeamento dos modelos de negócio, descritos por modelos conceituais, no modelo físico da solução de software e, (b) permitir a inserção de tecnologias necessárias para o atendimento dos diversos requisitos associados a transações de negócio no desenvolvimento da solução de software.

Avançamos nossa investigação no contexto de tecnologias para ISI, em particular na aplicação de tecnologias baseadas na Computação Orientada a Serviços. Apresentamos uma especificação arquitetural considerando o uso de Programação Orientada a Aspectos (Kiczales et al., 1997), para o problema de adaptação de uma aplicação a um conjunto heterogêneo de serviços Web (Costa et al., 2004b). Esse trabalho contribuiu, principalmente para a definição de aspectos arquiteturais do arcabouço BASS.

Posteriormente, focalizamos na concepção de um arcabouço de software que facilitasse o desenvolvimento da solução de software para ISI baseada em processos de negócio e utilizando tecnologias baseadas na Computação Orientada a Serviços. Os aspectos relevantes dessa integração foram organizados por meio de um catálogo de requisitos para arcabouços de software voltados para a ISI (Costa et al., 2008b). Nesse catálogo, consideramos aspectos funcionais e não funcionais da ISI e aspectos do desenvolvimento da solução de software para ISI.

Desenvolvemos o arcabouço de software *Business Application Support through Software*

Services (BASS), como modelo físico para a implementação de soluções para ISI (Costa et al., 2007). O principal mecanismo de implementação em que se baseia o BASS é reflexão computacional. Por meio de reflexão, foi possível tornar transparente, para desenvolvedores da solução de ISI, aspectos de implementação e da infra-estrutura de integração. Em particular, seguindo conceitos discutido no Capítulo 2, foi possível promover o desacoplamento entre o desenvolvimento da lógica de negócio dos processos privados e os processos públicos das organizações envolvidas nas colaborações. Em termos do desenvolvimento da solução de ISI, esse desacoplamento tornou a lógica de negócios da aplicação independente dos aspectos de implementação da integração, como, por exemplo, API's utilizadas na integração, e das idiossincrasias dos processos públicos disponibilizados pelos sistemas envolvidos nas colaborações.

Elaboramos também elementos de processo que facilitassem a utilização BASS no desenvolvimento de soluções para ISI. Os seguintes elementos foram definidos: (a) um meta-modelo conceitual para a análise de requisitos da integração, (b) um modelo arquitetural e lógico que permite o desenho da solução de integração (c) um modelo lógico para a infra-estrutura de integração que permite implementar, por meio de personalização, a solução de software da infra-estrutura de integração. O modelo lógico da infra-estrutura de integração foi elaborado concomitantemente com a implementação do BASS. Esses elementos de processo foram desenvolvidos e testados durante o desenvolvimento do BASS, em experimentos e estudos de caso. Os elementos de processo definidos foram apresentados de forma preliminar (Costa et al., 2007, 2008b) e posteriormente organizados em um método único que denominamos método BASS (Costa et al., 2008a).

No contexto da ISI, um aspecto que pudemos observar diz respeito à relação entre processo e tecnologia. O desenvolvimento do BASS representou o atendimento a necessidades de desenvolvimento por meio da extensão e modificação da tecnologia subjacente. Ao mesmo tempo, novos elementos de processo foram necessários para a utilização do BASS. À medida em que a tecnologia foi modificada, o processo precisou ser também modificado em função da tecnologia.

Realizamos experimentos para caracterização do método BASS com alunos da disciplina de Engenharia de Software, cursada usualmente por alunos do sétimo período do curso de Ciência da Computação da UFMG. Realizamos três experimentos. O objetivo dos experimentos foi confrontar a utilização do BASS com um método de desenvolvimento *ad hoc* (sem a prescrição de elementos específicos para o desenvolvimento de soluções para ISI). Em ambos os tratamentos (BASS e *ad hoc*) nos baseamos no Processo Unificado.

Nos experimentos, avaliamos o esforço dedicado a atividades de desenho e implementação de soluções para ISI. Os resultados dos experimentos mostraram a efetividade do uso do BASS com relação à redução do esforço de desenvolvimento de soluções de software para ISI utilizando tecnologias baseadas na Computação Orientada a Serviços (Costa et al., 2008b). Não realizamos experimentos com medidas relativas a atividades de análise de requisitos. Realizamos também, durante o desenvolvimento do BASS, implementações considerando serviços Web comercialmente difundidos e em operação (*Amazon.com*). A complexidade desses serviços e dos tipos de dados utilizados pelos mesmos foi um importante teste da aplicabilidade do BASS.

Uma contribuição que consideramos importante foi, em nosso método, posicionar o modelo de serviços quanto ao nível de abstração em que ele deve ser considerado (Costa et al., 2007). Esse é um aspecto importante e até o momento ainda em aberto. Modelos de serviços têm sido utilizados na descrição conceitual da integração e, ao mesmo tempo, constituem modelos físicos dessa integração. No que tange ao desenvolvimento de soluções para ISI, mostramos que, o modelo de serviços pode ser utilizado apenas como modelo de implementação. Esse aspecto permite salvaguardar especificações conceituais de sistema, como modelos de análise, das idiossincrasias dos serviços envolvidos em uma integração. No desenvolvimento da integração

de um sistema com muitos parceiros de negócio (situação esperada nas arquiteturas orientadas a serviços), descrever modelos conceituais, utilizando as diferentes interfaces de serviço e os diferentes tipos de dados empregados, pode ser um fator complicador. Com a utilização do BASS, a modelagem conceitual é feita considerando entidades do domínio de aplicação, de maneira muito semelhante à modelagem de dados comum ao desenvolvimento de sistemas de informação.

A utilização das operações RLSDU como abstração das operações disponibilizadas pelos serviços foi fundamental para permitir a separação entre modelos conceituais e serviços. Nos experimentos e exemplos implementados, apesar da variação do grau de dificuldade, todas as operações de integração necessárias puderam ser abstraídas em operações RLSDU. A utilização de expressões OCL para descrição de regras de negócio de integração e a sua tradução em termos de operações de serviços também se mostrou bastante conveniente. Junto com o meta-modelo proposto, a utilização de OCL permite descrições de sistemas independentes de aspectos de implementação que são transformáveis, por meio do método BASS, em modelos de implementação baseados em serviços.

Outra contribuição foi a definição preliminar de um mecanismo para personalização da solução de integração que remove tarefas de programação. Desenvolvemos de forma preliminar a ferramenta *BASS Tuner* baseada neste mecanismo. Esta implementação evidencia a possibilidade de personalização do BASS com base nos modelos conceituais da integração e nos processos públicos das organizações envolvidas.

5.2 Trabalhos Futuros

Planejamos diversos trabalhos futuros considerando o arcabouço BASS e o seu emprego no desenvolvimento de software. Esses trabalhos compreendem desde melhorias nos aspectos já existentes até a concepção de novos elementos. A seguir listamos alguns dos trabalhos que pretendemos realizar futuramente.

- Experimentos envolvendo atividades de análise de requisitos: Acreditamos que a análise de requisitos de soluções para ISI utilizando o método BASS pode ser significativamente facilitada. Entretanto não pudemos realizar medidas que comprovassem essa melhora. Assim, um das primeiras tarefas que pretendemos realizar é quantificar os esforços de análise de requisitos com a utilização do método BASS.

- *Workflow* conceitual: Pretendemos desenvolver uma linguagem diagramática para *Workflow* conceitual (Meyer-Wegener e Böhm, 1999) baseada nas operações RLSDU e considerando o meta-modelo conceitual proposto. Assim, uma aplicação descrita nesta linguagem, por exemplo, por analistas de negócio, iria definir a lógica de negócios da integração e gerar composições automáticas de serviços, dado que as operações RLSDU já possuiriam um mapeamento para serviços.

- Definição de um perfil UML considerando o meta-modelo conceitual definido na Seção 3.2.1. Na definição desse meta-modelo especificamos apenas de maneira informal os elementos que irão permitir a definição de um perfil de UML.

- Melhoria na modelagem e desenvolvimento do BASS: O foco do trabalho foi estabelecer um núcleo de implementação e uma arquitetura para o desenho que suportasse os requisitos da integração. Não focalizamos em aspectos de modelagem de arcabouços. Com esse núcleo, um dos trabalhos futuros que pretendemos é modelar o BASS como um Perfil da UML, e produzir plug-ins de geração semi-automática da camada de aplicação

também. Podemos ainda, por exemplo, utilizar a linguagem de modelagem de arcabouços de software UML-F (Fontoura et al., 2000). A partir desta modelagem poderemos localizar as modificações na implementação que sejam mais importantes.

- Focalização em aspectos não funcionais da ISI: Devido ao escopo do trabalho, não foi possível desenvolver suporte específico no BASS para diversos aspectos não funcionais, como por exemplo, aspectos de segurança. Pretendemos abordar esse tema em trabalhos futuros, por exemplo através da acomodação de mecanismos baseadas em WS-Policy (Box et al., 2003) no BASS.

- Ambiente de especificação para ISI: Muitos investimentos têm sido feitos em sistemas gerenciadores e ferramentas de modelagem de processos de negócio. Esses investimentos visam, dentre outros objetivos, reduzir esforços de desenvolvimento de software. Entretanto, a realização da integração de ISI continua demandando grandes esforços de desenvolvimento. Em nosso trabalho buscamos reduzir esses esforços, mas, sobretudo, considerando *a necessidade* desse desenvolvimento. Acreditamos que uma iniciativa promissora para continuarmos buscando esse objetivo é definir um ambiente de especificação de ISI capaz de integrar o desenvolvimento de software com o gerenciamento e modelagem de processos de negócio.

Referências Bibliográficas

Agarwal, N. e Rathod, U. (2005). Defining Success for Software Projects: an Exploratory Revelation. *International Journal of Project Management*, 24:358–370.

Aissi, S.; Malu, P. e Srinivasan, K. (2002). E-Business Process Modeling: The Next Big Step. *Computer*, 35(5):55–62.

Alliance, O. T. (2007). 2007a Open Travel Alliance Specifications. [Online] Available: http://www.opentravel.org.

Alonso, G.; Casati, F.; Kuno, H. e Machiraju, V. (2004). *Web Services - Concepts, Architectures and Applications*. Springer.

Amazon (2006). Amazon Web Service. [Online] Available: http://developer.amazonwebservices.com/.

Andrade, A. M.; Resende, R. F. e Costa, M. B. (2007). Uma Ferramenta de Suporte a Integração de Aplicações utilizando Serviços de Software. Projeto orientado de computação, Departamento de Ciência da Computação - DCC - UFMG, Belo Horizonte - MG - Brasil.

APACHE (2006a). Apache Axis. [Online] Available: http://ws.apache.org/axis/.

APACHE (2006b). Struts. [Online] Available: http://struts.apache.org/.

B. Verheecke, M. A. C. e Jonckers, V. (2003). AOP for Dynamic Configuration and Management of Web Services. In *Proceedings of the 3rd Int. Conference on Web Services, (ICWS-Europe'03)*.

Baligand, F. e Monfort, V. (2004). A Concrete Solution for Web Services Adaptability using Policies and Aspects. In *ICSOC '04: Proceedings of the 2nd Int. conference on Service oriented computing*, pp. 134–142, New York, NY, USA. ACM Press.

Bioernstad, B.; Pautasso, C. e Alonso, G. (2006). Enforcing Web Services Business Protocols at Run-Time: a Process Driven Approach. *International Journal of Web Engineering and Technology (IJWET)*, 2:396–411.

Box, D. et al. (2003). Web Services Policy Framework (ws-policy).

BPMI (2004). Business Process Modeling Notation. Specification Version 1.0. Technical report, Business Process Management Initiative - Object Management Group.

Bray, T.; Paoli, J. e Sperberg-McQueen, C. M. (1998). Extensible Markup Language (XML) 1.0 - W3C recommendation 10-february-1998. Technical Report REC-xml-19980210, World Wide Web Consortium.

Bussler, C. (2001). The Role of B2B Protocols in Inter-Enterprise Process Execution. In *TES '01: Proceedings of the Second International Workshop on Technologies for E-Services*, pp. 16–29, Berlin, DE. LNCS – Springer-Verlag.

Cardoso, J. (2006). Poseidon: a Framework to Assist Web Process Design Based on Business Cases. *Int. J. Cooperative Inf. Syst.*, 15(1):23–56.

Cardoso, J. et al. (2004). Quality of Services for Workflow and Web Services Processes. *Journal of Web Semantics*, 1(3):281–308.

Chessell, M.; Griffin, C.; Vines, D.; Butler, M.; Ferreira, C. e Henderson, P. (2002). Extending the Concept of Transaction Compensation. *IBM Syst. J.*, 41(4):743–758.

Chiba, S. (2000). Load-Time Structural Reflection in Java. In *ECOOP '00: Proceedings of the 14th European Conference on Object-Oriented Programming*, pp. 313–336, London, UK. Springer-Verlag.

Clements, P. C. (1996). A Survey of Architecture Description Languages. In *Proceedings of the 8th Int. Workshop on Software Specification and Design*. IEEE Computer Society.

Cockburn, A. (2001). *Writing Effective Use Cases*. Addison Wesley.

Costa, M. B.; Resende, R. F.; Alves, M. F. e Segatto, M. V. (2004a). Business-to-Business Transaction Modeling and WWW Support. In *BPM'2*, pp. 132–147. LNCS N. 3080 - Springer Verlag.

Costa, M. B.; Resende, R. F.; Nakamura, E. F. e Segatto, M. V. (2008a). Information Systems Integration Development based on Software Services (submitted). *Int. Journal of Cooperative Information Systems*.

Costa, M. B.; Resende, R. F.; Nakamura, E. F. e Segatto, M. V. (2008b). Software Frameworks for Information Systems Integration based on Web Services. In *ACM SAC'08*, pp. 777–783, Fortaleza - CE, Brazil. ACM Press.

Costa, M. B.; Resende, R. F.; Nakamura, E. F.; Segatto, M. V. e Fonseca, N. (2007). BASS: Business Application Support through Software Services. In *SEKE '07*, pp. 523–528, Boston, MA, USA. Knowledge Systems Institute.

Costa, M. B.; Resende, R. F.; Neto, P. S. e Alves, M. H. F. (2004b). Utilização de Aspectos no Desenvolvimento de Aplicações baseadas em Serviços Web. In *Anais do 1. Workshop Brasileiro de Desenvolvimento Orientado a Aspectos*.

Curbera, F.; Khalaf, R.; Mukhi, N.; Tai, S. e Weerawarana, S. (2003). Service-Oriented Computing: The Next Step in Web Services. *Communications of the ACM*, 46(10):29–34.

Curbera, F.; Nagy, W. e Weerawarana, S. (2001). Web Services: Why and How. In *OOPSLA'01: Proceedings of the Workshop on Object-Oriented Web Services*. ACM.

Dalal, S.; Temel, S.; Little, M.; Potts, M. e Webber, J. (2003). Coordinating Business Transactions on the Web. *IEEE Internet Computing*, 7(1):30–39.

Datta, S. e van Engelen, R. (2006). Effects of Changing Requirements: a Tracking Mechanism for the Analysis Workflow. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pp. 1739–1744, New York, NY, USA. ACM Press.

Davenport, T. H. (2000). *Mission Critical: Realizing the Promise of Enterprise Systems.* Harvard Business School Press, Boston, MA, USA.

Davis, A. M. (1993). *Software Requirements: Objects, Functions, and States.* Prentice-Hall, Inc.

De Michelis, G.; Dubois, E.; Jarke, M.; Matthes, F.; Mylopoulos, J.; Papazoglou, M. P.; Pohl, K.; Schmidt, J.; Woo, C. e Yu, E. (1998). Cooperative Information Systems: a Manifesto. In Papazoglou, M. P. e Schlageter, G., editores, *Cooperative Information Systems - Trends and Directions*, pp. 315–363. Academic Press Ltd.

Della-Libera, G. et al. (2003). Web Services Security Policy Language (wssecuritypolicy).

Deutsch, L. P. (1989). *Design Reuse and Frameworks in the Smalltalk-80 System*, pp. 57–71. ACM, New York, NY, USA.

Dey, A.; Mankoff, J.; Abowd, G. e Carter, S. (2002). Distributed Mediation of Ambiguous Context in Aware Environments. In *UIST '02: Proceedings of the 15th annual ACM symposium on User interface software and technology*, pp. 121–130, New York, NY, USA. ACM Press.

Eberhart, A. (2002). Towards Universal Web Service Clients. In *Proceedings of the Euroweb.*

Fayad, M. E.; Schmidt, D. C. e Johnson, R. E. (1999). *Building Application Frameworks: Object-oriented Foundations of Framework Design.* John Wiley & Sons, Inc., New York, NY, USA.

Feiler, P. H. e Humphrey, W. S. (1993). Software Process Development and Enactment: Concepts and Definitions. In *Proceedings of ICSP*, pp. 28–40.

Foster, H. et al. (2006). LTSA-WS: a Tool for Model-based Verification of Web Service Compositions and Choreography. In *ICSE '06*, pp. 771–774, New York, NY, USA. ACM Press.

Fraser, M. D.; Kumar, K. e Vaishnavi, V. K. (1991). Informal and Formal Requirements Specification Languages: Bridging the Gap. *IEEE Trans. Softw. Eng.*, 17(5):454–466.

Fuggetta, A. (2000). Software Process: A Roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pp. 25–34. ACM Press.

Gamma, E.; Helm, R.; Johnson, R. e Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, Boston, MA, USA.

Garlan, D. e Shaw, M. (1993). An Introduction to Software Architecture. In *Advances in Software Engineering and Knowledge Engineering*, volume I. World Scientific Publishing Company, River Edge, NJ.

Gibbs, W. W. (1994). Trends in Computing: Software's Chronic Crisis. *Scientific America*, 271(3):72–80.

Gray, J. (1981). The Transaction Concept: Virtues and Limitations. In *VLDB '1981: Proceedings of the Seventh International Conference on Very Large Data Bases*, pp. 144–154. VLDB Endowment.

Greenspan, S.; Mylopoulos, J. e Borgida, A. (1994). On Formal Requirements Modeling Languages: RML Revisited. In *Proceedings of the 16th Int. Conference on Software engineering*, pp. 135–147. IEEE Computer Society Press.

Guizzardi, G.; Wagner, G.; Guarino, N. e van Sinderen, M. (2004). An Ontologically Well-Founded Profile for UML. In *Proceedings of the 14th Int. Conference on Advanced Information Systems Engineering (CAISE'04)*. Springer Verlag - LNCS.

Gulla, J. A. (1996). A General Explanation Component for Conceptual Modeling in CASE Environments. *ACM Trans. Inf. Syst.*, 14(3):297–329.

Guo, J. (2006). Inter-Enterprise Business Document Exchange. In *ICEC '06*, pp. 427–437, New York, NY, USA. ACM Press.

Hasselbring, W. (2000). Information System Integration. *Commun. ACM*, 43(6):32–38.

Hibernate (2006). Relational Persistence for Java and .Net. [Online] Available: http://www.hibernate.org/.

Hoffmann, M. et al. (2004). Requirements for Requirements Management Tools. In *RE '04: Proceedings of the 12' Int. Requirements Engineering Conference,*, pp. 301–308, Washington, DC, USA. IEEE Computer Society.

Hohpe, G. e Woolf, B. (2003). *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Humphrey, W. S. (1995). *A Discipline for Software Engineering*. Addison-Wesley Longman Publishing Co., Inc.

IBM (2001). Web Service Flow Language (WSFL 1.0).

IBM (2003). Business Process Execution Language for Web Services Version 1.1. [Online] Available: http://www.ibm.com/ developerworks/library/specification/ws-bpel.

Ionescu, B. et al. (2005). A Testbed and Research Network for Next Generation Services over Next Generation Networks. In *TRIDENTCOM '05*, pp. 22–31, Washington, DC, USA. IEEE Computer Society.

Irani, Z.; Themistocleous, M. e Love, P. E. D. (2003). The Impact of Enterprise Application Integration on Information System Lifecycles. *Inf. Manage.*, 41(2):177–187.

ISO/IEC (1991). ISO/IEC 9126: Information Technology - Software Product Evaluation - Quality Characteristics and Guidelines for Their Use.

ISO/IEC (2000). SPICE: Software Improvement and Capability Determination: ISO 15504. Technical report, ISO/IEC.

ISO/IEC (2008). System and Software Engineering – Software Life Cycle Processes: ISO/IEC 12207. Standard, ISO.

Jackson, M. (2003). Why Software Writing is Difficult and Will Remain So. *Inf. Process. Lett.*, 88(1-2):13–25.

Jacobson, I.; Booch, G. e Rumbaugh, J. (1998). *The Unified Software Development Process*. Addison Wesley.

Jacobson, I.; Christerson, M.; Jonsson, P. e Övergaard, G. (1992). *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley.

Jalote, P. (2000). *CMM in practice: Processes for Executing Software Projects at Infosys*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Johnson, R. E. (1997). Frameworks = (components + patterns). *Comm. of the ACM*, 40(10):39–42.

Johnson, R. E. e Foote, B. (1988). Designing Reusable Classes. *Object-Oriented Programming*, 1(2).

Kalepu, S.; Krishnaswamy, S. e Loke., S. W. (2004). Verity: A QoS Metric for Selecting Web Services and Providers. In *Proc. of forth Int. Conference on Web Information System Engineering Workshops - WISEW'3*. IEEE Computer Society.

Köhler, H. J.; Nickel, U.; Niere, J. e Zündorf, A. (2000). Integrating UML Diagrams for Production Control Systems. In *ICSE*.

Kiczales, G. et al. (1997). Aspect-Oriented Programming. In *Proceedings of the European Conference on Object-Oriented Programming*, volume 1241, pp. 220–242. Springer-Verlag, Berlin, Heidelberg and New York.

Kilov, H. (1990). From Semantic to Object-Oriented Data Modeling. In *ISCI '90: Proceedings of the First International Conference on Systems integration '90*, pp. 385–393, Piscataway, NJ, USA. IEEE Press.

Lassila, O. e Swick, R. (2002). Resource Description Framework (rdf) Model and Syntax Specification.

Lübke, D., editor (2007). *Proceedings of the Workshop on Software Engineering Methods for Service-oriented Architecture 2007 (SEMSOA 2007), Hannover, Germany, online CEUR-WS.org/Vol-244/*, volume 1, Hannover, Germany. Leibniz University at Hannover, FG Software Engineering.

Leffingwell, D. e Widrig, D. (2000). *Managing Software Requirements: a Unified Approach*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Lindland, O. I.; Sindre, G. e Solvberg, A. (1994). Understanding Quality in Conceptual Modeling. *IEEE Softw.*, 11(2):42–49.

Linthicum, D. S. (2000). *B2B Application Integration: e-Business-Enable your Enterprise*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.

Liskov, B. (1987). Keynote Address - Data Abstraction and Hierarchy. In *OOPSLA '87: Addendum to the proceedings on Object-oriented programming systems, languages and applications (Addendum)*, pp. 17–34, New York, NY, USA. ACM Press.

Maes, P. (1987). Concepts and Experiments in Computational Reflection. *SIGPLAN Not.*, 22(12):147–155.

Mashkoor, A. (2004). *Investigating Model Driven Architecture*. Msc Dissertation, Umea University, Umea, Sweden.

McIlraith, S.; Son, T. C. e Zeng, H. (2001). Semantic Web Services. *IEEE Inteligent Systems*, 16(2):46–53. Special Issue on the Semantic Web.

Medjahed, B. et al. (2003). Business-to-Business Interactions: Issues and Enabling Technologies. *The VLDB Journal*, 12(1):59–85.

Meyer-Wegener, K. e Böhm, M. (1999). Conceptual Workflow Schemas. In *COOPIS '99: Proceedings of the Fourth IECIS International Conference on Cooperative Information Systems*, p. 234, Washington, DC, USA. IEEE Computer Society.

Montgomery, D. C. (2006). *Design and Analysis of Experiments*. John Wiley & Sons.

Mylopoulos, J.; Chung, L. e Nixon, B. (1992). Representing and Using Nonfunctional Requirements: A Process-Oriented Approach. *IEEE Trans. Softw. Eng.*, 18(6):483–497.

OASIS (2006). Web Services Security Specification. [Available:] http://www.oasis-open.org/.

OMG (2003). Object Constraint Language Description. [Online] Available: http://www.omg.org.

OMG (2004). Model Driven Architecture. [Online] Available: http://www.omg.org/mda.

OMG (2007). UML Infrastructure Specification. [Online] Available: http://www.omg.org/docs/formal/07-11-04.pdf.

Osterweil, L. (1987). Software Processes are Software Too. In *ICSE'87: Proceedings of the 9th Int. Conference on Software Engineering*, pp. 2–13. IEEE Computer Society Press.

Papazoglou, M. P. (2003). Web Services and Business Transactions. *World Wide Web: Internet and Web Information Systems*, 6(1):49–91.

Papazoglou, M. P. e Georgakopoulos, D. (2003). Service-Oriented Computing: Introduction. *Communications of the ACM*, 46(10):24–28.

Papazoglou, M. P. e Yang, J. (2002). Design Methodology for Web Services and Business Processes. In *TES '02: Proceedings of the Third Int. Workshop on Technologies for E-Services*, pp. 54–64, London, UK. Springer-Verlag.

Paula-Filho, W. P. (2003). *Engenharia de Software: Fundamentos, Métodos e Padrões*. Segunda Edição, Livros Técnicos e Científicos Editora.

Pierce, M.; Fox, G.; Youn, C.; Mock, S.; Mueller, K. e Balsoy, O. (2002). Interoperable web services for computational portals. In *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pp. 1–12, Los Alamitos, CA, USA. IEEE Computer Society Press.

Pressman, M. (2005). *Software Engineering: A Practitioner's Approach*. McGraw-Hill, N. Y. USA, 6 edição.

Ramamurthy, K. e Premkumar, G. (1995). Determinants and Outcomes of Electronic Data Interchange Diffusion. *IEEE Trans. on Engineering Management*, 42(4):332–351.

Randell, B. (1979). Software Engineering in 1968. In *ICSE '79: Proceedings of the 4th International Conference on Software Engineering*, pp. 1–10, Piscataway, NJ, USA. IEEE Press.

Reisig, W. (1985). *Petri Nets: an Introduction*. Springer-Verlag New York, Inc., New York, NY, USA.

Roser, S. e Bauer, B. (2005). A Categorization of Collaborative Business Process Modeling Techniques. In *CECW '05*, pp. 43–54, Washington, DC, USA. IEEE Computer Society.

RosettaNet (2008). [Online] Available: http://www.rosettanet.org.

Schmidt, M.-T.; Hutchison, B.; Lambros, P. e Phippen, R. (2005). The Enterprise Service Bus: Making Service-Oriented Architecture Real. *IBM Syst. Journal*, 44(4):781–797.

Schmit., B. e Dustdar, S. (2005). Model-Driven Development of Web Service Transactions. *Int. Journal of Enterprise Modeling and Information Systems Architecure*, 1(1).

Schulz, K. A. e Orlowska, M. E. (2004). Facilitating Cross-Organisational Workflows with a Workflow View Approach. *Data Knowl. Eng.*, 51(1):109–147.

SEI (2005). Capability Maturity Model Integration (CMMI) Overview. Technical report, Carnegie Mellon University, Software Engineering Institute, Pittsburgh.

SEI (2006). Capability Maturity Model Integration (CMMI). Technical report, Carnegie Mellon University, Software Engineering Institute, Pittsburgh.

Shaw, M. e Garlan, D. (1996). *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall Publishing.

Sivashanmugam, K.; Verma, K.; Sheth, A. e Miller, J. (2003). Adding Semantics to Web Services Standards.

Smith, H. e Fingar, P. (2006). *Business Process Management: The Third Wave*. Meghan-Kiffer.

SUN (2006). Java Enterprise Edition. [Online] Available: http://java.sun.com/javaee/.

Szyperski, C. (2002). *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Trastour, D.; Bartolini, C. e Preist, C. (2002). Semantic Web Support for the E-Commerce B2B Life Cycle. In *Proceedings of the 11th Int. World Wide Web Conference (WWW'02)*. ACM.

van der Aalst, W. M. P.; Barros, A. P.; ter Hofstede, A. H. M. e Kiepuszewski, B. (2000). Advanced Workflow Patterns. In *CooplS '02: Proceedings of the 7th International Conference on Cooperative Information Systems*, pp. 18–29, London, UK. Springer-Verlag.

van der Aalst, W. M. P.; Hofstede, A. e Weske, M. (2003). Business Process Management: A Survey. In *BPM'03: Proceedings of the 1st Int. Conference on Business Process Management*.

van Lamsweerde, A. (2000). Requirements Engineering in The Year 00: a Research Perspective. In *Proceedings of the 22nd Int. Conference on Software engineering*, pp. 5–19. ACM Press.

Vinoski, S. (2003). Integration with Web Services. *IEEE Internet Computing*, 7(6):75–77.

von Knethen, A. (2002). Change-Oriented Requirements Traceability: Support for Evolution of Embedded Systems. In *ICSM '02: Proceedings of the International Conference on Software Maintenance (ICSM'02)*, p. 482, Washington, DC, USA. IEEE Computer Society.

W3C (2001). Web Services Description Language - W3C Note. [Online] Available: http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

W3C (2002). Web Services Architecture: W3C Working Draft. [Online] Available: http://www.w3.org/TR/2002/WD-ws-arch-20021114/.

W3C (2006). SOAP specifications. [Online] Available: http://www.w3.org/TR/soap/.

Walls, C.; Richards, N. e Oberg, R. (2003). *XDoclet in Action*. Manning Publications Co., Greenwich, CT, USA.

Wohlin, C. et al. (2000). *Experimentation in Software Engineering: an Introduction*. Kluwer Academic Publishers, Norwell, MA, USA.

Xia, W. e Lee, G. (2005). Complexity of Information Systems Development Projects: Conceptualization and Measurement Development. *J. Manage. Inf. Syst.*, 22(1):45–83.

Zeng, L.; Flaxer, D.; Chang, H. e Jeng, J.-J. (2002). PLM Flow - Dynamic Business Process Composition and Execution by Rule Inference. In *TES '02: Proceedings of the Third Int. Workshop on Technologies for E-Services*, pp. 141–150, London, UK. Springer-Verlag.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo