Um Processo de Desenvolvimento de Aplicações Web baseado em Serviços

Fabio Zaupa[1], Itana M. S. Gimenes[1], Don Cowan[2], Paulo Alencar[2], Carlos Lucena[3]

[1]Departamento de Informática – Universidade Estadual de Maringá (UEM)
Maringá – PR – Brasil

[2]Computer System Group, The University of Waterloo
Waterloo-ON, Canadá

[3]Departamento de Informática, PUC-Rio
Rio de Janeiro – RJ - Brasil

zaupa@embusca.com.br, itana@din.uem.br, dcowan@csg.uwaterloo.ca, palencar@csg.uwaterloo.ca, lucena@inf.puc-rio.br

Abstract. *Web applications are currently widely disseminated. However, traditional development methods for these applications still require a lot of modeling and programming. They do not take much advantage of reuse. This paper presents an environment, called WIDE-PL, that supports the generation of Web applications based on the product line approach and SOA. In particular, this paper describes its application generation process, called ADESE. The evaluation of the process was carried out as a case study. The results show that the process offers several advantages such as it increases reuse and provides an explicit separation between the services and the business process.*

Resumo. *Apesar da disseminação de aplicações Web, o que se nota é que os métodos tradicionais de desenvolvimento de aplicações Web ainda requerem muita modelagem, programação e não tiram muito proveito de reutilização. Este artigo apresenta um ambiente, chamado WIDE-PL, para apoiar a geração de aplicações Web baseado em SOA seguindo os princípios de linha de produto. Em particular, este artigo descreve seu processo de desenvolvimento de aplicações, chamado ADESE. A avaliação do processo foi realizada por meio de um estudo de caso. Os resultados obtidos mostram que o processo oferece vantagens como o aumento da reutilização e uma explícita separação da lógica do negócio dos serviços.*

1. Introdução

A *Web* vem crescendo continuamente nos últimos anos. Muitas organizações, visualizando a possibilidade de realização de novos negócios, expandem suas atividades, criando sistemas de informação baseados em tecnologia *Web*. Esta tecnologia inclui *browsers*, servidores *Web*, linguagens de programação *Web* e protocolos de comunicação que viabilizam o funcionamento dos sistemas de informações através da *Web*. Neste contexto, os sistemas de informação são conhecidos como aplicações *Web*. O número de aplicações *Web* tem crescido exponencialmente, porém a qualidade das aplicações disponíveis não cresce proporcionalmente. Qualidade inclui confiabilidade, disponibilidade, eficiência, usabilidade, entre outros. Casati e Shan (2000) apontam que a fachada *Web* esconde enormes deficiências, operações manuais e sujeitas a erros, inflexíveis e complexas, e sistemas de difícil gerenciamento.

Aplicações *Web* podem ser autocontidas ou podem ser concebidas como uma cooperação de serviços que podem inclusive demandar cooperações inter-organizacionais. Um exemplo disso é um sistema de agência de viagem capaz de chamar serviços oferecidos por outras organizações, como o serviço de reserva de passagem, o serviço de reserva de hotel ou até o serviço de reserva de veículo.

A Arquitetura Orientada a Serviço (SOA[1]) é vista como capaz de oferecer a infra-estrutura tecnológica necessária para que uma aplicação possa ser definida por meio da composição de serviços eletrônicos. Dessa forma, oferece apoio à composição de aplicações distribuídas de uma forma flexível e com baixo custo. Em SOA, a composição de serviços é vista como um processo de negócio dividido em componentes reutilizáveis e interoperáveis.

Apesar da disseminação de aplicações *Web*, o que se percebe é que os métodos tradicionais de desenvolvimento aplicações *Web,* como OOHDM (Schwabe; Rossi, 1998) e OOWS (Pastor; Fons; Pelechano, 2003) ainda requerem muita modelagem, programação e não tiram muito proveito de reutilização. Dessa forma, o *Computer System Group* (CSG) da Universidade de Waterloo no Canadá propôs o WIDE (Cowan et al., 2004), que consiste em um grupo articulado de *frameworks* de apoio ao desenvolvimento de aplicações *Web*. O ambiente WIDE-PL é uma evolução de WIDE para desenvolvimento de aplicações *Web* que segue os princípios de linha de produto (Kang, 1990; Czarnecki et al., 2005). O objetivo de WIDE-PL é apoiar a geração de aplicações *Web* baseado em SOA. Em WIDE-PL, os serviços utilizados em uma aplicação *Web* podem ser especificados e compostos em uma linguagem de especificação e composição de serviços.

Este artigo apresenta o ambiente WIDE-PL e, em particular, um de seus elementos que é o seu processo de desenvolvimento de aplicações, o qual é chamado de ADESE. Este processo consiste de atividades para: (i) definir o domínio da aplicação; (ii) modelar serviços com base nos conceitos de modelo de características de linha de produto; (iii) instanciar o modelo de características; (iv) mapear o modelo instanciado para um correspondente diagrama de classes; (v) implementar os serviços a partir do diagrama de classes; e, (vi) gerar aplicações com base nos serviços definidos. ADESE utiliza WSDL para especificação de serviços e BPEL4WS para especificação de processos de negócio. A avaliação do processo ADESE foi realizada por meio de um estudo de caso. Os resultados obtidos mostram que o processo oferece vantagens em aspectos como: reutilização, separação explícita da lógica do negócio e dos serviços, redução de tempo e custo de desenvolvimento, interoperabilidade e manutenibilidade.

Este artigo está organizado da seguinte forma. A Seção 2 apresenta uma visão da arquitetura *Web* orientada a serviços. A Seção 3 apresenta conceitos básicos de linha de produto. A Seção 4 apresenta o ambiente WIDE-PL. Seção 5 apresenta o processo ADESE cuja avaliação é apresentada na seção 6. A Seção 7 apresenta as conclusões.

2. Arquitetura Orientada a Serviços (SOA)

A *Web* pode ser vista como uma coleção de serviços interconectados por protocolos de comunicação sobre a Internet. Segundo Papazoglou e Georgakopoulos (2003), SOA utiliza serviços eletrônicos como elementos fundamentais para o desenvolvimento de

[1] Do inglês Service Oriented Architecture.

aplicações distribuídas. Um serviço *Web* é um tipo específico de serviço eletrônico, e pode ser caracterizado como um componente aberto, auto-descritivo, com ampla padronização, grande interoperabilidade e acessíveis a outras aplicações ou serviços por meio de um endereço URL. A tecnologia de serviços *Web* tem como principais padrões XML (XML, 2006), WSDL (Christensen et al., 2001), SOAP (SOAP, 2006), HTTP (HTTP, 2006) e UDDI (UDDI, 2006). Em resumo, um serviço *Web* pode ser entendido como uma aplicação que tem uma interface descrita em WSDL, registrada em um diretório de serviços via protocolo UDDI, e interage com clientes por meio da troca de mensagens XML encapsuladas em envelopes SOAP, que são transportadas por protocolo HTTP (Fantinato, Toledo e Gimenes, 2005).

Processos de negócio são utilizados para compor serviços *Web*. Um processo de negócio é uma seqüência de atividades executadas, possivelmente, por múltiplas organizações que atuam cooperativamente para atingir um objetivo de negócio comum. Existem algumas linguagens para especificação de processos de negócio baseados em serviços, dentre as quais destaca-se BPEL4WS (*Business Process Execution Language for Web Services*) a qual chamamos, neste artigo, simplesmente de BPEL. BPEL permite especificar um processo de negócio descrevendo o relacionamento entre os serviços participantes, manipulação de variáveis, fluxo de dados e ações para fluxo de exceções, sendo executados por um motor que age como intermediário na troca de mensagens (BEA Systems et al., 2003).

3. A Abordagem de Linha de Produto

Uma linha de produto de software consiste de um conjunto de sistemas de software, também chamado de família de aplicações, que compartilham um conjunto gerenciável de características de um seguimento particular de mercado ou missão (SEI, 2006) (Clements; Northrop, 2001). As aplicações (ou produtos) da família, chamadas de membros, são desenvolvidas a partir de um conjunto comum de artefatos (*assets*) que constituem a arquitetura da linha de produto. Esta arquitetura deve ser genérica o suficiente para representar todos os membros da linha de produto. Um membro da família é gerado adaptando-se os requisitos da linha de produto para atender às suas necessidades. Eventualmente, novos componentes podem ser desenvolvidos e adicionados à linha de produto, alimentando assim sua constante evolução.

Uma forma comum de representação do domínio de uma linha de produto é o modelo de características[2]. Este modelo permite a representação das capacidades comuns e variáveis de uma família de produtos de um determinado domínio. O conceito de características vem da engenharia de domínio (Kang, 1990) e tem evoluído para atender as demandas de linha de produto (Sochos et al., 2004; Czarnecki et al., 2005; van Gurp et al., 2001). Características são usualmente representadas por meio de diagramas de características que são estruturas em forma de árvore com anotações que indicam os tipos de características de uma família de produtos (Sochos et al., 2004; Czarnecki et al., 2005).

No contexto deste artigo, conceitos de linha de produto são aplicados para apoiar a modelagem dos serviços de domínio do ambiente WIDE-PL que comporão as aplicações *Web* geradas pelo ambiente. Um serviço de domínio é representado por características configuráveis.

[2] Do inglês features.

4. O Ambiente WIDE-PL

O *Computer System Group* (CSG) da Universidade de Waterloo tem desenvolvido tecnologias para portais *Web* desde o início dos anos 90 (Cowan et al., 2004). Como resultado, uma série de portais chamados Espaço Público de Aprendizagem foram construídos. O ambiente usado para apoiar o desenvolvimento dos portais é chamado WIDE (*Waterloo Informatics Development Environment*). WIDE possui um grupo articulado de *frameworks* baseados em XML de apoio ao desenvolvimento de aplicações *Web*, tais como serviços de mapas, serviços de diagramas, serviços de relatórios, serviços de gerenciamento, serviços de controle de acesso, serviços de pesquisa em base de dados, serviços de notificação e serviços de agente. WIDE-PL (*WIDE - Product Line*) é uma evolução do WIDE que tem como objetivo gerar aplicações *Web* baseadas em SOA seguindo os princípios da LP (Kang, 1990; Czarnecki et al., 2005). Em WIDE-PL, os serviços utilizados em uma aplicação *Web* podem ser especificados e compostos em uma linguagem de especificação e composição de serviços. WIDE-PL é composto de serviços de domínio que têm um grupo de dados e metadados associados. Os serviços são coordenados pelo processo de negócio da aplicação. WIDE-PL utiliza um princípio similar ao ambiente proposto por Ceri (2003), porém tem por objetivo gerar aplicações a partir de serviços e de um processo de negócio explícito.

Neste trabalho, utilizou-se a tecnologia de serviços *Web* para especificação e composição de serviços, devido à atual disponibilidade de padrões e de infra-estrutura de apoio que permitem a especificação de serviços em WSDL (Christensen et al., 2001) e especificação de processos de negócio em BPEL (BEA Systems et al., 2003). A vantagem de se utilizar uma linguagem como BPEL para compor os serviços de aplicações que são locais, distribuídos ou baseados em padrões da Internet, é obter uma base tecnológica que minimize a complexidade de usar tecnologias heterogêneas. No entanto, linguagens específicas para WIDE-PL podem ser projetadas no futuro.

Em WIDE-PL, uma aplicação *Web* é representada por um conjunto de características de alto nível (Kang, 1990; Czarnecki et al., 2005). Essas características são abstrações que representam as capacidades de aplicações *Web*. Cada característica de alto nível de uma aplicação é realizada por um serviço de domínio. Cada serviço de domínio abrange as operações e os dados necessários para realizar uma característica da aplicação (Kang, 2002).

A Figura 1A ilustra o modelo de características de alto nível das aplicações *Web* em WIDE-PL, produzido pela ferramenta *Feature Plug-in* (Antkiewicz; Czarnecki, 2004). Os demais modelos de características deste artigo também seguem essa notação. Uma aplicação *Web* em WIDE-PL tem ao menos um serviço de base de dados, um serviço de interação, um serviço de interface e um serviço de processo de negócio. Assim, essas características são representadas como obrigatórias. Serviços opcionais de domínio podem ser adicionados a WIDE-PL de acordo com a evolução do ambiente e as necessidades das aplicações. Exemplos de serviços que podem ser adicionados, baseados na estrutura atualmente disponível no WIDE são: interface interativa, multimídia e agente. As características de um serviço de domínio podem ser refinadas de tal maneira que possibilite aos clientes da aplicação configurar os serviços de acordo com suas necessidades. A Figura 1B apresenta um exemplo do refinamento do modelo de características para o serviço de interface interativa. Este serviço permite a definição

e a operação de diversos tipos de exposição interativa tais como mapas, gráficos e diagrama. A característica de mapa também é refinada, e apresenta as características de desenho, contêiner e resolução como obrigatórias, pois elas são necessárias em qualquer mapa, enquanto as características de interação, mensagem e pesquisa são opcionais, podendo ser inseridas no mapa de acordo com a necessidade da aplicação.

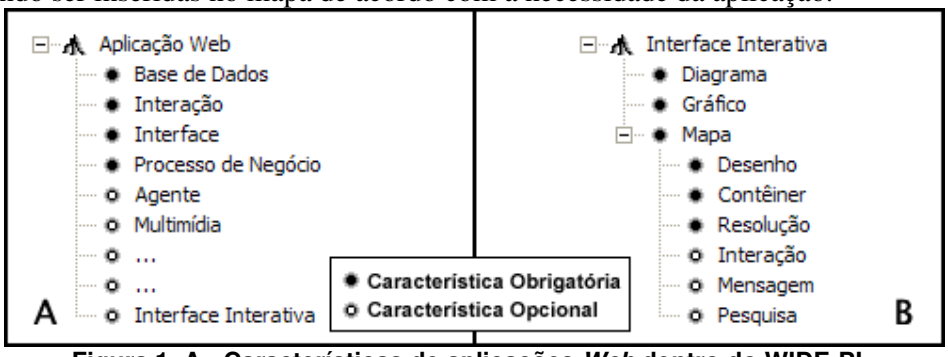

Figura 1. A - Características de aplicações *Web* dentro do WIDE-PL, B - Modelo de características do serviço de interface interativa.

Nas seções seguintes são apresentados os serviços obrigatórios e a arquitetura lógica de uma aplicação gerada em WIDE-PL.

4.1 Serviços Obrigatórios

Os serviços obrigatórios de WIDE-PL são essenciais para o funcionamento de uma aplicação *Web*. São eles:

- Serviço de base de dados: responsável por qualquer interação entre o processo de negócio e a base de dados da aplicação. Com este serviço é possível criar e atualizar a estrutura da base de dados da aplicação, além de manipular e recuperar seus respectivos dados.

- Serviço de interação: responsável por controlar as interações entre o serviço de processo de negócio e os usuários da aplicação. Deve receber um *script* de página gerado no serviço de interface e exibi-lo ao usuário por meio de seu *browser*. Ao receber requisições do usuário, deve tratar e repassar ao serviço de processo de negócio para o devido processamento.

- Serviço de interface: responsável por receber os dados e formatar a página que será exibida ao usuário. Neste serviço é definido o estilo da aplicação, escolhendo a fonte de letra e as cores que serão usadas nas páginas. Também é possível definir o conteúdo da página, informando o que deve ser apresentado no cabeçalho, rodapé, menu e corpo da página. Em relação ao corpo da página, o serviço está preparado para receber, tratar e exibir no local as listas de itens e as solicitações de formulários vindas do serviço de processo de negócio. Depois de inseridas todas as configurações, um *script* referente à página pode ser gerado.

- Serviço de processo de negócio: coordena a execução de uma aplicação, definindo o comportamento da aplicação. Contém as regras de negócio, a seqüência de invocação dos serviços participantes, as ações de controle de exceções, a utilização de funções e variáveis. Somente este serviço pode interagir com os demais serviços utilizados pela aplicação, assim qualquer requisição ou resposta a um serviço deve passar por ele.

4.2 Arquitetura Lógica

A arquitetura lógica de aplicações geradas por WIDE-PL visa deixar explícita a separação entre o processo de negócio que controla a aplicação e seus serviços. Dessa forma, a aplicação é adaptável uma vez que o processo de negócio pode ser alterado sem necessariamente modificar os serviços, assim como serviços podem ser adicionados e modificados causando impacto mínimo no processo de negócio. Esta arquitetura, conforme mostra a Figura 2, é composta de três camadas:

- Coordenação: responsável por fazer a coordenação do processo de negócio da aplicação. Contém o serviço de interação e o serviço de processo de negócio.
- Aplicação: contém uma configuração dos serviços de domínio, obrigatórios ou opcionais, selecionados para executar uma aplicação. Os serviços de domínio são invocados pelo serviço do processo do negócio.
- Repositório de dados: contém os dados e os metadados que podem ser usados pelos serviços da camada de aplicação.

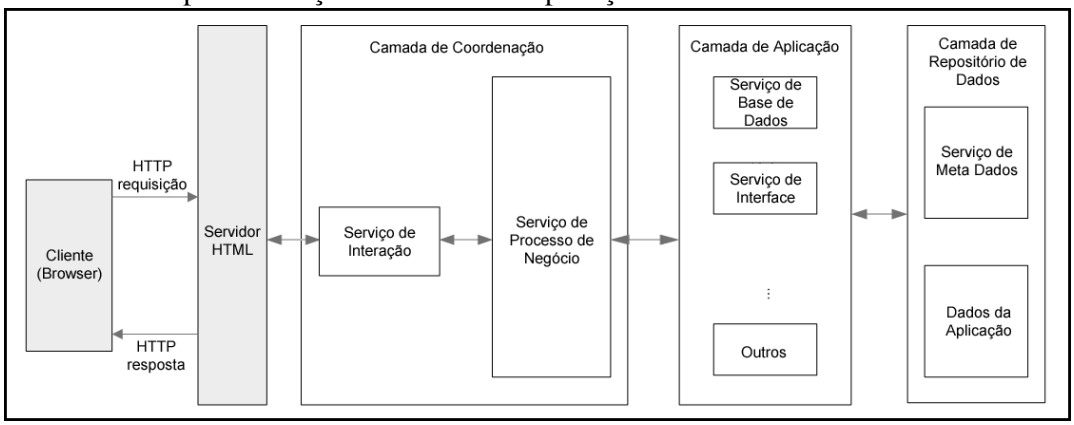

Figura 2. Visão geral da arquitetura lógica do WIDE-PL.

A seqüência para execução de uma aplicação de acordo com a arquitetura lógica de WIDE-PL é: (i) a aplicação é iniciada na camada de coordenação ativando o serviço de processo de negócio; (ii) o serviço de processo de negócio faz, de acordo com o fluxo da especificação da aplicação, solicitações a outros serviços contidos na camada de aplicação; (iii) os serviços contidos na camada de aplicação podem acessar a camada de repositório de dados na busca por dados e metadados referentes à aplicação, como é o caso do serviço de base de dados; (iv) após as solicitações para carregar a página, o serviço de processo de negócio solicita ao serviço de interface que gere o *script* de página a ser exibido ao usuário. Depois de gerado, o *script* é repassado do serviço de processo de negócio ao serviço de interação, que o exibe ao usuário por meio de seu *browser*; e (v) o usuário pode fazer as interações pertinentes e enviar as requisições novamente ao serviço de interação, que as tratam e repassam ao serviço de processo de negócio para continuar o fluxo da especificação do processo de negócio da aplicação, consistindo no retorno ao passo (ii). Os passos descritos ocorrem repetidas vezes até o encerramento do serviço de processo do negócio, quando a aplicação é encerrada.

No ambiente WIDE-PL, o serviço de processo do negócio é o único ponto de controle e responsável por coordenar a execução da aplicação, assim o seu o gerenciamento é por orquestração (Peltz, 2003). Dessa forma, o serviço de processo de negócio contém o motor de execução e os outros serviços utilizados na aplicação

possuem apenas uma infra-estrutura básica capaz de executar suas partes no processo e comunicar-se com o processo de negócio.

5. O Processo ADESE

A Figura 3 apresenta o processo ADESE representado graficamente por um diagrama de blocos SADT (Ross; Schoman, 1977) que contém as três etapas do processo e as informações manipuladas por ele. Essas etapas são descritas a seguir.

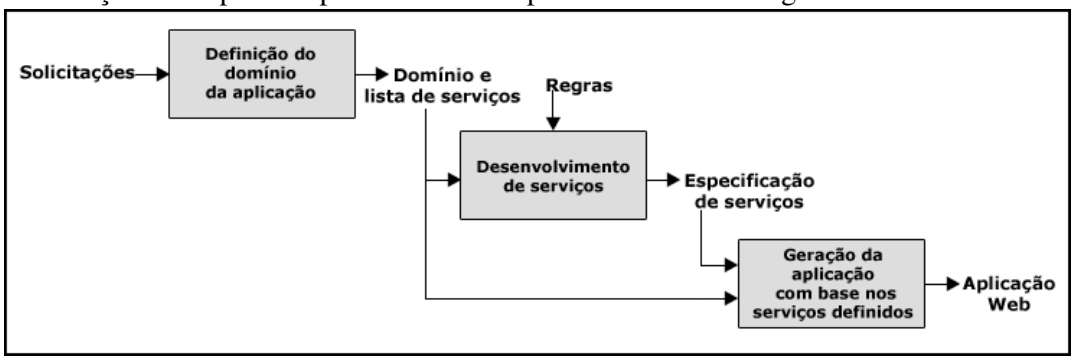

Figura 3. Representação gráfica do processo ADESE.

Etapa 1 - Definição do domínio da aplicação

Nesta etapa é definido o domínio da aplicação por meio de uma descrição textual simples. Esta descrição é tomada como base para realizar a análise de domínio que consiste em gerar os modelos de características que determinam a lista dos respectivos serviços obrigatórios e opcionais necessários para o desenvolvimento da aplicação *Web*.

Etapa 2 - Desenvolvimento de serviços

Esta etapa deve ser executada cada vez que um novo serviço é necessário em uma aplicação *Web*. Caso os serviços necessários já estejam disponíveis em WIDE-PL deve-se passar a etapa seguinte. Para o desenvolvimento de cada novo serviço é necessário:

- Definir o modelo de características do serviço: consiste definir o modelo por meio da captura de seus atributos e variabilidades.
- Instanciar o modelo de características: consiste em selecionar uma configuração do modelo contendo as características obrigatórias e opcionais a serem utilizadas na aplicação.
- Mapear o modelo de características para o diagrama de classes UML: consiste em converter o modelo de características em um correspondente diagrama de classes identificando os elementos que compõem o diagrama como classes, relacionamentos, atributos e operações.
- Implementar o serviço a partir do diagrama de classes: consiste nos seguintes passos: (i) programar as classes encontradas no diagrama de classes; (ii) converter as classes em um serviço; e, (iii) especificar o serviço gerado em uma interface padronizada.

O mapeamento do modelo de características para o diagrama de classes visa converter um modelo que resume as capacidades comuns e diferentes de uma aplicação, para um diagrama que representa a sua estrutura física. As regras definidas neste processo para conversão do modelo de características para diagrama de classes são:

- Identificar classes: criar uma classe para cada característica não "folha" do modelo de características.

- Identificar relacionamentos: adicionar o relacionamento de agregação entre as classes que tem relacionamento no modelo de características, sendo que a classe agregadora é a classe que, no relacionamento, tem nível mais alto no modelo de características.

- Identificar operações e atributos: transformar as características "folhas" do modelo de características em operações ou atributos das classes já criadas, sendo que cada característica "folha" que será transformada em operação ou atributo deverá ser incluída na classe que representa sua respectiva característica de nível mais alto no modelo de características.

- Identificar operações da classe principal: a característica raiz do modelo de características é a classe principal do diagrama de classes e as operações incluídas nesta classe serão as operações convertidas em operações do serviço. Assim, caso alguma outra classe do diagrama contenha operações que devam ser operações do serviço, estas operações deverão ser incluídas na classe principal, e na implementação, as operações contidas na classe principal deverão fazer a invocação da operação original nas outras classes.

- Incluir operações e atributos auxiliares: operações e atributos adicionais como os casos das operações *get* e *set* podem ser incluídos no diagrama para auxiliarem posteriormente na execução de alguma operação do serviço.

Etapa 3 - Geração da aplicação com base nos serviços definidos

Esta etapa visa gerar uma aplicação *Web* baseada em serviços de acordo com o domínio escolhido na Etapa 1. As atividades que compõe esta etapa são:

- Criação de um processo de negócio síncrono: consiste em criar a estrutura básica do processo de negócio da aplicação. Esta estrutura básica deve conter informações como as definições iniciais do processo, variáveis iniciais, serviços participantes iniciais e uma definição vazia do fluxo do processo. Como se trata de um processo síncrono, este será iniciado pela execução de uma operação contida no serviço de processo de negócio e encerrado após a execução de todo seu fluxo, retornando alguma informação à operação do serviço de processo de negócio que o iniciou.

- Adição de serviços: consiste em adicionar à especificação do processo de negócio da aplicação, os serviços necessários para a geração da aplicação. Os passos para adicionar os serviços são: (i) importar para a ferramenta de desenvolvimento de escolha do desenvolvedor a especificação de cada um dos serviços necessários para geração da aplicação; (ii) configurar a especificação de cada serviço necessários para a geração da aplicação incluindo uma extensão da linguagem, que são códigos da linguagem usados apenas em casos específicos, com a finalidade de permitir a comunicação entre o processo de negócio e os seus respectivos serviços; e, (iii) adicionar os serviços já configurados ao processo de negócio.

- Especificação do processo de negócio: consiste em complementar a especificação do processo de negócio com o fluxo da aplicação de acordo com

as regras da linguagem utilizada, as regras de negócio do domínio da aplicação e as regras para execução, que estabelecem a seqüência em que os serviços devem executados, de acordo com a arquitetura lógica do WIDE-PL.

6. Avaliação do Processo ADESE

Esta seção apresenta a avaliação do processo ADESE por meio de um estudo de caso, cujo desenvolvimento completo pode ser encontrado em Zaupa (2007). Este estudo consistiu de duas aplicações *Web* independentes provenientes de diferentes domínios, sendo uma do domínio de controle de estoques e outra do domínio de locação de vídeos. As aplicações concebidas utilizam os mesmos serviços obrigatórios, porém seus respectivos serviços de processo de negócio são compostos de acordo com as regras específicas de cada domínio. Além disso, as aplicações possuem serviços opcionais que são necessários a cada domínio. As ferramentas de apoio utilizadas no desenvolvimento do estudo de caso foram: (i) Netbeans IDE 5.5 (Netbeans, 2006) com a função de programar, compilar e executar os serviços e os processos de negócio; (ii) *Feature Plug-in* com a função de definir o modelo de características; e, (iii) o banco de dados MySQL com a função de armazenar e buscar dados usados nas aplicações.

A seguir serão apresentadas as etapas do processo ADESE para o desenvolvimento da aplicação de locação de vídeo, bem como comentários sobre as partes que seriam comuns para o desenvolvimento da aplicação de controle de estoque.

Etapa 1 - Definição do Domínio da Aplicação

A aplicação deve controlar a locação de vídeos, assim deve possuir cadastros de clientes, filmes, fitas, controle de locações, reservas e pagamentos, além de controle de acesso dos usuários da aplicação. Para o desenvolvimento desta aplicação serão necessários os serviços obrigatórios já disponíveis em WIDE-PL e um serviço opcional de calendário que será responsável por informar datas e horários à aplicação, como a data do dia, o horário do momento, diferença de dias entre datas, calcular uma data a partir da soma de dias, entre outras.

Etapa 2 - Desenvolvimento de Serviços

Nesta etapa é apresentado o desenvolvimento do serviço opcional de calendário para ilustrar o desenvolvimento de um serviço não disponível em WIDE-PL. Para tal, é necessário definir o modelo de característica, instancia-lo, mapeá-lo para o diagrama de classes correspondente e posteriormente implementa-lo. A Figura 4A apresenta o modelo de características obtido para o serviço de calendário. Este modelo contém a característica `Calendário` como característica raiz, e possui o restante das características divididas em dois grupos maiores chamados `Data` e `Horário`, sendo que as características `Hoje`, `Dia`, `Mês` e `Ano`, contida no grupo `Data`, e `Agora`, `Hora` e `Minuto`, contida no grupo `Horário`, são características obrigatórias. Em seguida, o modelo de característica do serviço de calendário foi instanciado definindo-se as características opcionais a serem utilizadas.

O próximo passo consiste do mapeamento do modelo de características instanciado para um diagrama de classes. A Figura 4B apresenta o diagrama de classes produzido seguindo as regras de conversão descritas na seção 5 (Etapa 2).

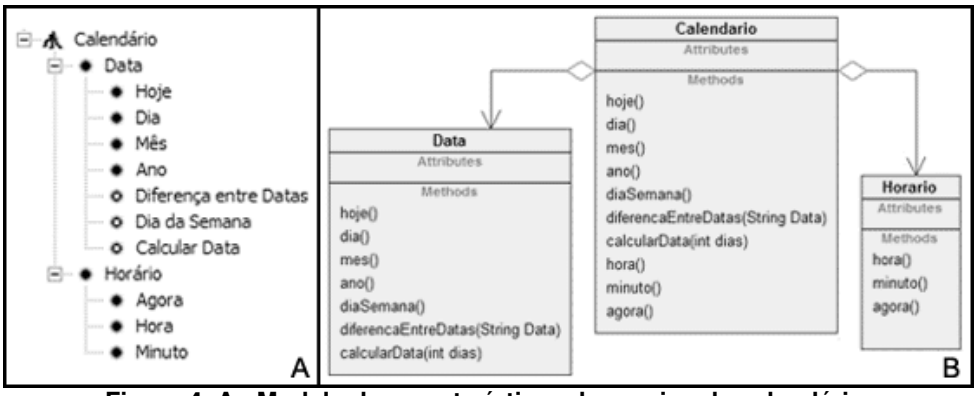

Figura 4. A - Modelo de características do serviço de calendário, B – Respectivo diagrama de classes.

Obtido o diagrama de classes, deve-se implementar o serviço. Para tal, é preciso programar e compilar as classes que compõem o diagrama. Esta programação é feita manualmente pelo desenvolvedor com auxílio de alguma ferramenta de desenvolvimento, como neste estudo de caso, a ferramenta NetBeans (2006). Em seguida, deve-se converter as classes em serviço. A conversão de classes em serviços também depende da ferramenta utilizada. A Figura 5 apresenta parte da estrutura criada pelo NetBeans para o serviço de calendário. A pasta `Serviços Web` contém o serviço, denominado `wsCalendario`, com suas respectivas operações. Estas são as mesmas operações que estão na classe `Calendário`. Após o desenvolvimento do serviço, é necessário especificar sua interface. Conforme já descrito, esta especificação é feita em WSDL. Algumas ferramentas, como é o caso da ferramenta NetBeans, criam automaticamente o documento WSDL correspondente.

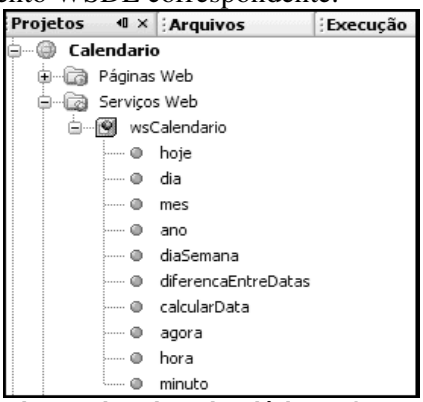

Figura 5 - Estrutura do serviço de calendário na ferramenta Netbeans 5.5.

Etapa 3 - Geração da Aplicação com Base nos Serviço Definidos

Nesta etapa são apresentadas as atividades realizadas e alguns trechos da especificação em BPEL. A primeira atividade é a criação de um processo de negócio síncrono. O processo criado representa o serviço obrigatório de processo de negócio, contendo uma única operação, que quando executada, iniciará a execução do processo de negócio da aplicação de locação de vídeos. A estrutura básica do novo processo contém descrições em BPEL como as definições iniciais do processo, variáveis iniciais, serviços participantes iniciais e a uma definição vazia do fluxo do processo.

Após a criação do processo, é preciso adicionar os serviços necessários. Para tal, deve-se importar sua especificação para a ferramenta de desenvolvimento para que

possam ser configurados. Esta configuração consiste em incluir em cada especificação de serviço, uma extensão da linguagem, com a finalidade de estabelecer a comunicação entre o processo de negócio e o respectivo serviço. O passo final é adicionar os serviços já configurados ao processo. A Listagem 1 apresenta a descrição em BPEL em que aparecem os serviços participantes. Cada linha entre os elementos `<partnerLinks>` e `</partnerLinks>` representa um serviço, caracterizado por um nome e uma referência a sua especificação em WSDL que identificam as suas operações.

```
<partnerLinks>
  <partnerLink name="Calendario" partnerLinkType="ns5:wsCalendarioSEILinkType" partnerRole="wsCalendarioSEIRole"/>
  <partnerLink name="Database" partnerLinkType="ns4:wsDatabaseSEILinkType" partnerRole="wsDatabaseSEIRole"/>
  <partnerLink name="Interaction" partnerLinkType="ns3:wsInteractionSEILinkType" partnerRole="wsInteractionSEIRole"/>
  <partnerLink name="Interface" partnerLinkType="ns2:wsInterfaceSEILinkType" partnerRole="wsInterfaceSEIRole"/>
  <partnerLink name="BusinessProcess" partnerLinkType="ns1:partnerlinktype1" myRole="partnerlinktyperole1"/>
</partnerLinks>
```

Listagem 1. Descrição BPEL dos serviços incluídos no processo de negócio.

Para fazer a especificação do processo de negócio, é preciso inicialmente declarar as variáveis e atribuir valores aos elementos usados durante toda a execução da aplicação, como os valores de conexão com a base de dados, cabeçalho, rodapé, menu e estilo da página. A Listagem 2 apresenta um trecho BPEL em que são atribuídos valores respectivamente às variáveis de `login`, `password`, `localhost` e `drive`, e na seqüência é invocada a operação `connect` do serviço de base de dados enviando as variáveis como parâmetro para validar a conexão com a base de dados.

```
<assign name="Assign1">
        <copy><from>string('')</from><to>$ConnectIn1.parameters/String_1</to></copy>
        <copy><from>string('')</from><to>$ConnectIn1.parameters/String_2</to></copy>
        <copy><from>string('jdbc:mysql://localhost/mestrado')</from><to>$ConnectIn1.parameters/String_3</to></copy>
        <copy><from>string('com.mysql.jdbc.Driver')</from><to>$ConnectIn1.parameters/String_4</to></copy>
 </assign>  <invoke name="ConnectDB" partnerLink="Database" operation="connect" portType="ns2:wsDatabaseSEI"
inputVariable="ConnectIn1" outputVariable="ConnectOut1"/>
```

Listagem 2. Trecho BPEL que invoca a operação connect do serviço de base de dados.

O corpo da página deve ser sempre atualizado com informações pertinentes às interações com o usuário. Para isso é necessário enviar as informações que deverão ser mostradas no corpo da página ao serviço de interface. Após o envio dessas informações ao serviço, a operação `format`, que faz a formatação e gera o *script* de página deve ser invocada. O *script* gerado retorna ao processo de negócio que o encaminha ao serviço de interação para ser exibido ao usuário. A Listagem 3 apresenta a situação descrita.

```
<invoke name="Page1" partnerLink="Interface" operation="format" portType="ns2:wsInterfaceSEI"
inputVariable="FormatFalse" outputVariable="Page"/>
<assign name="Ass3"><copy><from>$Page.result/result</from><to>$ResultInteraction.result/result</to></copy></assign>
<invoke name="Interaction1" partnerLink="Interaction" operation="requestResponse" portType="ns3:wsInteractionSEI"
inputVariable="PageInteraction" outputVariable="ResultInteraction"/>
```

Listagem 3. Trecho BPEL que formata a página e envia ao serviço de interação.

Após a interação com o usuário, as requisições serão tratadas e repassadas ao processo de negócio para continuar o seu fluxo. Para cada uma das funções da aplicação como cadastrar, locar, efetuar pagamento e reservar filmes, os procedimentos são semelhantes aos citados. Para casos em que é necessário exibir uma lista de dados que se encontra na base de dados, a única diferença é recuperar os dados antes de exibi-los através do serviço de base de dados, e repassá-lo ao serviço de interface. Para finalizar o processo de negócio e consequentemente a aplicação, é necessário selecionar a opção correta no menu da aplicação para levar ao fechamento dos laços.

A especificação do processo de negócio da aplicação de controle de estoque segue os mesmos procedimentos descritos para a aplicação de locação de vídeos. A diferença entre a aplicação de locação de vídeos, a aplicação controle de estoque e qualquer outro sistema de informação que venha a ser desenvolvida seguindo o processo ADESE está nas regras de negócio da aplicação. Dessa forma, para cada aplicação devem ser definidas as configurações do processo como cores e fontes, a base de dados que será usada, as informações que serão exibidas no cabeçalho, rodapé, menu e principalmente no corpo da página. A estrutura da linguagem de especificação e as regras para execução da arquitetura lógica de WIDE-PL não mudam de uma aplicação para outra, o que torna a concepção de aplicações *Web* muito flexível.

Finalmente, é necessário um motor BPEL para executar as especificações dos processos de negócio das aplicações. Embora, alguns motores BPEL tenham sido testados no contexto deste trabalho como: o motor embutido no Netbeans 5.5 (Netbeans, 2006), Bexee (Bexee, 2007) e PXE (PXE, 2007), nenhum se mostrou adequado, ou por dificuldades de acesso, ou instalação ou disponibilidade. Assim, optou-se por simular a execução dos processos de negócio em um *servlet* Java, sendo apenas um *servlet* por aplicação, e nele invocar as operações do serviço na mesma seqüência em que são invocados no processo de negócio descrito em BPEL. Os *servlets* das aplicações são ativados pela chamada de seu endereço no *browser*. Soluções mais recentes permitem inclusive a conversão automática de BPEL para Java (B2J, 2007).

6.1 Resultados

Após a realização do estudo de caso foi feita uma análise qualitativa. Esta análise foi realizada comparando-se alguns aspectos do processo ADESE com os métodos tradicionais de desenvolvimento de aplicação *Web*. Os aspectos analisados mostram o potencial da abordagem como segue:

- Reutilização: há indicação de ganho do processo ADESE porque os serviços, uma vez desenvolvidos para uso em uma aplicação *Web*, ficam disponíveis no ambiente WIDE-PL para uso posterior em outras aplicações.

- Separação explícita da lógica do negócio e dos serviços: um potencial de ganho pode ser evidenciado porque a abordagem separa completamente a lógica do negócio, que fica no serviço de processo de negócio, das funções necessárias para o desenvolvimento da aplicação que estão embutidas nos serviços e são invocadas quando necessário pelo processo de negócio.

- Redução de tempo e custo de desenvolvimento: há indicação de ganho porque o maior esforço de desenvolvimento passa a ser no desenvolvimento do processo de negócio da aplicação. Os serviços necessários para compor a aplicação são adicionados ao processo para serem invocados quando for preciso, poupando muita modelagem e programação das funções que constam nesses serviços, reduzindo assim tempo e consequentemente custo.

Entre alguns requisitos não funcionais também se pode citar indicações de ganhos e perdas com o uso do processo ADESE em relação aos métodos tradicionais. Exemplos de ganho são: interoperabilidade porque permite que serviços produzidos em diferentes linguagens e plataformas se comuniquem entre si; e, manutenibilidade porque a aplicação é composta por serviços independentes e com poucas e bem definidas dependências entre si facilitando a manutenção. Exemplos de perda são: desempenho

porque os serviços utilizam XML como formato padrão para a troca de mensagens, e estes são bem maiores que o equivalente em formato binário; e segurança porque aumenta o risco de interceptação dos dados, já que serviços trocam dados pela *Web*. Pode ocorre perda em: disponibilidade porque a aplicação normalmente utilizará serviços que estão distribuídos em vários servidores, e isso aumenta o risco que em algum momento algum desses serviços não esteja disponível.

7. Conclusões

Este artigo apresenta o processo ADESE para desenvolvimento de aplicações *Web* baseado em serviços no contexto de WIDE-PL. Como contribuições, destacam-se a definição e especificação do processo, que inclui em sua etapa de desenvolvimento de serviços, regras de mapeamento do modelo de características para o diagrama de classes. Na literatura, ainda há uma carência de abordagens de mapeamento do modelo de características em outros modelos. Um exemplo é a abordagem proposta por Czarnecki e Antkiewicz (2005), porém o foco é no mapeamento do modelo de características para diagrama de atividades UML. Os resultados obtidos mostram vantagens em relação aos métodos tradicionais. A principal delas é a separação explícita da lógica do negócio e dos serviços. Além disso, há indicações de vantagens em aspectos como: reutilização; redução de tempo e custo de desenvolvimento; interoperabilidade; e, manutenibilidade. O processo ADESE mostra que é possível um desenvolvimento mais automatizado de aplicações *Web* baseado em serviços, e assim abre caminho para uma nova abordagem. Nesta abordagem o foco é no processo de negócio, e serviços são adicionados e suas operações são invocadas de acordo com as regras de negócio do domínio da aplicação. Algumas desvantagens foram observadas como o atual estágio da tecnologia em que ainda falta um motor BPEL eficiente para executar os processos de negócio das aplicações, bem como em requisitos como desempenho, segurança e disponibilidade. Porém, essas desvantagens podem ser superadas com o avanço da tecnologia.

Referências

Antkiewicz, M.; Czarnacki, K. (2004), **FeaturePlugin: Feature Modeling Plug-in for Eclipse,** In Eclipse '04: Proceedings of the OOPSLA, Canada, ACM Press.

B2J (2007), **BPEL to Java Subproject Eclipse**, http://www.eclipse.org/stp/b2j, Março.

BEA Systems, IBM, Microsoft, SAP AG, Siebel Systems (2003), **Business Process Execution Language for Web Services Version 1.1**.

Bexee (2007), **Bexee BPEL Execution Engine,** http://sourceforge.net/projects/bexee, Janeiro.

Casati, F., Shan, M-C (2000), **Process Automation as the Foundation for E-Business,** In: Proceedings of 26th International Conference on Very Large Databases, Egito.

Ceri, S. et al. (2003), **Architectural Issues and Solutions in the Development of Data-Intensive Web Applications**, Proc. of CIDR'03, USA.

Christensen, E., Curbera, F., Meredith, Weerawarana, Sanjiva (2001), **Web Services Description Language (WSDL)** 1.1, W3C Note 15.

Clements, P., Northrop, L. (2001), **Software product lines: practices and patterns,** 1ed, Boston: Addison-Wesley, p. 608.

Cowan, D., Fenton, S., Mulholland, D. (2004), **The Waterloo Informatics Development Environment (WIDE)**. CSG Internal Note.

Czarnecki, K., Helzen, S.; Eisenecker, U. (2005), **Staged configuration through specialization and multi-level configuration of feature models. To appear in special issue on "Software Variability: Process and Management"**, Software Process Improvement and Practice, 10(2).

Czarnecki, K., Antkiewicz, M. (2005), **Mapping features to models: A template approach based on superimposed variants,** GPCE 2005, v. 3676, p. 422, Springer.

Fantinato, M., Toledo, M., Gimenes, I (2005), **Arquitetura de Sistemas de Gerenciamento de Processos de Negócio Baseado em Serviços,** Rel.T., Unicamp.

Gimenes, I. et al (2005), **O projeto preliminar de WIDE-PL**, Relato de estágio de pós-doutorado na Universidade de Waterloo, Canadá, CAPES-MEC, Brazil.

Kang, K. (1990), **Feature-oriented domain analysis (FODA) - feasibility study**, Technical Report CMU/SEI-90-TR-21, SEI/CMU, Pittsburgh.

Kang, K., Lee, J., Donohoe, P. (2002), **Feature-oriented Product Line Engineering**, IEEE Software.

Netbeans (2006), **Netbeans IDE 5.5**, http://www.netbeans.org, Abril.

Papazoglou, M., Georgakopoulos, D. (2003), **Service-oriented computing**, Communications of the ACM: Service-Oriented Computing.

Pastor, O., Fons, J., Pelechano, V. (2003), **OOWS**, Department of Information Systems and Computation Technical University of Valencia.

Peltz, C. (2003), **Web Services Orchestration and Choreography**, HP Company.

PXE (2007), **Process Execution Engine,** http://sourceforge.net/projects/pxe, Janeiro.

Ross, D., Schoman, K. (1977), **Structured Analysis for Requirements Definition**, IEEE Transactions on Software Engineering 3(1).

Schwabe, D; Rossi, G (1998), **Developing Hypermedia Applications using OOHDM**, Hypermedia Development Processes, Methods and Models, Hypertext, USA.

SEI (2006), Software Engineering Institute, **A framework for software product line practice 4.2,** http://www.sei.cmu.edu/productlines/framework.html, Abril.

Sochos, P; Philipow, I; Riebish, M (2004) **Feature-oriented development of software product lines: mapping feature models to the architecture.** Springer, p.138.

SOAP (2006), **Simple Object Access Protocol,** http://www.w3.org/TR/SOAP, Abril.

UDDI (2006), **Universal, Description, Discovery and Integration,** http://www.uddi.org, Abril.

van Gurp, J., Bosch, J., Svahnberg, M.(2001), **On the notion of variability in software product lines**, in: Proc. The Working IEEE/IFIP, WICSA, The Netherlands.

XML (2006), **Extensible Markup Language,** http://www.w3.org/TR/REC-xml, Maio.

Zaupa, F. (2007), **Um Processo de Desenvolvimento de Aplicações Web baseado em Serviços**, Dissertação de Mestrado em Ciência da Computação, UEM, Maringá.