CARLOS HENRIQUE ESTEVES MENDONÇA
CRISTINA NAOMI MAKIBARA
FÁBIO HIDEKI ARAGAKI

SERVIDOR DE AUTORIZAÇÃO DE EVENTOS ASSÍNCRONOS PARA APLICAÇÕES DE MOBILIDADE

SÃO PAULO
2006

CARLOS HENRIQUE ESTEVES MENDONÇA
CRISTINA NAOMI MAKIBARA
FÁBIO HIDEKI ARAGAKI

SERVIDOR DE AUTORIZAÇÃO DE EVENTOS ASSÍNCRONOS PARA APLICAÇÕES DE MOBILIDADE

PROJETO DE FORMATURA DO CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA, ÊNFASE COMPUTAÇÃO, DA ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO.

ORIENTADOR: PROF. DR. REGINALDO ARAKAKI
CO-ORIENTADOR: ENG. MARCOS ANZAI

SÃO PAULO
2006

Sumário

1 Introdução

1.1 Objetivo

O objetivo do projeto ***Servidor de Autorização de Eventos Assíncronos para Aplicações de Mobilidade*** é conduzir um estudo de técnicas e tecnologias através da proposição e criação de um produto final concreto utilizando-se dos conhecimentos adquiridos ao longo do curso de graduação em Engenharia de Computação. O projeto, em si, visa propor uma infra-estrutura capaz de viabilizar aplicações que se apresentam como soluções para problemas da sociedade resolvíveis através do emprego da tecnologia. Esta infra-estrutura viabiliza também a criação de produtos comerciais através da exploração de oportunidades e formulação de modelos de negócio.

O ***Servidor*** consiste de um conjunto de componentes que atuam cooperativamente para criar eventos assíncronos que carregam uma requisição e que são entregues para clientes com mobilidade – telefones celulares, por exemplo. Esta requisição deve consistir tipicamente de uma pergunta cuja resposta pode ser apenas afirmativa ou negativa. O ***Servidor*** tem como tarefa receber a resposta, mas caso não seja possível contatar um dos clientes com mobilidade, deve retransmitir a requisição para os próximos clientes na lista de opções. Ao final deste processo, o ***Servidor*** consolida os resultados, deixando-os prontos para serem retirados. Verifica-se, portanto, que o ***Servidor*** deve ser conectado a algum serviço legado, pois sua função é fornecer um serviço de autorização totalmente assíncrono e desacoplado. Este serviço legado fornece parâmetros como tempo de resposta, lista de clientes que serão contatados e a pergunta que será feita.

Através do que foi discutido, conclui-se que o projeto propõe o estudo de características de negócio como a mobilidade, a segurança, a alçada múltipla e o assincronismo e de características de arquitetura como o desempenho, a garantia de entrega e a precisão.

O projeto de formatura foi desenvolvido ao longo de 2006 na Escola Politécnica da Universidade de São Paulo por Carlos Henrique Esteves Mendonça, Cristina Naomi Makibara e Fabio Hideki Aragaki sob a orientação do Prof. Dr. Reginaldo Arakaki e co-orientação do Eng. Marcos Anzai. A empresa Scopus Tecnologia Ltda. participou do desenvolvimento do projeto com suporte técnico e de negócios, exercendo um importante papel como parceira do projeto de formatura.

1.2 Motivação

O projeto ***Servidor de Autorização de Eventos Assíncronos para Aplicações de Mobilidade*** foi idealizado a partir de discussões com o orientador do grupo, Prof. Dr. Reginaldo Arakaki. A estratégia de formulação da proposta foi a análise *top-down* e, para tal, elegeu-se um problema que possuísse duas características especiais, a saber: fosse um problema social, isto é, que afetasse diretamente as pessoas, e que fosse um problema resolvível através da aplicação da Engenharia para a elaboração de uma solução que fizesse uso de tecnologias computacionais.

O problema observado foi a dificuldade e os riscos que crianças e idosos enfrentam ao utilizar os sistemas financeiros, que por sua vez contam com dispositivos de segurança cada vez mais intrincados. O caso especial observado e explorado pelo grupo é o da compra em lojas através do uso do cartão magnético, tanto de crédito quanto de débito. No caso do cartão de crédito, a autorização da transação e a autenticação do portador envolvem uma assinatura, cuja verificação costuma ser freqüentemente negligenciada pelo lojista. O cartão de débito, por

outro lado, baseia-se na senha eletrônica como forma de autorização e autenticação e, para pessoas com dificuldade em memorizar senhas – como é o caso de idosos – verifica-se freqüentemente que muitos optam por tê-la anotada consigo, fragilizando todo o sistema.

Além disso, o uso de cartões de crédito por crianças levanta a questão da responsabilidade inerente. Trata-se, portanto, de um dos dilemas dos pais: como garantir independência e promover a educação financeira ao mesmo tempo em que se evita o mau uso do dinheiro?

A resposta a estes problemas é o que motiva o sistema computacional desenvolvido como projeto de formatura do grupo. Propõe-se minimizar os problemas através da transferência da responsabilidade de se autorizar uma transação para outras pessoas que respondam financeiramente e, eventualmente, legalmente pelo comprador. No caso de crianças, esta responsabilidade será assumida, por exemplo, pelos pais, padrinhos, irmãos ou professores e no caso de idosos, a responsabilidade pode ser assumida, por exemplo, pelos filhos, netos ou médicos. Estes responsáveis possuem, portanto, a tarefa de responder positivamente ou negativamente à requisição de autorização que podem receber a qualquer momento. O grupo considera que o melhor meio de se receber esta requisição é através do telefone celular, já que se trata de um dispositivo e uma rede que, por construção, possuem características de conectividade permanente e funcionamento regido por eventos assíncronos.

Sendo assim, é necessário um sistema computacional que atue como agente de transporte das requisições de autorização e de suas respostas. Este sistema computacional é, portanto, o produto desenvolvido como projeto de formatura. Em outras palavras, o que se realizou foi a observação da sociedade, escolha de um problema que representasse uma oportunidade de ação, idealização de uma aplicação que resolvesse este problema e, por fim, utilização da

Engenharia como ferramenta para a construção da infra-estrutura necessária para viabilizar estas idéias. O emprego da análise *top-down* representa este processo.

O grupo reconhece que esta aplicação introduz riscos ao processo de compra. Entretanto, o foco do projeto de formatura não é o desenvolvimento da aplicação, e sim o desenvolvimento de sua infra-estrutura. Por causa disso, duas ações foram necessárias: estudar os eventuais riscos da aplicação – a fim de mitigá-los através da introdução de customizações na infra-estrutura e através de um conhecimento sólido a cerca dos problemas que eventualmente afetarão o usuário final – e estudar outras aplicações que se beneficiam deste sistema computacional e também justificam sua construção.

O estudo de outras aplicações além da ***Venda para Clientes com Necessidade Especiais***, são mais motivações para o desenvolvimento do sistema computacional, pois é possível perceber que há muitas oportunidades a serem exploradas e que podem fazer uso do ***Servidor de Autorização de Eventos Assíncronos para Aplicações de Mobilidade***, sendo necessário apenas a formulação de planos de implementação ou mesmo de modelos de negócio.

Do ponto de vista computacional, o desenvolvimento desde sistema é motivado pela oportunidade de se estudar as soluções necessárias que atendem aos requisitos do sistema. Desafios de negócios e de arquitetura, bem como o uso de novas tecnologias foram os tópicos que o projeto de formatura permitiu estudar. Trata-se, portanto, de conhecimento disponível para a construção de novos projetos ou mesmo para a extensão deste.

1.3 Organização

O objetivo deste documento é apresentar de forma abrangente, porém objetiva, o processo de criação do projeto ***Servidor de Autorização de Eventos Assíncronos para Aplicações de Mobilidade***, bem como apresentar conclusões e levantar questões a cerca do estudo

conduzido. Para tanto, este documento aborda tanto o produto final do projeto de formatura quanto o seu processo de criação e, para isso, o grupo estabeleceu junto com o seu orientador, Prof. Dr. Reginaldo Arakaki, a seguinte estrutura de tópicos:

- Capítulo 2, "Aspectos conceituais"

  Este capítulo é iniciado com a apresentação da proposta do ***Servidor de Autorização de Eventos Assíncronos para Aplicações de Mobilidade*** sobre o ponto de vista de Engenharia, isto é, a descrição da infra-estrutura computacional e de comunicações do projeto. Em seguida, é apresentado como foi a evolução da arquitetura do sistema e então são apresentadas as tecnologias utilizadas. Por fim, são discutidas algumas considerações a cerca destes aspectos conceituais e sua influência no produto final.

- Capítulo 3, "Especificação do projeto de formatura"

  Este capítulo apresenta as especificações do projeto, isto é, os requisitos de software e os documentos pertinentes segundo as metodologias de Engenharia de Software.

- Capítulo 4, "Metodologia"

  Este capítulo foca na metodologia utilizada durante o processo completo de construção do projeto de formatura.

- Capítulo 5, "Projeto e implementação"

  Este capítulo apresenta os documentos referentes ao planejamento do projeto de formatura, o progresso e o resultado final do processo.

- Capítulo 7, "Considerações finais"

Este capítulo resume as observações feitas pelo grupo a cerca do produto final e do processo. Por fim, são apresentadas sugestões da continuação do estudo iniciado por este projeto para inspirar trabalhos futuros.

2 Aspectos conceituais

2.1 Proposta

Conforme descrito no capítulo 1.2, a proposta de construção do ***Servidor de Autorização de Eventos Assíncronos para Aplicações de Mobilidade*** envolve garantir certas características de negócio e de arquitetura, a saber:

2.1.1 Características de negócio

- **Mobilidade:** o sistema deve ser capaz de se comunicar com clientes com mobilidade como, por exemplo, os telefones celulares. Esta comunicação deve ser feita através de interfaces comuns que abstraiam a implementação eventualmente única da cada rede de acesso. Por outro lado, o sistema deve também ser capaz de interpretar os dados fornecidos pelos componentes de comunicação a fim de extrair informações de comportamento. Por exemplo, para cada rede de telefonia celular ou operadora, pode existir um conjunto de componentes proprietários que realize o envio de mensagens. Se estes componentes oferecerem medidas de desempenho ou códigos de resultado das operações, estes dados devem ser interpretados e convertidos para os códigos internos utilizados pelo ***Servidor***.

- **Segurança:** o projeto deve prever um nível mínimo de segurança entre os componentes com autenticação em nível de aplicação para cada ator participante.

- **Alçada múltipla:** o processo de autorização deve ser feito através de canais distintos, ou seja, o canal de entrada e saída que recebe do servidor legado um novo pedido de autorização deve ser diferente do canal de entrada e saída que recebe dos clientes a resposta positiva ou negativa da autorização.

- **Eventos assíncronos:** todos os eventos ocorridos no ***Servidor*** devem ser assíncronos, isto é, não é necessário que um evento seja iniciado apenas mediante uma requisição. Para os clientes com mobilidade, o resultado será que irão receber as requisições de autorização assim que elas estiverem preparadas e forem enviadas, sem precisarem interagir com o ***Servidor*** para recebê-las.

2.1.2 Características de arquitetura

- **Garantia de entrega:** toda a comunicação entre componentes deve ter garantia de entrega, ou seja, todo componente que envia mensagens – seja internamente entre componentes do Servidor ou externamente entre o ***Servidor*** e os serviços a ele conectados – deve ter informações sobre o recebimento bem ou mal sucedido das mensagens. Quando esta informação não estiver disponível, mecanismos de temporização devem controlar a validade das informações.

- **Desempenho:** o sistema deve possuir mecanismos que permitam a escalabilidade, isto é, com o aumento de carga no ***Servidor***, deve ser possível alocar dinamicamente recursos de *hardware* para comportar esta demanda. Requisições que não puderem ser atendidas, não podem ser perdidas. O comportamento esperado é que elas permaneçam em estado de espera até que seja possível reservar os recursos necessários para o atendimento. Mecanismos de temporização devem controlar a validade das informações que permanecem em estado de espera.

- **Precisão:** o sistema deve utilizar os mecanismos de temporização para garantir tempo de resposta determinístico em determinados pontos de comunicação. Desta

forma, os serviços legados acoplados terão uma informação até certo grau precisa de quanto tempo será necessário, no pior caso, para obter uma resposta.

Estes dois conjuntos de características foram os requisitos levantados para um sistema computacional que permitisse implementar as funcionalidades enunciadas no capítulo 1.2. Conforme será discutido no capítulo 2.2, a arquitetura evoluiu para acomodar as características de negócios e de arquitetura pretendidas e conforme evoluiu o estudo da plataforma sobre a qual foi construído o ***Servidor***.

2.2 Evolução da arquitetura

De acordo com a proposta apresentada no capítulo 2.1, foi projetada uma arquitetura que evoluiu conforme o estudo das tecnologias utilizadas avançou e o projeto amadureceu. Em razão de o estudo inicial ter sido bem conduzido, não foi necessário realizar mudanças significativas na arquitetura ao longo do projeto e apenas pequenas alterações foram necessárias, ora para atender requisitos de implementação da plataforma, ora para introduzir alguma melhoria de implementação.

2.2.1 Primeiro passo: a definição inicial

Precisava-se criar um conjunto de componentes que atendessem os seguintes requisitos básicos:

i. utilização de um padrão de comunicação entre componentes externos;

ii. desacoplamento entre componentes internos e entre componentes externos; e

iii. comunicação através de mensagens assíncronas;

Para atender estes requisitos, foi determinado que dois *web services* estabeleceriam os canais de comunicação, atendendo desta forma o requisito (i). O primeiro seria bidirecional e seria o canal entre o servidor legado e o ***Servidor***. No caso deste *web service* bidirecional, para que não fosse desrespeitado o requisito (iii), seria utilizado o recurso de comunicação assíncrona, iniciando-se o envio da requisição e obtenção da resposta em instantes determinados em tempo de programação. O segundo seria unidirecional (entrada apenas) e seria o canal de comunicação entre os dispositivos móveis e o ***Servidor***.

A lógica de gerenciamento do ciclo de vida das transações seria implementada por um componente central que criaria várias instâncias de objetos representando cada transação, cada um deles associado a *n* máquinas de estado, onde *n* seria o número de dispositivos móveis que cada objeto representando uma transação poderia contatar. A comunicação entre os componentes internos do SAEA seria feita através de uma entidade do tipo "Topic" do Java Messaging Service (JMS), atendendo desta forma o requisito (ii) e (iii).

2.2.2 Segundo passo: a introdução de um *web service* adicional

Ao se estudar os exemplos de aplicação que poderiam utilizar o ***Servidor***, foi constatado que tipicamente haveria um tempo de resposta da ordem de minutos a dias. Sendo assim, embora isso não representasse problemas para as conexões HTTP – devido à utilização de *web services* no modo assíncrono – entre o servidor legado e o ***Servidor***, seria interessante separa o canal de comunicação para que se tivesse dois unidirecionais, aumentando o desacoplamento de componentes e maximizando o potencial de aumento de desempenho através do balanceamento de carga.

2.2.3 Terceiro passo: considerações a cerca da arquitetura *stateless* dos Enterprise Java Beans

Conforme será apresentada ao longo do capítulo 2.4, a programação para a plataforma J2EE estabelece algumas restrições inexistentes na plataforma J2SE. É o caso, por exemplo, dos Enterprise Java Beans, que possuem como característica não guardarem seu estado após os processos de instanciação e execução. Sendo assim, logo que sua execução termina, ele pode ser descartado em algum momento aleatório pelo processo *garbage collector* e, conseqüentemente, suas variáveis internas que armazenam dados do estado de execução de uma transação são perdidas.

Esta característica da plataforma exigiu que se reformulasse a implementação a fim de incluir uma camada de persistência no ***Servidor***. Esta camada de persistência seria a responsável por armazenar os dados referentes aos vários momentos de manipulação de uma transação, isto é, da sua criação até sua finalização. Esta foi a última mudança da arquitetura e que produziu a versão final do ***Servidor***.

Conforme foi discutido, a definição da arquitetura foi um processo iterativo e que acompanhou o estudo da plataforma e das tecnologias auxiliares. Este processo iterativo foi necessário porque a plataforma J2EE possui diversas particularidades e detalhes que exigem um estudo aprofundado. Em razão de restrições de tempo, as atividades de estudo e implementação ocorram paralelamente.

2.3 Tecnologias (conceitos)

Nesta seção estão apresentados os conceitos por trás das tecnologias utilizadas na construção do ***Servidor***, independente de uma implementação específica. A apresentação dos conceitos é feita propositalmente de forma superficial, apenas para contextualizá-los. Sugestões de literatura recomendada para cada tecnologia estão relacionadas ao final de cada seção.

2.3.1 *Web services*

2.3.1.1 Descrição

Web services são definidos pelo World Wide Web Consortium (W3C) – principal órgão de definição de padrões para a World Wide Web (W3) – como "[...] um sistema de software projetado para prover inter-operação máquina-máquina através de uma rede."[1].

Há ao menos três principais implementações do conceito de *web services*: Remote Procedure Call (RPC), Services Oriented Architecture (SOA) e RESTful Web Services (REST). Entretanto, para o projeto, o interesse está na implementação SOA, que é um tipo de arquitetura de Tecnologia de Informação e Comunicações (TIC) que provê integração de operações de negócio através de serviços interligados que podem ser acessados através de uma rede. Estes serviços são caixas pretas e interagem com o meio – outros sistemas, também chamados de "clientes consumidores" – através de interfaces bem definidas. Estas interações são feitas através de entidades conhecidas como "contratos de serviço", que definem as operações possíveis entre provedores de serviço e clientes consumidores. A interação será, portanto, feita através de um sistema de passagem de mensagens.

As mensagens em questão e os contratos de serviço são descritos em XML, uma linguagem de marcação recomendada como padrão pela W3C e, portanto, um padrão *de facto* de comunicação de dados em sistemas modernos e que buscam independência de plataforma.

2.3.1.2 Influência no projeto

Com a utilização de *web services* para a interface entre os componentes internos do ***Servidor*** e os componentes externos, foi possível garantir que até certo grau, o sistema garantiria

[1] WEB SERVICE, Wikipedia, the free encyclopedia
Disponível em: <http://en.wikipedia.org/wiki/Web_service>.
Acessado em: 03 de dezembro de 2006.

interoperabilidade independente de plataforma. Como se espera ter diferentes plataformas de clientes com mobilidade e de servidores de serviço legados, esta garantia é importante na análise do fator atratividade do uso do ***Servidor***.

2.3.1.3 Sugestões de literatura para aprofundamento

JAVA TECHNOLOGY AND WEB SERVICES
Disponível em: <http://java.sun.com/webservices/>.
Acessado em: 03 de dezembro de 2006.

APACHE WEB SERVICES PROJECT
Disponível em: <http://ws.apache.org/>.
Acessado em: 03 de dezembro de 2006.

WEB SERVICES AND THE MICROSOFT PLATFORM
Disponível em: <http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnwebsrv/html/wsmsplatform.asp>.
Acessado em: 03 de dezembro de 2006.

A WEB SERVICES PRIMER, webservices.xml.com
Disponível em:
<http://webservices.xml.com/pub/a/ws/2001/04/04/webservices/index.html?page=2>.
Acessado em: 03 de dezembro de 2006.

2.3.2 Messaging

2.3.2.1 Descrição

O conceito de *messaging* está atrelado ao conceito de *message-oriented middleware* (MOM), que se define como um software de comunicação intra-aplicações e que se baseia no mecanismo de passagem de mensagens assíncronas, ao invés do mecanismo mais comum de requisição seguida de resposta[1]. Ele se originou como uma alternativa para a integração de sistemas legadas e também para viabilizar o processamento distribuído, especialmente em sistemas interligados através de uma rede.

---

[1] MESSAGE ORIENTED MIDDLEWARE, Wikipedia, the free encyclopedia
Disponível em: <http://en.wikipedia.org/wiki/Message_Oriented_Middleware>.
Acessado em 03 de dezembro de 2006.

Tipicamente um sistema de mensagens é implementado através de filas, de forma que a comunicação é estabelecida ponto-a-ponto. Existe, entretanto, implementações – como a estrutura Topic do JMS – que realizam *multicast* e *broadcast* de mensagens. As filas e as outras estruturas com a mesma funcionalidade possuem recursos de garantia de entrega e persistência de mensagens. No caso de um sistema *middleware*, funcionalidades adicionais estão tipicamente presentes, como tradução entre protocolos e tratamento dos dados.

2.3.2.2 Influência no projeto

A necessidade de se utilizar um sistema de passagem de mensagem no projeto surgiu da necessidade de estabelecer a comunicação entre vários módulos independentes dentro do ***Servidor***. Conforme descrito no capítulo 2.2, a arquitetura evolui até o ponto em que inclui uma camada de persistência dos dados. Esta camada atua como o ponto comum de acesso para os vários módulos e é possível demonstrar que, neste caso, não é necessário passar mensagens entre os componentes,.

Entretanto, o estudo da implementação JMS demonstrou que manter a comunicação baseada em mensagens traz vantagens como a eliminação do acoplamento entre componentes internos e entre componentes internos e externos e a possibilidade de se distribuir o processamento das transações em *clusters* de forma totalmente transparente para a aplicação.

2.3.2.3 Sugestões de literatura para aprofundamento

THE JAVA EE 3 TUTORIAL
Disponível em: <http://java.sun.com/javaee/5/docs/tutorial/doc/>.
Acesseado em: 03 de dezembro de 2006.

JAVA EE 5 SDK
Disponível em: <http://java.sun.com/javaee/5/docs/api/index.html>.
Acesseado em: 03 de dezembro de 2006.

2.3.3 *Object-relational mapping*

2.3.3.1 Descrição

O *object/relational mapping* ou mapeamento entre objetos e dados relacionais é uma técnica de programação que estabelece uma ligação direta entre a representação de dados da orientação a objetos e a representação de dados dos bancos de dados relacionais, criando o efeito de um banco de dados de objetos virtual[1].

Com a introdução da programação orientada a objetos criou-se um novo desafio já que se fez necessário interligar dois modelos de armazenamento de dados com características de construção distintas: o modelo de dados relacional e o modelo de dados orientado a objetos. Este desafio é conhecido como o "conflito de impedância no mapeamento objeto-relacional" e apresenta as dificuldades de se transportar dados entre as duas representações sem perda de informação, já que existe uma falha semântica, isto é, uma diferença na descrição dos dados entre as duas representações. Por exemplo, no modelo relacional, não é possível representar conceitos comuns da orientação a objetos como classes de objetos, herança e polimorfismo. Entretanto, o modelo relacional define a representação de dados através de relações *n-árias*, permitindo operações nos dados através da lógica de predicados.

A solução para minimizar o problema de representação dos dados e conversão entre os dois modelos veio com a introdução das ferramentas para mapeamento entre objetos e dados relacionais. Sua função é atuar como uma camada de abstração do banco de dados e se mostrar para a aplicação como uma camada de persistência dos dados. Muitas bibliotecas que implementam esta técnica foram criadas e, desde então, alguns padrões surgiram a fim de normalizar as interfaces.

---

[1] OBJECT-RELATIONAL MAPPING, Wikipedia, the free encyclopedia
Disponível em: <http://en.wikipedia.org/wiki/Object-relational_mapping>.
Acessado em: 03 de dezembro de 2006.

2.3.3.2 Influência no projeto

No caso do projeto de formatura, foi usada a biblioteca Hibernate, um projeto de código aberto escrito na linguagem Java e que inspirou a especificação Enterprise Java Beans 3.0 (ou JSR-220), também utilizado no projeto. O uso do mapeamento entre objetos e dados relacionais permitiu que se manipulasse diretamente objetos na aplicação, de forma que a camada de persistência – introduzida em razão da impossibilidade se manter uma cópia em memória dos dados de cada transação – teve sua implementação totalmente abstraída, de forma que sua operação é transparente para o resto da aplicação. Detalhes como integridade dos dados armazenados e acesso concorrente ao banco de dados passaram a ser responsabilidade da biblioteca Hibernate e da plataforma J2EE.

2.3.3.3 Sugestões de literatura para aprofundamento

HIBERNATE
Disponível em: <http://www.hibernate.org>.
Acesseado em: 03 de dezembro de 2006.

2.3.4 Dependency Injection

2.3.4.1 Descrição

Dependency Injection é um *design pattern* e um modelo de arquitetura também conhecido como Inversion of Control. A arquitetura tem como objetivo unir os componentes ao invés de deixar que eles se encarreguem de estabelecer sozinhos suas conexões. Desta forma, a união de componentes passa a ser responsabilidade de um componente chamado de *factory*.

2.3.4.2 Influência no projeto

As aplicações deste *design pattern* são inúmeras e muitos projetos fazem uso dela. Em especial, pode-se mencionar o projeto Spring que entre outras funções facilita o trabalho de ligação entre componentes dependentes e é a base da biblioteca Hibernate. Além disso,

Dependency Injection é utilizado extensivamente na plataforma J2EE a partir da versão 5. Este *design pattern* permite que declare dependência entre a aplicação e recursos do servidor que são verificados em tempo de instalação, ao contrário da abordagem comum, onde a verificação ocorre em tempo de execução. Com a exploração deste recurso, pode-se garantir uma redução dos incidentes de execução do *software* contanto que se tenha um forte processo de gerenciamento de configuração.

2.3.4.3 Sugestões de literatura para aprofundamento

INVERSION OF CONTROL CONTAINERS AND THE DEPENDENCY INJECTION PATTERN, Martin Fowler
Disponível em: <http://www.martinfowler.com/articles/injection.html>.
Acesseado em: 03 de dezembro de 2006.

2.3.5 Wap Push

2.3.5.1 Descrição

O Wap Push é ama tecnologia utilizada para o envio de mensagens assincronamente para dispositivos móveis. A mensagem recebida contém uma Unified Resource Identifier (URI) para alguma página na Internet, sendo que o usuário a acessa apenas aceitando essa mensagem. Essa tecnologia é baseada no modelo cliente-servidor, embora não exista uma requisição feita explicitamente pelo cliente ao servidor.

2.3.5.2 Arquitetura

A arquitetura do *framework push*[1] é formada por três entidades principais: o cliente Wap, o Push Proxy Gateway e o Push Initiator, sendo que a comunicação entre eles é feita por dois protocolos: o Push Access Protocol e o Push Over-The-Air Protocol.

[1] PUSH ARCHITECTURAL OVERVIEW
Disponível em:
<http://www.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-250-pusharchoverview-20010703-a.pdf>.
Acessado em: 03 de dezembro de 2006.

- **Push Initiator:** realiza a montagem e envio da mensagem *push*;
- **Push Access Protocol:** encapsula a mensagem *push* e a envia ao Push Proxy Gateway;
- **Push Proxy Gateway:** responsável pela conexão das redes *internet* e as redes utilizadas pelos dispositivos móveis;
- **Push Over-The-Air Protocol:** encapsula a mensagem *push* e a envia ao cliente Wap; e
- **Cliente Wap:** entidade que recebe a mensagem *push*.

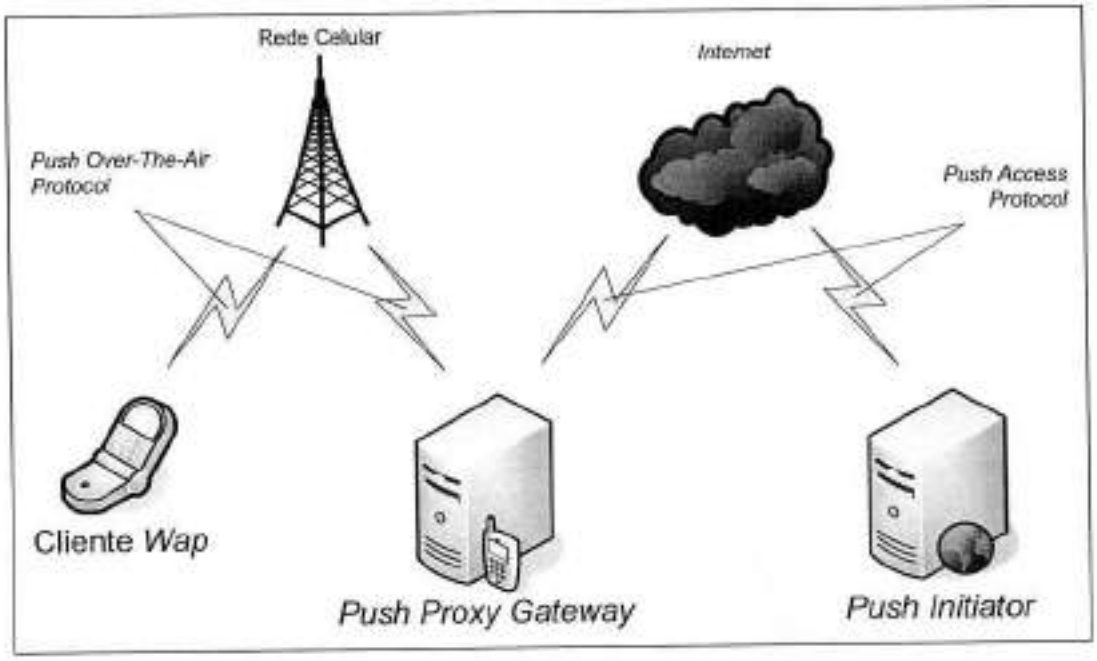

Figura 2.1: Arquitetura do *framework push*.

O desenvolvimento do módulo de envio de mensagens para dispositivos móveis do projeto concentrou-se no Push Initiator, ou seja, os dados necessários foram estruturados de maneira

que fossem encapsulados pelo protocolo Push Access Protocol e enviados ao Push Proxy Gateway para processamento.

2.3.5.3 Push Initiator

O Push Initiator é uma aplicação que realiza a montagem e envio das mensagens *push* (no formato XML) utilizando o Push Access Protocol sobre o protocolo HTTP.

A montagem da mensagem *push* deve ser feita de acordo com o tipo de conteúdo a ser enviado (Service Indication ou Service Loading), ou seja, são definidos os valores dos parâmetros de cada tipo, sendo o Push Proxy Gateway a entidade responsável pela validação dessa mensagem. Além disso, o Push Initiator deve definir também os valores dos parâmetros de controle de entrega da mensagem *push*.

Os dois principais tipos de conteúdo da mensagem *push* são o Service Indication e o Service Loading.

- Service Indication[1]: utilizado para o envio de notificações ao dispositivo móvel de maneira assíncrona. Essas notificações são compostas de uma mensagem de texto e uma URI, sendo que são exibidas assim que são recebidas. O usuário tem a possibilidade de acessar o serviço indicado pela URI no momento da chegada da mensagem ou acessá-lo posteriormente, já que a notificação pode ficar armazenada no dispositivo móvel.

[1] WAP SERVICE INDICATION SPECIFICATION
Disponível em:
<http://www.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-167-serviceind-20010731-a.pdf?doc=wap-167-serviceind-20010731-a.pdf>.
Acessado em: 03 de dezembro de 2006.

- Service Loading[1]: utilizado para o envio de notificações ao dispositivo móvel de maneira assíncrona, mas não oferecendo a possibilidade de acesso ao serviço indicado pela URI para o usuário. Isso ocorre, pois, no momento em que a notificação é recebida, o serviço é acessado automaticamente, ou seja, a interação do usuário é mínima.

O tipo de conteúdo utilizado no projeto foi o Service Indication, pois um usuário, que tenha recebido uma mensagem que não era para ele, não precisa acessar o serviço indicado pela URI, sendo a mensagem descartada em seguida, ou seja, o usuário ainda tem a possibilidade de aceitá-la ou não, sendo que no Service Loading essa escolha não existe.

2.3.5.4 Push Access Protocol

O protocolo Push Access Protocol foi criado com o objetivo de realizar a entrega de conteúdo do Push Initiator ao Push Proxy Gateway. Sua especificação foi projetada de modo que ele possa utilizar os serviços oferecidos por qualquer protocolo de transporte, embora o protocolo predominante seja o HTTP.

A mensagem *push* criada pelo Push Initiator possui duas entidades em sua estrutura, sendo uma de controle e a outra de conteúdo.

---

[1] WAP SERVICE LOADING SPECIFICATION
Disponível em:
<http://www.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-168-serviceload-20010731-a.pdf>.
Acessado em: 03 de dezembro de 2006.

Figura 2.2: Estrutura de uma mensagem *push*.

A entidade de controle[1] é um documento XML descrito por um Document Type Definition (DTD) com a finalidade de estabelecer quais serão os parâmetros de entrega da mensagem *push*, além de qual operação deverá ser realizada.

A entidade de conteúdo não possui um formato específico, pois qualquer tipo de conteúdo pode ser enviado. No caso específico de envio de alertas, os dois principais tipos de conteúdos (Service Indication e Service Loading) estão no formato XML e cada um possui seu descritor DTD[2,3].

Por meio do Push Access Protocol, o Push Initiator pode realizar as seguintes operações:

- Enviar mensagem *push*;
- Substituir uma mensagem *push* enviada anteriormente;

[1] DOCUMENTO DTD
Disponível em: <http://www.openmobilealliance.org/tech/DTD/pap_2.1.dtd>.
Acessado em: 03 dezembro de 2006.
[2] DOCUMENTO DTD
Disponível em: <http://www.openmobilealliance.org/tech/DTD/si_1.0.dtd>.
Acessado em: 03 de dezembro de 2006.
[3] DOCUMENTO DTD
Disponível em: <http://www.openmobilealliance.org/tech/DTD/sl_1.0.dtd>.
Acessado em: 03 de dezembro de 2006.

- Cancelar uma mensagem *push*;
- Verificar qual o estado de uma mensagem *push* no Push Proxy Gateway; e
- Consultar características de dispositivos móveis.

O Push Proxy Gateway realiza somente uma operação:

- Confirmar que um dispositivo móvel recebeu a mensagem *push* ao Push Initiator.

Todas as operações descritas anteriormente são síncronas, ou seja, é feita uma requisição e uma resposta é obtida, sendo que cada operação tem uma resposta específica[1].

A única operação utilizada no projeto é a de envio de mensagens *push*, pois todas as outras dependem da implementação feita pelo Push Proxy Gateway, ou seja, apenas a operação básica (envio de mensagens) pode estar disponível.

2.3.5.5 Push Proxy Gateway

O Push Proxy Gateway é o elemento fundamental na arquitetura do *framework push*, pois ele é responsável pela conectividade entre as redes internet e as redes celulares.

As principais funções do Push Proxy Gateway são as seguintes:

- Processamento de mensagens enviadas pelo Push Initiator: aceita ou rejeita as mensagens *push* recebidas por meio de validação contra descritores DTD, além de encaminhar as mensagens aceitas ao destinatário utilizando o protocolo Push Over-The-Air;

---

[1] PUSH ACCESS PROTOCOL SPECIFICATION
Diponível em: <http://www.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-247-pap-20010429-a.pdf>.
Acessado em: 03 de dezembro de 2006.

- Notificação ao Push Initiator de que as mensagens foram recebidas pelos destinatários: envia uma mensagem ao Push Initiator notificando-o sobre o recebimento da mensagem *push* pelo dispositivo móvel. Isso ocorre somente se o Push Initiator solicitou essa função;

- Consulta sobre o estado de uma mensagem recebida do Push Initiator: uma verificação sobre o estado da mensagem *push* é feita, sendo a resposta enviada ao Push Initiator; e

- Cancelamento de uma mensagem *push*: cancela o envio de uma mensagem *push* desde que ela esteja pendente, ou seja, ainda não tenha sido enviada ao destinatário.

O endereçamento de dispositivos móveis[1] pode ser feito por meio de um mapeamento entre um identificador de usuário e um dispositivo móvel ou por meio de endereçamento direto do dispositivo.

2.3.5.6 Push Over the Air Protocol

O protocolo Push Over-The-Air[2] é utilizado na comunicação entre o Push Proxy Gateway e o dispositivo móvel para o envio da mensagem *push*, sendo que ele utiliza os serviços oferecidos pelo protocolo Wireless Session Protocol (WSP) ou HTTP durante essa comunicação.

---

[1] PUSH PROXY GATEWAY SERVICE SPECIFICATION
Diponível em: <http://www.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-249-ppgservice-20010713-a.pdf>.
Acessado em: 03 de dezembro de 2006.

[2] PUSH OTA PROTOCOL SPECIFICATION
Diponível em: <http://www.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-235-pushota-20010425-a.pdf>.
Acessado em: 03 de dezembro de 2006.

O modo como o dispositivo móvel se comunica com o Push Proxy Gateway para o recebimento da mensagem *push* não está no escopo do projeto.

2.3.5.7 Cliente Wap

O cliente Wap é o destinatário da mensagem *push* enviada pelo Push Initiator, sendo que o recebimento das mensagens é possível somente se existe uma conexão com o Push Proxy Gateway.

2.4 Tecnologias (pacotes)

2.4.1 Sun Java Enterprise Edition

2.4.1.1 Enterprise Java Beans

Um Enterprise Java Bean (EJB) é um componente de código contendo variáveis e métodos que implementam uma lógica de negócios. Este componente pode tanto ser utilizado sozinho ou como parte de um sistema maior através da interligação de vários EJBs. Há dois tipos de EJBs: Session Beans e Message Driven Beans. O primeiro é tipicamente uma associação temporária com um cliente, isto é, após sua execução – iniciada através da chamada e instanciação feitas por um cliente – ele é destruído e os dados nele armazenados são perdidos. O Message Driven Bean possui as mesmas características, mas é tipicamente acionado por chamadas realizadas através do JMS.

Até a versão 4 do J2EE, os Entity Beans também eram considerados EJBs. Entretanto, atualmente eles são simplesmente considerados entidades da plataforma de persistência do Java. Os Entity Beans agem como uma representação em memória de um dado armazenado em um banco de dados tipicamente relacional. Assim que é usado e removido, o gerenciador de persistências do J2EE se encarrega de manter a integridade dos dados junto ao banco de dados.

Um EJB precisa ser executado em um *container*. *Containers* são representados por servidores de aplicação que aderem às especificações do J2EE. Através deles, é possível criar facilmente aplicações cliente/servidor em clientes magros, pois toda a complexidade de mecanismos como gerenciamento de estado e de transação, *multithreading*, balanceamento de carga e outros detalhes de implementação de baixo nível são todos de responsabilidade do servidor de aplicação, de forma que o programador deve apenas se preocupar com a configuração dos recursos e da criação da lógica de negócios. Alguns dos principais *containers* disponíveis comercialmente são o Sun Application Server, IBM WebSphere, RedHat JBoss, BEA WebLogic e SAP NetWeaver.

Algumas das mais importantes características dos EJBs no contexto do ***Servidor*** são a possibilidade de se beneficiarem com balanceamento de carga totalmente transparente para a aplicação quando instalados em um *cluster* de servidores e o acesso direto a recursos do JMS e da arquitetura de banco de dados do J2EE. O Sun Application Server 9.0 – também conhecido como o projeto de código aberto Glassfish – foi selecionado para o projeto de formatura porque é um dos únicos atualmente que aderem às especificações J2EE 5, permitindo que fossem explorados os recursos de interação com o banco de dados segundo a especificação EJB 3.0, as novas ferramentas para criação de *web services* mais aderentes aos padrões estabelecidos na indústria e os recursos como Java Annotations que permitem explorar o *design pattern* Dependency Injection a fim de se obter um código final menor e menos sujeito a erros.

2.4.1.2 Message Driven Beans

Conforme discutido no capitulo 2.4.1.1, os Message Driven Beans (MDBs) possuem comportamento semelhante aos Session Beans, mas são tipicamente ativados através de uma chamada assíncrona gerada por uma mensagem do JMS. A lógica de um MDB é totalmente

implementada em um método `onMessage()` que é automaticamente e assincronamente chamado pelo JMS após ele ter instanciado o MDB e uma mensagem estar disponível. Para isso, o MDB deve estar ligado à uma das estruturas do JMS: Topic ou Queue.

A estrutura Topic representa um instrumento para transporte de mensagens entre componentes através de *multicast* segundo uma relação de publicação e inscrição. Em outras palavras, um componente ao publicar uma mensagem em uma estrutura Topic estará se comunicando com vários MDBs que nela estejam inscritos. A estrutura Queue, por outro lado, funciona através de uma relação ponto a ponto, isto é, apenas um componente pode entregar mensagens à estrutura e apenas um MDB pode recebê-las. Está é a implementação mais comum entre as arquiteturas de *messaging* e é também a mais simples e eficiente. A estrutura Topic possui um recurso complexo de persistência de mensagens no caso de MDBs só estiverem disponíveis e inscritos nela após a publicação da mensagem. Ambas, entretanto, possuem mecanismos de garantia de entrega.

2.4.1.3 Especificação EJB 3.0 (ou JSR-220)

A especificação EJB 3.0 define todos os tipos de Enterprise Java Beans, suas características e suas funcionalidades. Em especial, há definições a cerca do gerenciador de persistência do J2EE que utiliza uma interface comum para ser acesso através de EJBs independente da biblioteca de mapeamento entre objetos e dados relacionais. As definições, no entanto, foram em muitos casos baseadas nas definições anteriormente encontradas na biblioteca Hibernate.

2.4.2 Hibernate

O Hibernate é uma biblioteca de mapeamento entre objetos e dados relacionais e foi escolhida para o projeto pois além de ser distribuída gratuitamente, é a biblioteca mais usada em ambientes Java e inspirou a definição EJB 3.0. Além disso, outro ponto forte desta biblioteca

é que ela possui compatibilidade com versão anteriores do J2EE, do J2SE e até mesmo compatibilidade com o Microsoft .NET Framework 1.1 e 2.0. Desta forma, garantiríamos que mais um componente do ***Servidor*** seria facilmente portado entre as principais plataformas.

Conforme discutido no capítulo 2.3.3, uma ferramenta de mapeamento entre objetos e dados relacionais atua como um banco de dados orientado a objetos virtual. Desta forma, do ponto de vista da aplicação a implementação do banco de dados é transparente, pois manipula-se objetos diretamente. Do ponto de vista do banco de dados relacional, por outro lado, há algumas modificações. Cada tabela deve representar uma classe e heranças devem ser representadas através de relacionamentos entre chaves primárias e chaves estrangeiras. Este método de relacionamento é chamado *one-subclass-per-table*.

Conforme se pode examinar no código fonte, foram utilizadas Java Annotations para configurar o Hibernate, isto é, para descrever os relacionamentos entre as variáveis de cada objeto e as colunas de cada tabela. Esta descrições também estabelecem o relacionamento entre o tipo de dado de cada variável na linguagem Java e o tipo de dado de cada coluna no banco de dados relacional. Uma mudança na estrutura do banco de dados exige também uma mudança da estrutura de objetos e vice-versa.

2.4.3 Microsoft SQL Server

O Microsoft SQL Server 2005 foi selecionado para o projeto de formatura por diversas razões. Em primeiro lugar, o software pode ser obtido gratuitamente na versão Express, que para fins de criação de um protótipo, possui recursos suficientes. Além disso, a Escola Politécnica disponibiliza através de sua parceira Academic Alliance com a Microsoft versões mais completas do Microsoft SQL Server 2005 sem custo quando usado para fins acadêmicos.

Outra oportunidade explorada foi a de demonstrar a criação de um projeto totalmente independente de plataforma, isto é, a interface com componentes externos utiliza padrões de comunicação e mesmo a implementação de componentes internos pode se facilmente portada. Por fim, o uso do Microsoft SQL Server 2005 traz a oportunidade de explorar ferramentas como o Report Services, que atua como um servidor de armazenamento de geração de relatórios baseados nos dados armazenados. Ao desenvolver estes relatórios, o usuário terá uma representação visual dos resultado das operações armazenadas pelo ***Servidor***, podendo consolidá-los a fim de extrair resultados de operação.

2.4.4 NetBeans

O NetBeans é um projeto de código aberto desenvolvimento em parte pela Sun. Ele se apresenta como o melhor ambiente de desenvolvimento para atuar junto com o Sun Application Server, pois integra ferramentas de instalação, operação e manutenção de aplicações e recursos do *container*, além de oferecer ferramentas avançadas de documentação (pacote Enterprise) – com geração de diagramas UML, por exemplo – e ferramentas de testes (pacote Profiler).

Além dos testes de unidade comuns, estão disponíveis testes de carga que possibilitam a análise sob o ponto de vista macroscópico e microscópico. No ponto de vista microscópico, pode-se analisar quais fragmentos de código representam a maior carga de processador, memória e/ou dispositivos de entrada e saída, o que auxilia na etapa de análise de desempenho e otimização da aplicação.

2.5 Openwave

2.5.1 Openwave Wap Push Library Java 1.0

Implementação da Openwave[1] para a especificação do protocolo Push Access Protocol. Possibilita a realização de todas as operações, além da criação de todos os documentos XML necessários para a geração da mensagem *push*.

2.5.2 Openwave V7 Simulator

Simulador de um dispositivo móvel celular utilizado para a visualização das mensagens *push* recebidas, além de acesso ao serviço indicado pela URI. Deve estar conectado ao *gateway* Wap para que essas mensagens sejam recebidas.

2.5.3 Mobile Access Gateway

O Mobile Access Gateway é um *gateway* Wap oferecido para testes pela Openwave, sendo que ele oferece todas as funcionalidades de um Push Proxy Gateway.

Um cadastro deve ser feito em[2] para que uma identificação seja criada, sendo que a mesma é utilizada como endereço de destino das mensagens *push*. Esse endereço deve ser sincronizado com o endereço Internet Protocol (IP) do computador onde está instalado o Openwave V7 Simulator para que exista um mapeamento.

---

[1] WAP PUSH LIBRARY
Disponível em: < http://developer.openwave.com/docs/wappushjava10/developer_guide.pdf >.
Acessado em: 03 de dezembro de 2006.

[2] WAP GATEWAY
Disponível em: <http://developer.openwave.com/dvl/tools_and_sdk/test_servers/wap_gateway/devgate2.htm>.
Acessado em: 03 de dezembro de 2006.

2.6 Considerações a cerca da escalabilidade

Conforme apresentado no capítulo 2.4.1, o potencial de escalabilidade do ***Servidor*** está intimamente relacionado com a o container sobre o qual ele é instalado e opera – no caso, o Sun Application Server. Como a arquitetura de componentes foi totalmente baseada em EJBs, e como não há acoplamento entre os componentes e poucas dependências de tempo, isto é, operações síncronas, os EJBs podem ser distribuídos ao longo de um *cluster* de servidores de forma totalmente transparente para a aplicação.

Por exemplo, supor o recebimento de uma nova transação através do *web service* de entrada. O comportamento esperado é que os dados alimentados no *web service* pelo servidor legado sejam empacotados em uma mensagem e entregues ao Java Messaging System. A partir deste momento, a operação do *web service* termina, liberando a conexão HTTP com o servidor legado e os recursos alocados para criar o Session Bean do *web service*. A mensagem agora armazenada pelo Java Messaging System só será atendida quando o servidor tiver recursos para alocar um Message Driven Bean. Este Message Driven Bean, quando alocado, pode estar localizado em qualquer servidor de um *cluster*. Como a camada de persistência é centralizada, isto é, todos os servidores estão conectados a um mesmo banco de dados – que, internamente, pode também estar configurado como um cluster, neste caso atuando de forma transparente para os servidores J2EE – a integridade das informações é preservada. Neste caso, há a necessidade de configurações especiais em relação aos limites das transações com o banco de dados. Este comportamento se repete durante todo o ciclo de vida da aplicação e caracteriza o potencial de escalabilidade do sistema.

3 Especificação do projeto de formatura

O documento de Especificação de Requisitos de Software está incluído no Apêndice A.

4 Metodologia

Esse item trata de metodologias de desenvolvimento de software pertinentes ao projeto apresentado. Inicialmente, apresentaremos uma introdução teórica sobre as metodologias, e em seguida, mostraremos como cada uma delas foi utilizada no projeto, que documentos e artefatos foram gerados, e qual o impacto da utilização dessas metodologias no desenvolvimento do projeto.

4.1 Reuso

O conceito de reutilização é um processo já utilizado em várias áreas da engenharia. Na engenharia de software, o aumento da complexidade de sistemas e os prazos cada vez menores para o desenvolvimento de sistemas com qualidade, indicam a necessidade de desenvolvimento mais organizada de reuso. A engenharia de *software* baseada em componentes e o desenvolvimento utilizando-se o modelo de Orientação a Objetos possibilitam essa abordagem de reutilização.

A engenharia de software baseada em componentes (Component-based Software Engeneering (CBSE)) foca no desenvolvimento de grandes sistemas através da integração de componentes de software. Uma aplicação é montada a partir de um conjunto de partes, os componentes de software padronizados e pré-construídos, que são disponibilizados para a integração.

O paradigma de Orientação a Objetos enfatiza a criação de classes que encapsulam os dados tanto quanto os algoritmos usados para manipulá-los. Se bem implementadas, as classes

podem ser reutilizadas em vários projetos de software, através da sua disponibilização em bibliotecas ou repositórios.

Em sistemas complexos, quando os requisitos ainda não estão maduros ou há a incerteza do desenvolvedor em relação à arquitetura, como implementá-la e incertezas sobre a integração dos componentes, o paradigma de prototipação pode ser a melhor abordagem.

A prototipação é o projeto rápido de um protótipo que pode ser avaliado pelo usuário para validar os requisitos. Para os desenvolvedores, o protótipo valida a arquitetura de modo rápido, permitindo que mudanças nos projetos, caso seja constatada a necessidade através do protótipo, aconteçam logo no seu início, quando os custos de alterações no projeto ainda são baixos.

4.2 Prototipação

Existem duas abordagens sobre o paradigma da prototipação [Pressman]. A abordagem de finalidade fechada utiliza protótipos descartáveis, cuja finalidade é demonstrar, de forma rústica, os requisitos. Pode-se, através dele, entender o domínio da aplicação, modelar o problema, mas os requisitos podem ainda estar instáveis e ambíguos.

Na abordagem de finalidade aberta, o protótipo de software é uma primeira evolução do sistema acabado, que pode ser evoluído para dar continuidade ao projeto. Nele, os requisitos de software estão mais maduros, e há um entendimento maior deles pela equipe de desenvolvimento.

Para realizar a prototipagem rápida, três classes genéricas de métodos e ferramentas podem ser utilizadas, a saber:

- Técnicas de 4ª Geração;
- Componentes de software reutilizáveis; e
- Ambientes de especificação formal e prototipagem.

O ciclo de vida da prototipação envolve vários ciclos de revisões sobre o protótipo [Yourdon], conforme mostrado na Figura 4.1. Apesar da abordagem dada por essa autora dizer que o prótotipo é somente um modo de modelar os requisitos do sistema, e que deve ser descartado sempre, esse ciclo de vida reflete bem o modo como um protótipo se desenvolve durante a especificação dos requisitos.

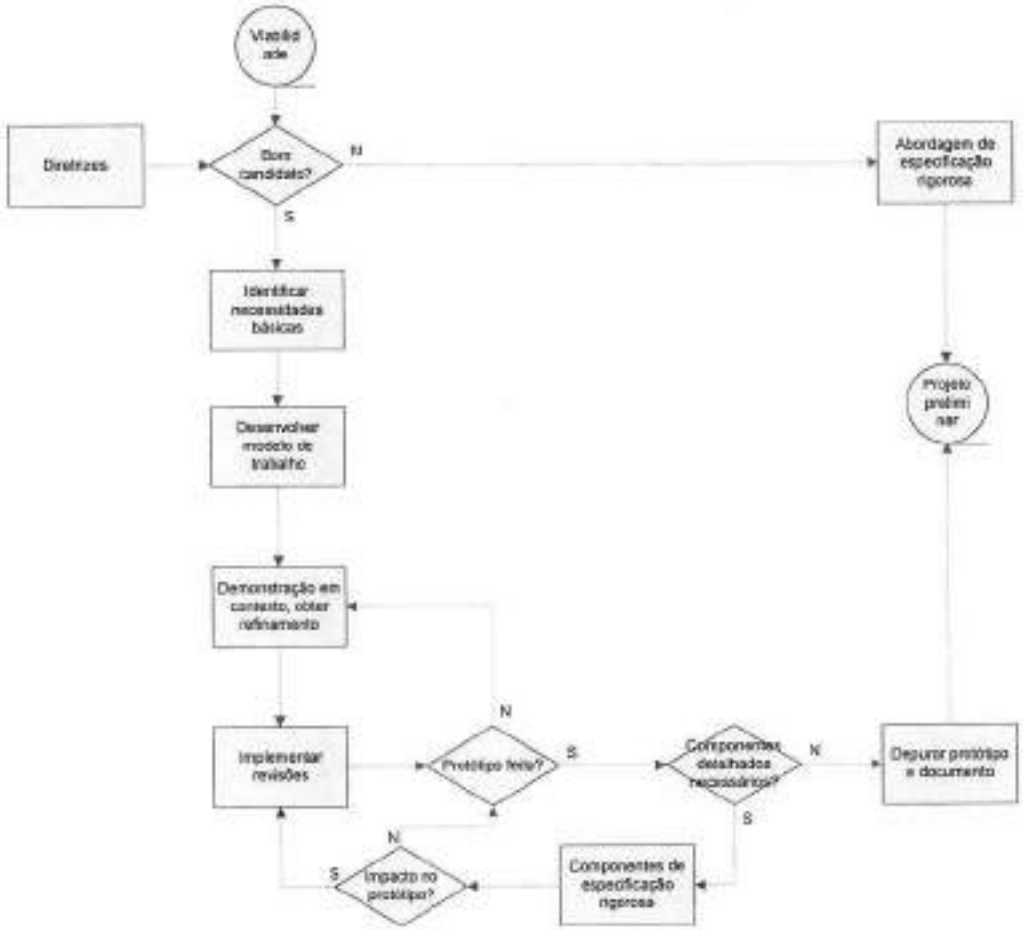

Figura 4.1: Ciclo de vida da prototipação.

4.2.1 Dificuldades

Não havendo o claro entendimento dos desenvolvedores e dos clientes sobre qual a exata função do protótipo, ele pode acabar a se tornar o produto pela vontade do cliente, por achar que o sistema está pronto, já que atende parte de seus requisitos. Isso pode se converter em um problema, já que o protótipo pode não atender a manipulações eficientes de grandes volumes de transações, além de não estar provido de detalhes operacionais como recuperação de erros, documentação do usuário, entre outros.

4.3 Componentização

O termo componente possui diversas definições, dentre as quais podemos destacar:

- Uma parte não trivial de um sistema, independente e substituível, que preenche uma função clara no contexto de uma arquitetura;

- "Componentes são artefatos auto-contidos que nós identificamos claramente em nossos sistemas. Eles têm uma interface, descrevem e/ou executam funções específicas, são documentados separadamente e apresentam um status de reutilização bem definido" [Sametinger]; e

- "Um componente de software é uma unidade de composição com interfaces especificadas contratualmente e dependências explícitas somente de contexto. Um componente de software pode ser implantado independentemente e é usado para composição com terceiros." [Szyperski].

Existem dois subprocessos relacionados à componentização [Pressman]: a engenharia de domínio e o desenvolvimento baseado em componentes.

A engenharia de domínio se preocupa em identificar, construir, catalogar e disseminar um conjunto de componentes de software que tem aplicabilidade aos softwares, em um domínio de aplicação particular. Sua meta principal é permitir que os desenvolvedores compartilhem e reusem os componentes nos vários projetos.

O desenvolvimento baseado em componentes qualifica, adapta e integra os componentes disponibilizados pela engenharia de domínio para utilização em novos sistemas. Há ainda a engenharia de novos componentes desenvolvidos de acordo com as necessidades especificas de um novo projeto.

4.4 Extreme Programming

Extreme Programming é um conjunto de práticas para o desenvolvimento ágil de *softwares*. Essas práticas são organizadas em quatro valores fundamentais:

- **Simplicidade:** deve-se começar da forma mais simples, e ir aperfeiçoando-a, conforme as necessidades presentes;

- **Comunicação:** o desenvolvimento de sistemas de software exige a comunicação, tanto entre os clientes e desenvolvedores quanto entre os próprios desenvolvedores. As práticas do XP colaboram para a comunicação freqüente durante todo o processo de desenvolvimento;

- ***Feedback:*** retorno dos clientes e do próprio sistema (através de testes de unidade e de aceitação) possibilitam manter o projeto dentro do planejado, ou ainda permitem que falhas sejam detectadas antes de iniciar o desenvolvimento de novas funcionalidades; e

- **Coragem:** Os projetos XP partem do principio de que problemas irão ocorrer. No entanto, a equipe deve utilizar redes de proteção para que as conseqüências desses problemas sejam reduzidas ou eliminadas. Iterações curtas, desenvolvimento iterativo e a incorporação do feedback de cada etapa são algumas práticas que tem a função de prover essa rede de proteção ao desenvolvimento.

No projeto, a utilização do XP foi baseada em algumas das 12 práticas [XP]. As práticas utilizadas e os resultados com a utilização são mostrados abaixo.

4.4.1 Programação por pares

Essa prática tem por objetivos colocar dois programadores em um único computador para escrever o código. As vantagens dadas por essa abordagem são:

- As decisões de projeto envolvem pelo menos duas pessoas;
- O código é sempre revisado por duas pessoas; e
- Pelo menos duas pessoas estão familiarizadas com cada parte do sistema.

Mensalmente, reunímamos-nos em duplas e realizávamos a programação em pares, além da revisão em dupla do código gerado individualmente.

Tivemos a percepção que a programação em dupla realmente trouxe uma maior eficiência na programação, já que o trabalho colaborativo facilita tanto na definição de métodos, lógicas, e resolução de problemas.

4.4.2 Testes

A metodologia XP define dois tipos de testes: testes de unidade e de aceitação. Os testes de unidade garantem o funcionamento das diversas partes do sistema, enquanto os testes de aceitação indicam quando o sistema está com as funcionalidades implantadas conforme especificado.

Os testes de unidade permitem que defeitos possam ser corrigidos antes de se desenvolver novas funcionalidades, além de garantir que o código está funcional.

Os testes de aceitação permitem que os clientes validem os casos de uso especificados, ajudando a medir quanto do sistema está pronto.

Os testes de unidade foram os mais utilizados, já que era necessário garantir que cada componente, desenvolvido separadamente, teria suas funcionalidades mantidas após a integração com os outros componentes do sistema.

Os testes de aceitação ocorreram com o sistema devidamente integrado, verificando-se o funcionamento do sistema como um todo.

4.4.3 Projeto simples

O XP possui a preocupação em manter o projeto simples, funcional, mudando-o somente quando ele não refletir a realidade. Para isso, deve-se manter o código sem duplicidades, com o menor número possível de classes e métodos, e sempre testados.

A simplicidade, no projeto, foi mantida durante a codificação, com a simplificação dos métodos, da modelagem e implementação das classes.

4.4.4 Componentização

O desenho da arquitetura de componentes do sistema é mostrado na Figura 4.2.

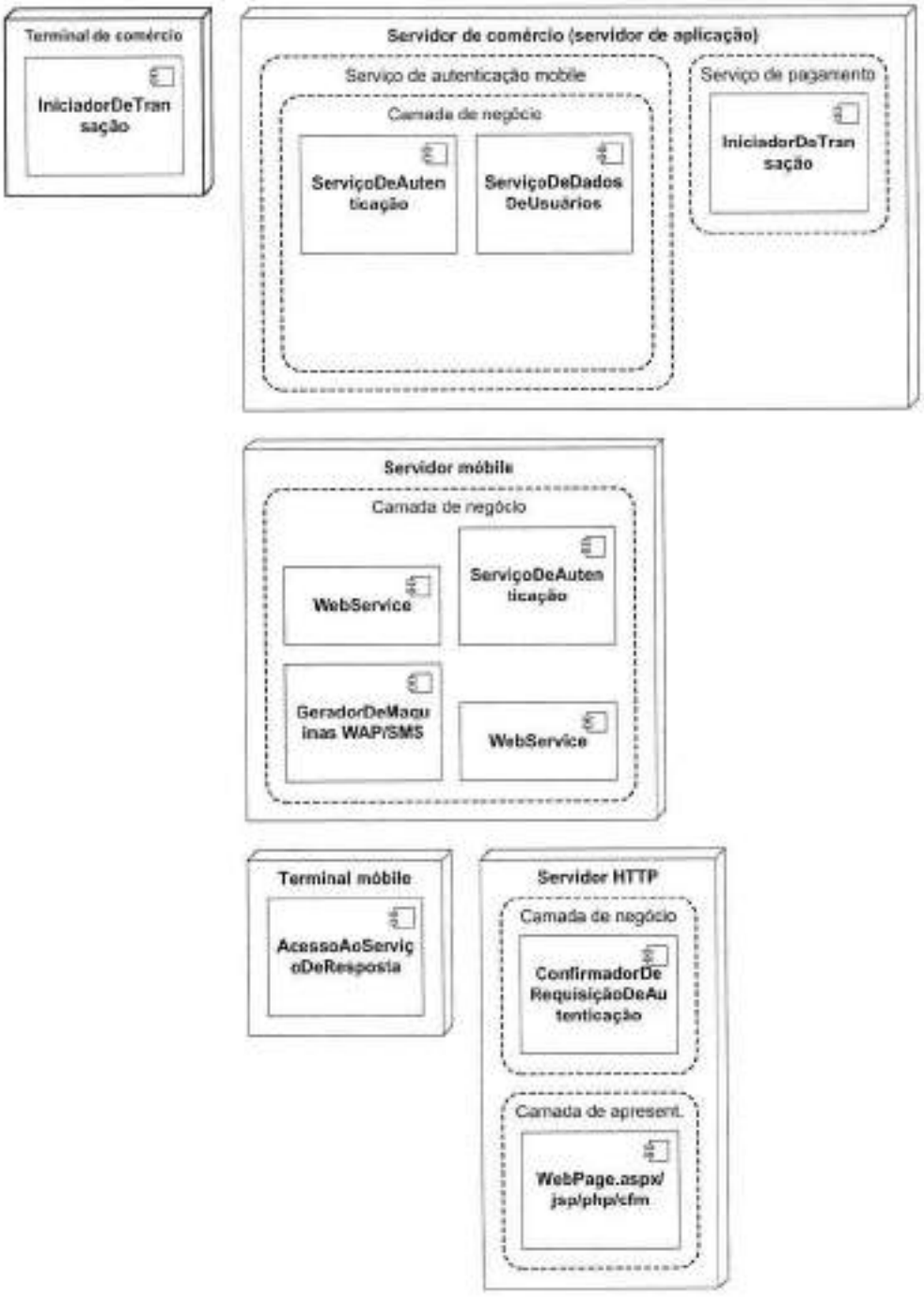

Figura 4.2 : Arquitetura de componentes.

O componente "Gerador de Máquinas de Estado" é um componente gerenciador de transações. Ele possui uma interface com o *web service* legado, para que receba as transações vindas desse componente. Com a transação, e todas as informações sobre os agentes de autorização, é capaz de inicializar as máquinas de estado responsáveis pelos estados das transações e gerenciá-las.

Possui uma interface com o componente de envio de mensagem Wap Push para o envio de mensagens assíncronas para os celulares, acessado pelas máquinas de estado através de um método desse componente.

As Máquinas de Estado, que também fazem parte desse componente, guardam os estados da transação. Para garantir a qualidade no serviço prestado pelo servidor, as máquinas de estado possuem um timer, utilizado para o *time-out* das transações. Desse modo, se a mensagem não é entregue ao destinatário por problemas na operadora, as máquinas de estado entram em estado de *time-out* e iniciam o envio da transação pendente a um outro celular responsável também por aprovar tal transação.

Na resposta do celular, o *web service* de reposta Wap acessa o componente Gerenciador de Máquinas para a entrega da mensagem. O gerenciador é então capaz de identificar qual a máquina responsável pela resposta, direcionando a resposta à máquina correta.

Os estados de uma transação dependem da ativação ou não da máquina ("DESATIVADO"/"ENVIADO"), erro na conexão ao *gateway* Wap ("ERRO"), envio de mensagem com sucesso até o *gateway* Wap ("CONECTADO GATEWAY"), em *time-out* ("TIME-OUT"), ou envio com sucesso ao celular, identificado com a resposta do celular ao sistema ("ENVIO SUCESSO").

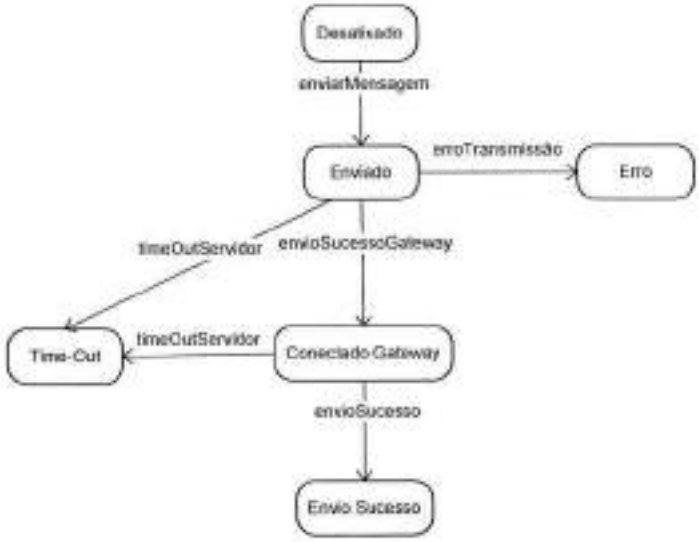

Figura 4.3: Diagrama de estados do componente Máquina de Estados.

4.4.5 Prototipação

Para entender os requisitos do sistema proposto, foi utilizada uma abordagem *top-down* do problema. A partir da especificação de aplicativos clientes do sistema, e da identificação de seus requisitos, pudemos identificar requisitos mais detalhados do nosso próprio sistema.

A abordagem de prototipação no desenvolvimento do projeto foi através do desenvolvimento de Provas de Conceito (POCs) que demonstrassem a validade das várias partes do sistema, além de dar maturidade aos requisitos do sistema.

Os POCs projetados foram feitos para validar a arquitetura de componentes, mostrada na Figura 4.2, e os casos de uso especificados no Apêndice A. Os POCs se basearam no desenvolvimento de componentes *dummies*, que implementam a lógica de negócios de modo simples, muitas vezes simulando transações e interfaces com outros componentes, garantindo que o foco no seu desenvolvimento seja a sua lógica de negócio. Preocupações com interfaces e comunicações entre componentes não são tratados em uma primeira fase de desenvolvimento do componente.

POC: Iniciar transação	
Componente	Gerenciador de Máquinas
Parâmetros de entrada	Mensagem com dados da transação e dos agentes de resposta
Especificação	O gerenciador de máquinas recebe a mensagem de inicio de transação.
Parâmetros de saída	Inicia a transação com a máquina de estados resultante da busca
POC: Receber Mensagem	
Componentes	Gerenciador de Máquinas e Máquina de Estado
Parâmetros de entrada	Resposta do celular
Especificação	O gerenciador de máquinas recebe a mensagem, e a direciona para a máquina de estados reponsável pela transação e resposta do celular
Parâmetros de saída	Timer da máquina de estados cancelada, e status da máquina atualizada

Tabela 4.1: Descrição dos POCs desenvolvidos.

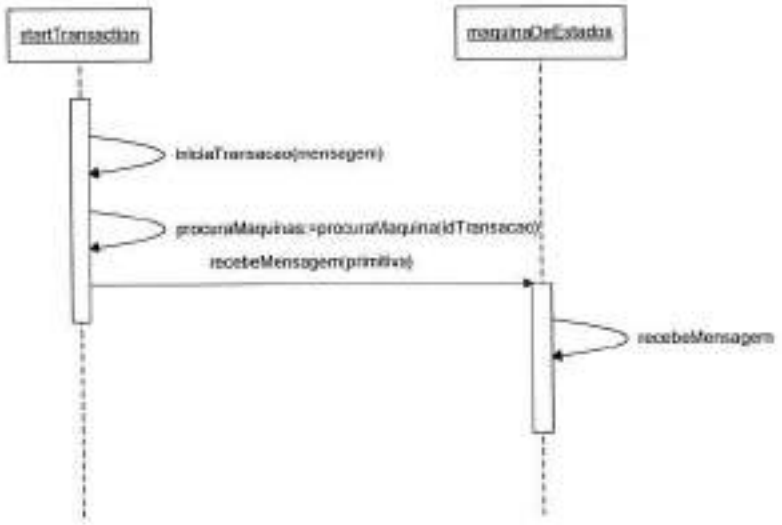

Figura 4.4: Diagrama de Seqüência do POC Iniciar transação.

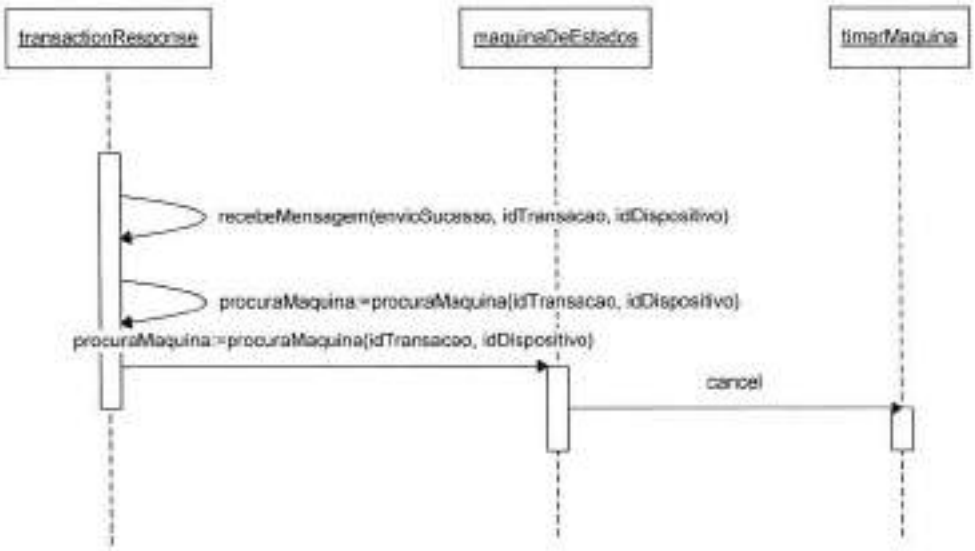

Figura 4.5: Diagrama de Seqüência do POC Receber mensagem.

5 Projeto e implementação

A estruturação dos pacotes de trabalho (Work Breakdown Structure (WBS)) do projeto de formatura pode ser observada na seguinte tabela:

Projeto	Especificação de software	Requisitos de Software	
		Arquitetura baseada em Componentes	
		Casos de Uso	
		Diagramas de Classes	
		Diagramas de Seqüência	
	Estudo sobre Tecnologias	Enterprise Java Beans	
		Message Beans	
		Web Services	
		O/R Mapping - Hibernate	
		Wap Push	
		Infra Estrutura	
	Desenvolvimento	Servidor	Modelagem da Base de Dados
			Componente de comunicação mobile (WAP)
			WebService de interface com aplicação
			WebService de interface com clientes mobile
			Componente de gerenciamento do serviço
			Componentes adicionais (log/O/R Mapping)
		Aplicação	Desenvolvimento da aplicação de exemplo
	Documentação	Documento de Software	
		Documentação de código	
	Testes	Ambiente simulado	
		Ambiente real	

Tabela 5.1: WBS do projeto

O projeto é formado por cinco grandes pacotes de trabalho, especificação de *software*, estudo sobre tecnologias, desenvolvimento, documentação e testes.

A especificação de *software* consiste na geração do documento de requisitos encontrado no Apêndice A. Seu principal objetivo é a identificação de forma clara e objetiva de todos os requisitos de software, além da criação de modelos que possibilitem a sua implementação. Primeiramente foram estabelecidas as funcionalidades que o servidor deveria fornecer às aplicações clientes e de que modo elas poderiam acessá-las. Após as definições, foi possível elaborar uma arquitetura baseada em componentes, enfatizando o desacoplamento, já que esta é a principal característica do projeto. A criação da arquitetura, em conjunto com os requisitos funcionais, possibilitou a geração dos documentos seguintes do processo de desenvolvimento de software, a saber: casos de uso, diagrama de classes e diagrama de seqüência.

O estudo sobre tecnologias foi necessário, pois não se sabia se elas forneceriam todas as funcionalidades necessárias para o desenvolvimento do projeto. Desse modo, foi necessário avaliar diversas ferramentas disponíveis no mercado antes do início da etapa de implementação do projeto. O estudo foi focado na plataforma Java, mais especificamente em Enterprise Java Beans, Java Messaging Services, *framework* de mapeamento de objetos e dados relacionais Hibernate e bibliotecas e simuladores Openwave Wap Push e Smarttrust para envio e recebimento de mensagens para dispositivos móveis. Um estudo sobre a infra-estrutura para a execução do projeto foi feito, pois era necessário saber qual sistema operacional, quais recursos computacionais, quais recursos de rede e quais servidores deveriam ser adotados para desenvolvimento e execução do projeto.

O desenvolvimento do projeto foi feito de forma distribuída, ou seja, cada integrante do grupo foi responsável por um componente. Inicialmente, foi realizado um prova de conceito para a validação da arquitetura proposta, sendo esta aprovada. Em seguida, foram feitos os componentes utilizando-se a tecnologia Enterprise Java Beans, mas problemas foram encontrados durante o desenvolvimento e a arquitetura do projeto foi modificada de forma a

solucionar esses problemas, conforme descrito no capítulo 2.2. O desenvolvimento do componente de envio de mensagens para dispositivos móveis também teve problemas, pois, apesar de existirem dois simuladores para recebimento de mensagens, apenas um funcionava corretamente, o Openwave Wap Push. Desse modo, o simulador da Smarttrust foi excluído do projeto. Os testes realizados sobre cada componente desenvolvido, tiveram a finalidade de comprovar o seu correto funcionamento.

A documentação gerada é composta de duas partes, uma do desenvolvimento do *software* e outra do código fonte. O documento de *software* é o mesmo que foi gerado no início do projeto, especificação de requisitos e modelagem do servidor, mas com algumas modificações feitas, já que existiram problemas durante o desenvolvimento, sendo que foram solucionados por meio e alterações na modelagem. A documentação de código foi gerada conforme os componentes estavam sendo desenvolvidos, ou seja, essa documentação explica qual a finalidade de cada classe e de cada método criado.

Os testes realizados tiveram como objetivo a validação dos requisitos propostos inicialmente no projeto. Os testes em ambiente simulado possibilitaram a validação dos requisitos. Os testes em ambiente real não foram realizados.

5.1 Cronograma

O cronograma elaborado e apresentado no arquivo <Cronograma.mpp> (contido no CD-ROM que acompanha a monografia)[1] corresponde às fases de desenvolvimento, documentação e testes. A duração total dessas três fases foi aumentada em uma semana, devido aos problemas encontrados durante a fase de desenvolvimento, já que a modelagem inicial teve que ser alterada.

[1] ARQUIVO DIGITAL DO PROJETO DE FORMATURA
2006, 1 CD-ROM

5.2 Riscos

São dois os principais riscos do projeto: a utilização de componentes externos para envio de mensagens *push* e os testes em ambiente real, ou seja, testes utilizando dispositivos móveis celulares ao invés de simuladores. A utilização de componentes externos para envio de mensagens é o principal risco do projeto, pois o servidor depende desse componente para o envio das requisições para dispositivos móveis. Além da utilização do componente externo, existe a necessidade de um *gateway* para a utilização do Wap Push, ou seja, o risco é muito alto, pois ele pode não funcionar e as mensagens nunca chegarão ao seu destinatário. O risco em testes em ambiente real está relacionado à possibilidade de se alterar o código fonte pronto para poder se adequar a esse ambiente. Como não foram feitos testes em ambiente real, esse risco não foi considerado.

6 Considerações finais

O desenvolvimento do projeto de formatura proporcionou uma oportunidade ímpar de pesquisa e desenvolvimento de um produto que não só tem aplicações acadêmicas como aplicações sociais e comerciais. Além do conhecimento reunido no processo de pesquisa estar disponível para outros projetos, o grupo teve sucesso ao identificar oportunidades de estudo a fim de complementar o ***Servidor***.

Em geral, grupo avalia que teve uma experiência satisfatória e pôde, ao longo destes dois semestres, reunir e aplicar os conhecimentos adquiridos ao longo da graduação, em especial os conhecimentos de redes e engenharia de *software*. Disciplinas relacionadas à modelagem de sistemas, modelagem de bancos de dados e gerenciamento de projetos, bem como os Laboratórios de Engenharia de Software com tópicos de metodologias e estudo de *design patterns* fizeram uma diferença crucial no planejamento e execução do projeto.

O grupo reconhece, por outro lado, que alguns objetivos não puderam ser atingidos, a saber: a construção de um sistema de relatórios dinâmico acoplado ao banco de dados de transação, a execução de testes de carga e observação de testes de balanceamento de carga e a construção de um sistema de autenticação integrado à operação dos *web services* segundo especificações de segurança.

6.1 Sugestões de estudo continuado

A partir da identificação dos objetivos não alcançados pelo projeto de formatura, mas principalmente a partir da identificação de oportunidades de estudo, elaborou-se uma lista de tópicos que podem ser explorados como estudo continuado, a saber:

- **Criação de uma implementação alternativa para acesso ao banco de dados:** ao utilizar-se o Hibernate como ferramenta de mapeamento entre objetos e dados relacionais, introduz-se uma carga adicional ao sistema. É interessante elaborar um estudo que compare esta implementação com alternativas como o acesso através de *stored procedures* ou outros *design patterns* de interesse;

- **Criação de sistemas de segurança (autenticação) utilizando-se a própria especificação dos *web services*:** a especificação de *web services* conta com várias especificações complementares. É o caso da especificação WS-Security, que propõe autenticar os atores de uma comunicação com um *web service*.

- **Realização de testes de carga:** a realização de testes de carga permitirá comprovar o potencial de escalabilidade do sistema e, comprovar se o sistema, da forma que foi implementado, é viável se operado em um ambiente de produção. Além dos testes com a implementação atual, recomenda-se a realização de testes em um ambiente distribuído (*cluster* de servidores) ou até mesmo com *containers* de outros fabricantes como IBM, SAP, BEA e RedHat. No caso de sistemas distribuídos, estudar a problemática inerente à sincronização de tempo ao longo de servidores conectados através de uma rede. Deve-se levar em consideração a impossibilidade de se ter um sincronismo absoluto entre relógios, levantar o pior caso e alterar a lógica de negócios da aplicação para acomodar esta limitação da arquitetura, isto é, responder como os mecanismos de *time-out* devem ser alterados para aceitar um intervalo pequeno, porém significativo, de imprecisão temporal.

- **Portar o sistema para outra plataforma:** os principais componentes do sistema estão presentes nas principais plataformas utilizadas comercialmente. Um estudo interessante envolve portar toda a implementação para uma plataforma alternativa, como por exemplo o Microsoft .NET Framework, e levantar medidas de desempenho e outras métricas de interesse que comparem ambas as arquiteturas. Explorar outros bancos de dados de fabricantes como IBM, Oracle

- **Criar relatórios dinâmicos:** utilizar o módulo Report Services do Microsoft SQL Server 2005 a fim de criar relatórios dinâmicos com informações de avaliação de tempo de resposta, taxa de sucessos e falhas e outras métricas de interesse.

Referências

WEB SERVICE, Wikipedia, the free encyclopedia
Disponível em: <http://en.wikipedia.org/wiki/Web_service>.
Acessado em: 03 de dezembro de 2006.

JAVA TECHNOLOGY AND WEB SERVICES
Disponível em: <http://java.sun.com/webservices/>.
Acessado em: 03 de dezembro de 2006.

APACHE WEB SERVICES PROJECT
Disponível em: <http://ws.apache.org/>.
Acessado em: 03 de dezembro de 2006.

WEB SERVICES AND THE MICROSOFT PLATFORM
Disponível em: <http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnwebsrv/html/wsmsplatform.asp>.
Acessado em: 03 de dezembro de 2006.

A WEB SERVICES PRIMER, webservices.xml.com
Disponível em:
<http://webservices.xml.com/pub/a/ws/2001/04/04/webservices/index.html?page=2>.
Acessado em: 03 de dezembro de 2006.

MESSAGE ORIENTED MIDDLEWARE, Wikipedia, the free encyclopedia
Disponível em: <http://en.wikipedia.org/wiki/Message_Oriented_Middleware>.
Acessado em 03 de dezembro de 2006.

THE JAVA EE 3 TUTORIAL
Disponível em: <http://java.sun.com/javaee/5/docs/tutorial/doc/>.
Acesseado em: 03 de dezembro de 2006.

JAVA EE 5 SDK
Disponível em: <http://java.sun.com/javaee/5/docs/api/index.html>.
Acesseado em: 03 de dezembro de 2006.

OBJECT-RELATIONAL MAPPING, Wikipedia, the free encyclopedia
Disponível em: <http://en.wikipedia.org/wiki/Object-relational_mapping>.
Acessado em: 03 de dezembro de 2006.

HIBERNATE
Disponível em: <http://www.hibernate.org>.
Acesseado em: 03 de dezembro de 2006.

INVERSION OF CONTROL CONTAINERS AND THE DEPENDENCY INJECTION PATTERN, Martin Fowler
Disponível em: <http://www.martinfowler.com/articles/injection.html>.
Acesseado em: 03 de dezembro de 2006.

PUSH ARCHITECTURAL OVERVIEW
Disponível em:
<http://www.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-250-pusharchoverview-20010703-a.pdf>.
Acessado em: 03 de dezembro de 2006.

WAP SERVICE INDICATION SPECIFICATION
Disponível em:
<http://www.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-167-serviceind-20010731-a.pdf?doc=wap-167-serviceind-20010731-a.pdf>.
Acessado em: 03 de dezembro de 2006.

WAP SERVICE INDICATION SPECIFICATION
Disponível em:
<http://www.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-167-serviceind-20010731-a.pdf?doc=wap-167-serviceind-20010731-a.pdf>.
Acessado em: 03 de dezembro de 2006.

DOCUMENTO DTD
Disponível em: <http://www.openmobilealliance.org/tech/DTD/pap_2.1.dtd>.
Acessado em: 03 dezembro de 2006.

DOCUMENTO DTD
Disponível em: <http://www.openmobilealliance.org/tech/DTD/si_1.0.dtd>.
Acessado em: 03 de dezembro de 2006.

DOCUMENTO DTD
Disponível em: <http://www.openmobilealliance.org/tech/DTD/sl_1.0.dtd>.
Acessado em: 03 de dezembro de 2006.

PUSH ACCESS PROTOCOL SPECIFICATION
Diponível em:
<http://www.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-247-pap-20010429-a.pdf>.
Acessado em: 03 de dezembro de 2006.

PUSH PROXY GATEWAY SERVICE SPECIFICATION
Diponível em:
<http://www.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-249-ppgservice-20010713-a.pdf>.
Acessado em: 03 de dezembro de 2006.

PUSH OTA PROTOCOL SPECIFICATION
Diponível em:
<http://www.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/wap-235-pushota-20010425-a.pdf>.
Acessado em: 03 de dezembro de 2006.

WAP PUSH LIBRARY
Disponível em: < http://developer.openwave.com/docs/wappushjava10/developer_guide.pdf >.
Acessado em: 03 de dezembro de 2006.

WAP GATEWAY
Disponível em:
<http://developer.openwave.com/dvl/tools_and_sdk/test_servers/wap_gateway/devgate2.htm >.
Acessado em: 03 de dezembro de 2006.

[Pressman] Pressman, R. S. Engenharia de Software; McGraw-Hill, 2000

[Sametinger] Sametinger, J. Software Engineering with Reusable Components, Springer-

[Szyperski] Szypersky, C. Component Software: Beyond Object-Oriented Programming. Addison Wesley, 2002

[XP] Extreme Programming: <http://www.extremeprogramming.org/index.html>

[Yourdon] Yourdon, E. Análise Estruturada Moderna, Editora Campus, 1990

Verlag, 1997.

Lista de siglas

EJB	Enterprise Java Bean
HTTP	HyperText Transfer Protocol
IP	Internet Protocol
J2EE	Java 2 Enterprise Edition
JMS	Java Messaging System
MDB	Message Driven Bean
MOM	Message-Oriented Middleware
RPC	Remote Procedure Call
SOA	Services Oriented Architecture
TIC	Tecnologia de Informação e Comunicações
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WSP	Wireless Session Protocol
WWW	World Wide Web
XML	eXtensible Markup Language

APÊNDICE A – Especificação de Requisitos de Software

1 Objetivo do Documento

O objetivo do documento de Especificação de Requisitos do Software é identificar de forma completa e clara todos os requisitos a serem atendidos pelo projeto de formatura a desenvolvido pelo grupo ao longo do ano.

Nele, são referenciados modelos de software do sistema, desenvolvidos e aperfeiçoados durante todo o projeto.

2 Objetivo do Sistema

2.1 Nome do Sistema

Servidor para validação assíncrona por dispositivos móveis

2.2 Escopo

Para o Projeto de Formatura propõe-se construir um servidor para aplicações de validação assíncrona por dispositivos *mobile* (envolvendo celulares).

A proposta apresentada para o Projeto de Formatura justifica-se a partir da observação do modelo de desenvolvimento de aplicações para dispositivos móveis atuais. Atualmente, há uma crescente tendência de cada vez mais agregar valor ao dispositivo, tornando-o capaz de responder a serviços complexos e que passam a envolver privacidade e segurança. As soluções comumente encontradas confiam ao dispositivo móvel a tarefa de guardar as chaves de acesso a esses serviços, aumentando assim sua importância elevando a gravidade de suas vulnerabilidades.

A proposta, por outro lado, aposta na manutenção das chaves de acesso ao serviço com o usuário, que por sua vez utiliza o dispositivo móvel unicamente como ferramenta de interação com os provedores. Além disso, a construção de um servidor provendo tal serviço permite a exploração de possíveis aplicações deste modelo de forma mais rápida, pois os detalhes da implementação relacionados com a conectividade entre os sistemas descritos estariam prontos.

Dada a componentização da arquitetura proposta, pode-se ainda falar na expansão da infra-estrutura para outras aplicações, isto é, uma vez que se tem uma arquitetura modular e bem documentada, pode-se por ou tirar partes do todo a fim de agregar funcionalidades.

Por fim, o projeto explora a aplicação das técnicas para um fim social. Um exemplo é o próprio exemplo de aplicação proposto, que tem a intenção de auxiliar pessoas com necessidades especiais a acessar serviços que requerem segurança, transferindo a responsabilidade da autorização para quem pode responder por ela, ou seja, os responsáveis. No caso do sistema de pagamento, o cliente com necessidades especiais pode ser desde menores de idade até idosos, que não tem condições de memorizar senhas ou entender os processos de pagamento.

Em suma, a proposta do Projeto de Formatura tenta unir os conhecimentos adquiridos na Universidade com as necessidades tanto de mercado (empresariais) quanto sociais do mundo não-acadêmico.

3 Descrição Geral

3.1 Perspectivas do Produto

Conforme Figura 1, tem-se no sistema de pagamentos clássico um usuário cliente fazendo uma solicitação de serviço financeiro mediante apresentação de autorização eletrônica ou não (respectivamente, senha ou assinatura). Munido de dados como o valor da transação, identificação do cliente e, eventualmente, de autorização eletrônica, o terminal comunica-se com um servidor de aplicação comercial. Este, por sua vez, é responsável em fazer a interface com o servidor *backoffice* da instituição comercial (banco ou operador de cartão de crédito). O servidor *backoffice* e sua base de dados são sistemas legados e não é importante saber sua implementação dada a componentização desta arquitetura. É importante apenas saber suas especificações de entrada e saída.

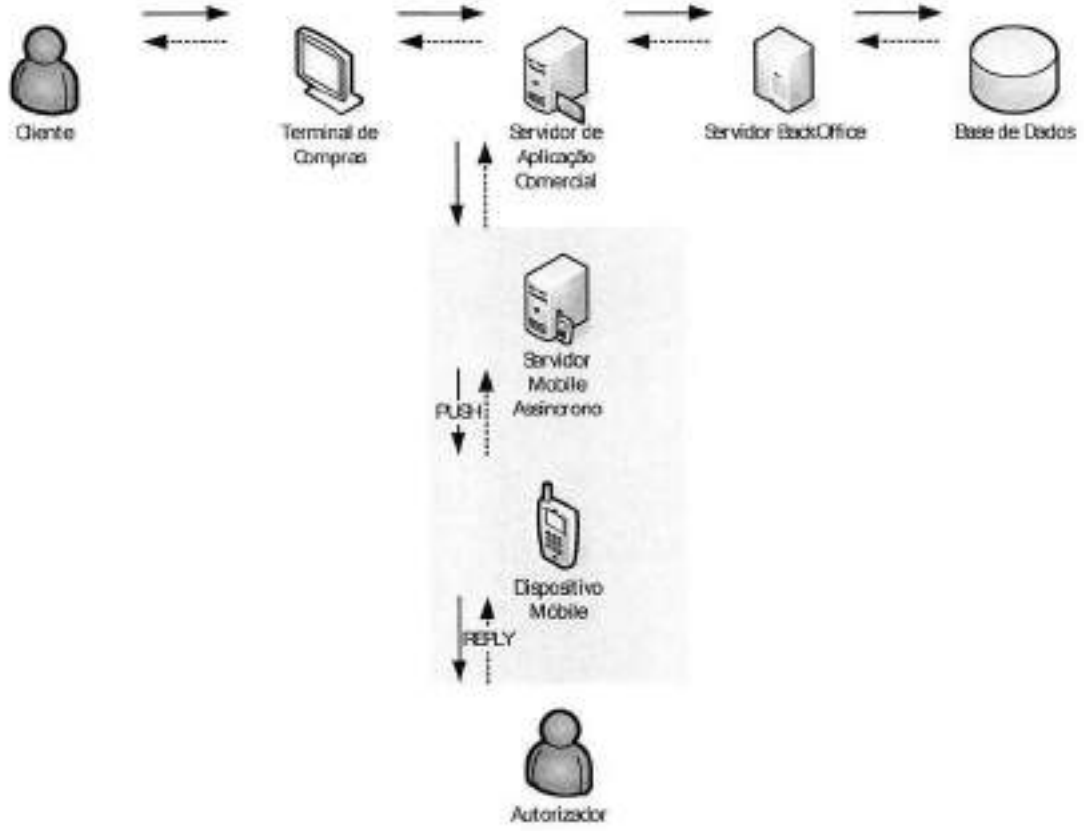

Figura 1: Transação de autorização

O servidor de aplicação comercial, por outro lado, deve ter sua implementação aberta. Daí surge a possibilidade de modificá-lo para incluir uma nova forma de autenticação de transação: a **requisição de autorização assíncrona e remota por dispositivo móvel**. Modificando-se o servidor de aplicação comercial, pode-se incluir um serviço que recebe a identificação do cliente e o valor da transação e que se comunica com um servidor *mobile* assíncrono (novo sistema), requisitando dele a autenticação. Caso ela seja aprovada, o servidor de aplicação comercial retoma a operação normal, conversando em seguida com o servidor *backoffice*.

O pedido de autenticação é realizado na forma de uma mensagem assíncrona e segura para celulares de usuários cadastrados. A operação deve ocorrer em tempo real, de forma que a mensagem possui um tempo de validade e é encaminhada para usuários secundários caso o primário não esteja disponível. Estes requisitos estão de acordo com a especificação WAP 2.0. A área sombreada da Figura 1 é na realidade uma visualização condensada de um sistema maior. Sem identificar nós específicos do sistema de telefonia celular, poderíamos expandir a área em questão da Figura 1 de acordo com a Figura 2, desta forma explicitando a implementação do servidor *mobile* sobre a plataforma da especificação WAP 2.0.

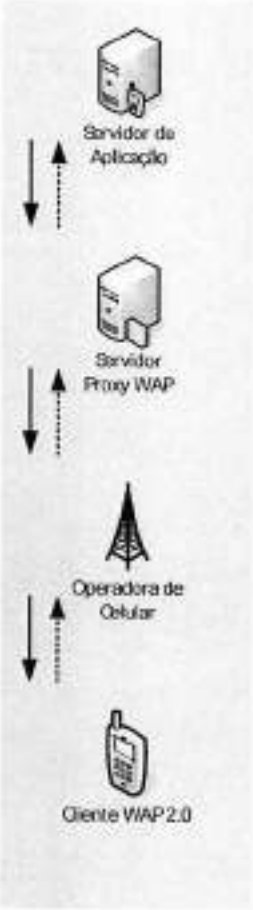

Figura 2: Implementação do servidor *mobile* sobre a plataforma WAP

3.1.1 Interface com o Sistema

A interface com os sistemas externos será realizado através de web services

3.1.2 Interfaces de Usuário

A interface com o cliente móvel se dá através de um servidor de aplicação, que disponibiliza uma página web para a visualização dos dados da transação a ser aprovada.

O acesso do usuário com dispositivo móvel é feito por meio de uma URI contida na mensagem assíncrona recebida. Desse modo, esse acesso é facilitado, pois o usuário não precisa digitar a URI no dispositivo móvel.

3.1.3 Interfaces de Hardware

A necessidade de interface de hardware será necessário caso haja a disponibilização de um ambiente real para envio de mensagens para celulares.

3.1.4 Interfaces com Software

Abaixo, são listadas as tecnologias utilizadas no desenvolvimento do sistema.

3.1.4.1 Linguagem de desenvolvimento: Java J2EE 5.0

3.1.4.1.1 Enterprise Java Beans

3.1.4.1.2 Message Beans

3.1.4.1.3 Web Services

3.1.4.2 *Framework* para mapeamento de Banco de Dados relacionais em classe: Hibernate

3.1.4.3 Banco de Dados: SQL Server

3.1.4.4 Ambiente de Desenvolvimento: NetBeans

3.1.4.5 Tecnologia de Envio de mensagens WAP Push: Openwave

3.1.4.5.1 Openwave Wap Push Library Java 1.0

3.1.4.5.2 Openwave V7 Simulator

3.1.4.5.3 Mobile Access Gateway

3.1.5 Interfaces de Comunicação

Para a comunicação com os dispositivos móveis, é utilizada a tecnologia WAP.

O envio de mensagens assíncronas para dispositivos móveis utiliza os protocolos *Push Over-The-Air* e *Push Access Protocol*.

3.2 Funções do Software

3.2.1 O sistema deve receber o início de uma transação e salvar todos os dados da transação recebidos pelo aplicativo legado

3.2.2 O sistema deve ser capaz de gerenciar várias transações simultâneas

3.2.3 O sistema deve ser capaz de enviar mensagens para os dispositivos móveis

3.2.4 Deverá disponibilizar uma página para acesso pelos usuários do dispositivo móvel, que conterá os dados da transação a ser aprovada

3.2.5 Após cada passo da transação, o sistema deve atualizar o estado da transação

3.2.6 O sistema deverá fornecer uma interface para a verificação do status da transação

3.2.7 Garantia de entrega: O servidor deve possuir mecanismos para garantir a entrega das mensagens aos dispositivos móveis.

3.3 Características dos Usuários

Os usuários do sistema são aplicações que buscam prover o serviço de autenticação das suas transações através de dispositivos móveis.

São aplicações já estão disponíveis no mercado (aplicações legado), que procuram dar uma maior abrangência aos seus serviços. Aplicações de transações bancárias, crédito, que necessitam atualmente de autorizações mediante uso de senhas ou assinatura, podem, com a utilização dos serviços do nosso servidor, requisitar a autorização da transação de um terceiro (devidamente cadastrado), utilizando um dispositivo móvel.

Além disso, pode ser utilizado para o desenvolvimento de novas aplicações, que necessitem de tal serviço. Como exemplo, podemos citar:

- Confirmação de emissão de receita médica: No momento da venda de um medicamento, a farmácia envia uma confirmação para o celular médico que o receitou. Essa prática diminuiria o número de vendas indevidas de medicamentos.

- Autorização de entrada de menores de idade em espetáculos: Pais podem autorizar a entrada de seus filhos em espetáculos ou outros eventos através de seus celulares.

- Confirmação de pagamentos de alto valor: Assim como hoje os bancos utilizam a ligação telefônica para confirmar transações de valores muito altos, que não é o perfil do correntista, esse sistema poderia ser utilizado para tal confirmação através dos dispositivos móveis. Isso garantiria uma maior automação desse processo, deixando de depender de atendentes para efetuar a ligação e confirmação da transação.

3.3.1 Criticalidade da aplicação

Por poder ser um serviço de Aplicações bancárias e de crédito são aplicações de alta criticidade, o sistema deve prover segurança dos dados nas transações.

3.3.2 Limitações

O teste com aparelhos celulares só será possível caso haja a disponibilidade de um ambiente real de conexão do servidor a um *gateway* WAP da operadora. Além disso, somente celulares com o *browser* desenvolvido pela Openwave recebem as mensagens enviadas pelo sistema.

Além disso, o tempo de resposta do sistema é muito dependente do sistema de telefonia móvel.

3.4 Hipóteses e Dependências

O projeto é dependente da disponibilidade do *gateway* WAP da *OpenWave*, *gateway* através do qual testamos o envio de mensagens WAP, e pelo qual poderia ser feito a conexão aos celulares reais, caso estes fossem configurados com essa funcionalidade.

4 Requisitos Específicos

4.1 Modelo de Casos de Uso

4.1.1 Diagrama de Casos de Uso

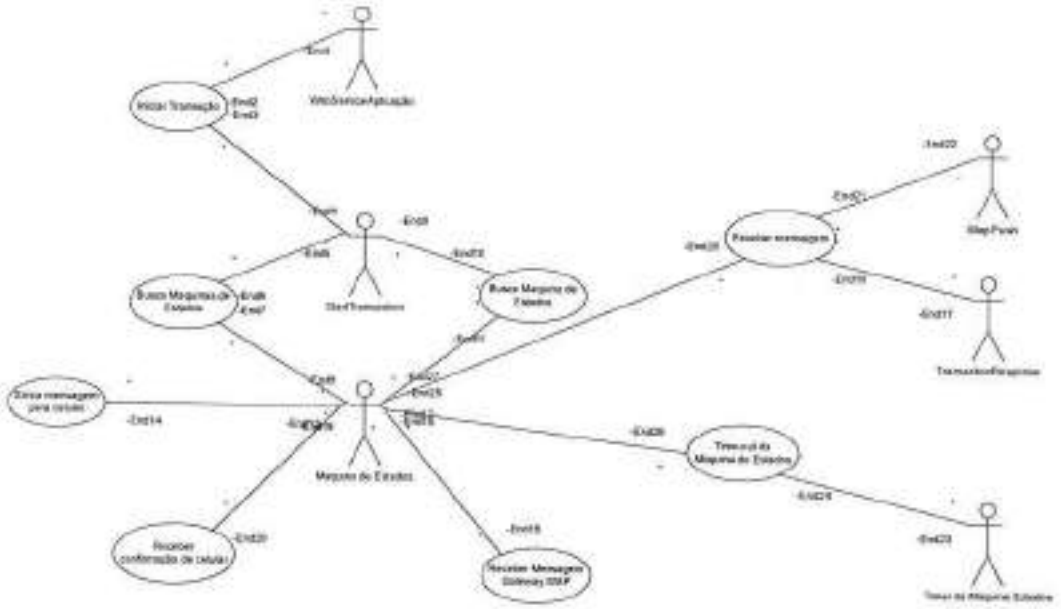

Figura 3: Diagrama de Casos de Uso

4.1.2 Descrição dos Atores

- WebService Aplicação: Interface entre aplicação e o sistema. Através dele as transações são iniciadas.

- Gerenciador de Máquinas (StartTransaction e TransactionResponse): Componente responsável pelo gerenciamento das máquinas de estados, responsável pela sua inicialização (StartTransaction) e pelo seu gerenciamento, com a entrega das respostas para a máquina de estados correta (TransactionResponse). Por motivos de implementação, no diagrama de casos de uso eles são separados, mas ambos fazem o gerenciamento das máquinas de estado.

- Máquinas de Estados: Possui os estados da transação. Possuem um temporizador associado a cada transação, para garantir que o servidor não fique esperando uma mensagem atrasada do celular.

- Wap Push: Interface com o servidor Web para os dispositivos móveis.

4.1.3 Descrição dos casos de Uso

4.1.3.1 Iniciar transação

Descrição: Esse caso de uso descreve o inicio de uma transação, com o Gerenciador de Maquinas de Estado iniciando uma nova máquina de estados

Evento Iniciador: Envio de uma requisição feita pelo WebService Legado

Atores: Webservice Aplicação (WSA)

Gerenciador de Máquina de Estados (GME)

Máquina de Estados

Pré-condição: Gerenciador de máquina de estados inicializado

Seqüência de eventos:

1. WSA envia para o GME a requisição de inicio de transição, enviando os seguintes dados: Os agentes de resposta, cada um com a sua coleção de máquinas de estado, o agente da requisição.

2. O GME recebe a transação e salva todos os dados referentes a essa transação no banco de dados

3. GME seleciona uma das máquinas de estado dessa transação e a ativa

4. A máquina de Estados formata os dados (usuário, valor, estabelecimento) em formato padrão para envio para celular

5. Máquina de Estados envia a mensagem para o celular, muda seu estado de DESATIVADO para ENVIADO, inicia o seu timer e salva o valor da hora atual (hora de início)

Pós-condição: Máquina no estado ENVIADO, com timer ativado, e mensagem enviada para o celular

Extensões:

a. A máquina de estados já existe (passo c): O Gerenciador de Máquinas de Estados verifica o estado na máquina existente.

Inclusões:

a. Busca Máquina de Estado (passo 3)

b. Recebe mensagem (passo 5)

c. Receber mensagem do *Gateway* WAP (passo 5)

4.1.3.2 Busca Máquinas de Estado

Descrição: Esse caso de uso descreve o processo de busca de máquinas de estados através da identificação da transação

Evento Iniciador: Gerenciador inicia uma nova busca

Atores: Gerenciador de Máquina de Estados (GME)

Máquina de Estados

Pré-condição: Conhecer a identificação da transação procurada

Seqüência de eventos:

GME faz a busca, dentre todas as máquinas existentes por aquelas que possuam a identificação da transação igual ao buscado

Pós-condição: Coleção de máquinas de estado com identificação da transação procurada

Extensões:

Máquina de estados não existe: É retornada uma coleção de Máquina de Estados nula (passo 1)

Inclusões: Não há

4.1.3.3 Busca Máquina de Estado

Descrição: Esse caso de uso descreve o processo de busca de uma máquina de estados através da identificação da transação e da identificação do dispositivo do Agente de Resposta

Evento Iniciador: Gerenciador inicia uma nova

Atores: Gerenciador de Máquina de Estados

Pré-condição: Conhecer a identificação da transação procurada e a identificação do dispositivo do agente de resposta

Seqüência de eventos:

GME faz a busca, dentre todas as máquinas existentes, por aquela que possui a identificação da transação igual à procurada, recebendo uma coleção de Máquinas de Estados

Dentre a coleção de máquinas, o GME procura aquela que possui a identificação do dispositivo igual a procurada

Pós-condição: Uma máquina de estados, unicamente identificada

Extensões: Não se aplica

Inclusões:

Busca Maquinas de Estados (passo 1)

4.1.3.4 *Time-out* da Máquina de Estados

Descrição: Esse caso de uso descreve o processo de time-out de uma máquina de estados.

Evento Iniciador: Timer indicando time-out

Atores: Timer da Máquina de Estados

Máquina de Estados

Pré-condição: Máquina de Estados no estado CONECTADO ou ENVIADO e timer ativado

Seqüência de eventos:

O timer, após o tempo de time-out configurado no servidor Mobile, envia a mensagem de time-out para a sua Máquina de Estados

A Máquina de Estados muda seu estado para TIMEOUT e pára o seu timer

A Máquina de Estados salva a hora atual (hora time-out)

O sistema faz uma busca por uma nova máquina de estado, da mesma transação da máquina que deu time-out

Essa máquina é iniciada, e é enviada uma nova mensagem para o seu Agente de Resposta

Pós-condição: Máquina de Estados no estado TIMEOUT uma nova mensagem enviada a um novo Agente de Resposta acionado

Extensões: Não se aplica

Inclusões:

Busca Máquinas de Estados (passo 4)

Envia mensagem para celular (passo 5)

4.1.3.5 Receber mensagem do *Gateway* WAP

Descrição: Esse caso de uso descreve o processo de recebimento de uma mensagem do gateway WAP após o envio da mensagem ao celular

Evento Iniciador: Recebimento de uma mensagem pelo Gerenciador

Atores: Wap Push

Máquina de Estados

Pré-condição: Recebimento de mensagem envioSucessogateway do gateway WAP

Seqüência de eventos:

A máquina de estados, no estado ENVIADO, atualiza seu estado para CONECTADO_GATEWAY

A Máquina de Estados recebe essa mensagem, atualiza seu estado atual

Pós-condição: Máquina de Estados com estado atualizado para CONECTADO_GATEWAY

Extensões:

A mensagem recebida é mensagem de erro na transmissão wap, erroTransmissao (passo 1): A Máquina de estados seta seu estado atual como ERRO, finaliza o seu timer e inicia uma nova transação com uma nova máquina de estado, com a identificação de transação igual a sua.

Inclusões:

Busca Máquina de Estados (passo 1)

4.1.3.6 Receber confirmação do celular

Descrição: Esse caso de uso descreve o processo de recebimento de uma mensagem do agente de resposta, através de seu dispositivo

Evento Iniciador: Recebimento de uma mensagem

Atores: Wap Push

Gerenciador de Máquinas de Estados

Pré-condição: Recebimento de mensagem envioSucesso do Wap Push

Seqüência de eventos:

O gerenciador busca pela máquina responsável pela resposta da transação que está sendo respondida

A máquina de estados, no estado CONECTADO_GATEWAY, atualiza seu estado atual para SUCESSO e finaliza o seu timer

Pós-condição: Máquina de Estados com estado atualizado para SUCESSO e transação finalizada com sucesso

Extensões:

Inclusões:

Busca Máquina de Estados (passo 1)

Receber mensagem (passo 2)

4.1.3.7 Autenticação de usuário

Descrição: Este caso de uso descreve a autenticação de um usuário.

Evento iniciador: acesso à página de autenticação por meio da URL fornecida pela mensagem Push.

Atores: usuário

Pré-condição: recebimento de uma mensagem com uma URL para acesso à página de autenticação. Essa mensagem possui um identificador pessoal.

Seqüência de eventos:

Usuário insere sua senha;

Sistema valida os dados de entrada;

Sistema exibe a informação a ser validada.

Pós-condição: usuário tem acesso às informações necessárias para validação.

Extensão:

Dados de entrada inválidos (passo 2): sistema retorna mensagem de erro.

Inclusão:

Validação de requisições (passo 3)

4.1.3.8 Validação de requisições

Descrição: Este caso de uso descreve a validação de requisições recebidas

Evento iniciador: recebimento de uma mensagem com uma URL para acesso à página de autenticação.

Atores: usuário

Pré-condição: usuário autenticado no sistema.

Seqüência de eventos:

Sistema exibe informações referentes à requisição (nome do responsável pela requisição, localização, horário, valor);

Usuário responde a essa requisição.

Pós-condição: requisição respondida e enviada ao servidor

Extensão:

Não se aplica

Inclusão:

Não se aplica

4.2 Modelo de Classes

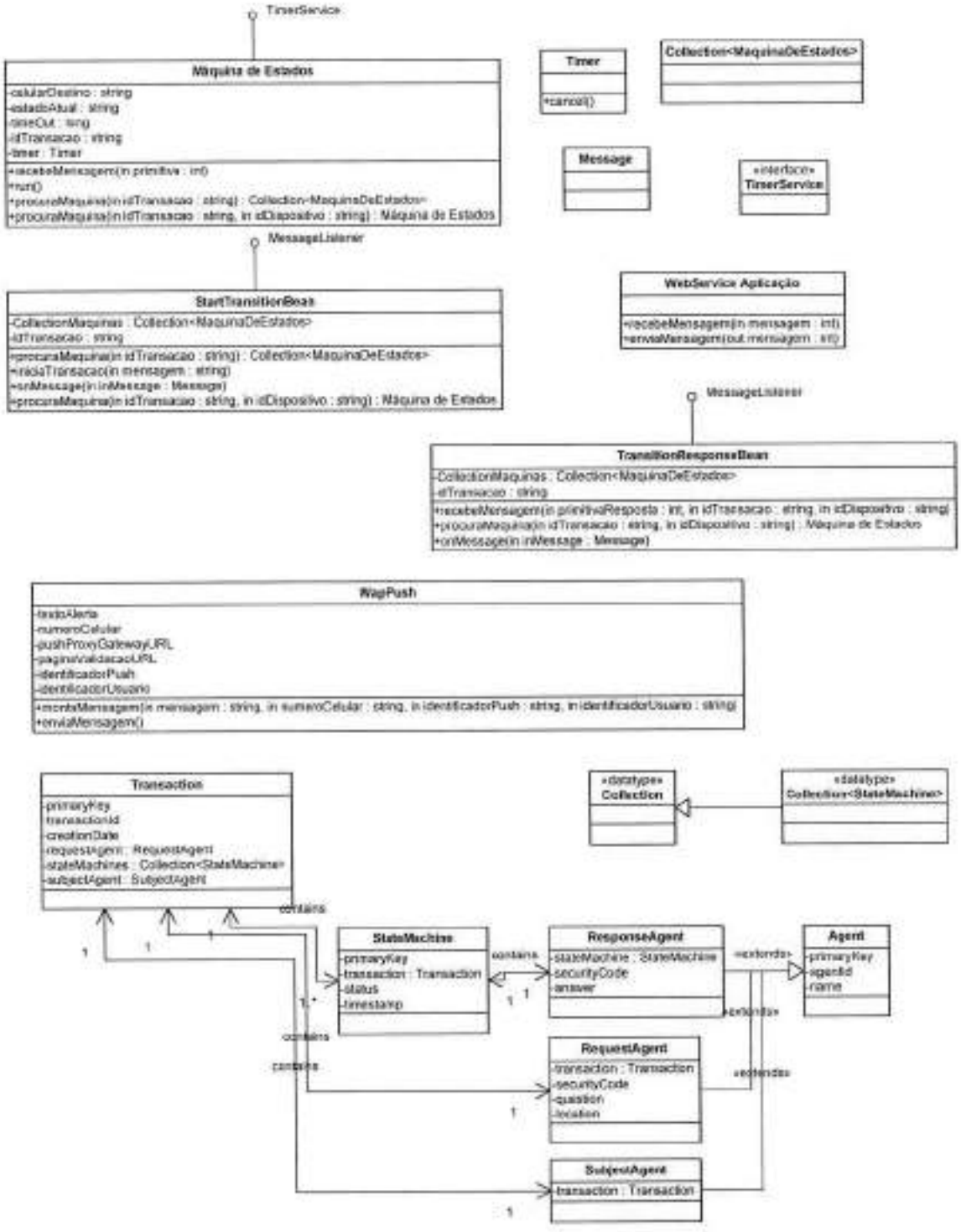

Figura 4: Diagrama de Classes

4.3 Modelo de Estados

A Figura 5 mostra os estados da máquina de estados durante uma transação, e quais as mensagens a máquina recebe para transitar entre os estados.

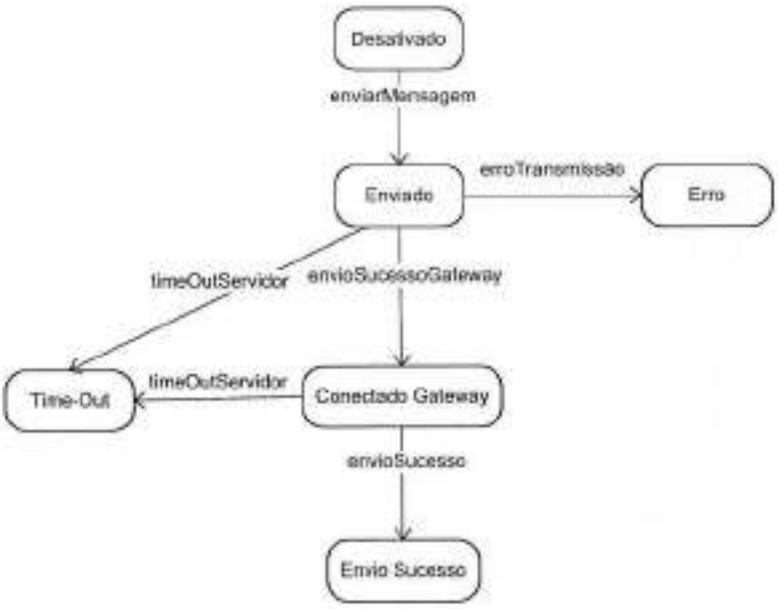

Figura 5: Diagrama de estados da máquina de estados

4.4 Requisitos de Bancos de Dados

Para que os vários componentes tenham acesso aos dados por eles compartilhados, há uma camada de persistência temporária formada por um banco de dados e uma biblioteca de mapeamento dos objetos. A camada de persistência definitiva armazena o resultado final das instâncias da aplicação.

Para cada operação e conjunto de operações (que eventualmente formam uma transação), são armazenados os resultados. O nível de detalhe destes relatórios e a forma de armazenamento podem ser ajustados.

5 Requisitos Não Funcionais

- Segurança de acesso: O sistema deve exigir senhas de acesso ao responsável pela autorização da transação, antes de exibir os dados da transação.

- Usabilidade: As interfaces do sistema devem ser simples, para a fácil integração com o sistema legado.

- Disponibilidade: O sistema deve estar disponível 24 horas por dia, 7 dias da semana.

- Desempenho: O sistema deve possuir um tempo de temporizadores para cada conexão com o celular. Isso deve ser feito para garantir a qualidade de serviço, diminuindo os efeitos de atrasos e incertezas de entrega de mensagem por parte da operadora de celulares, e permitindo que o sistema possua tempos de resposta determinísticos.

6 Apêndices

6.1 Diagramas de Seqüência

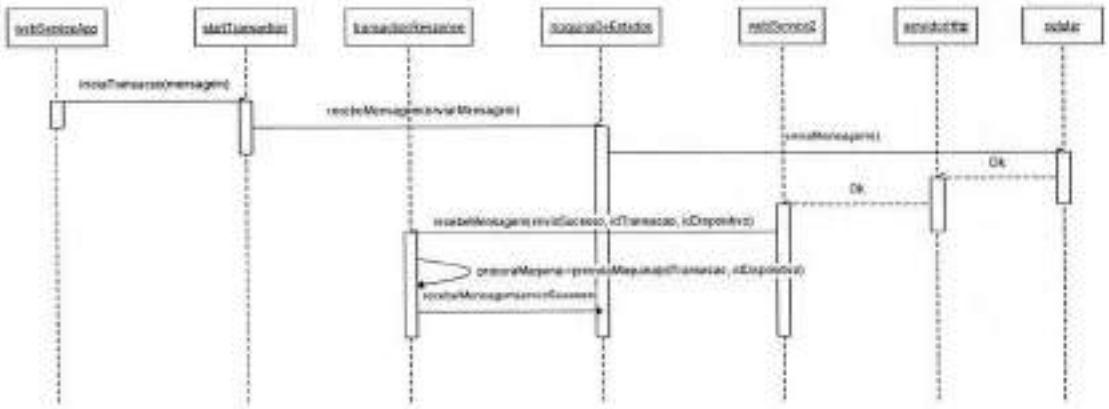

Figura 6: Diagrama de Seqüência de uma transação completa e com sucesso

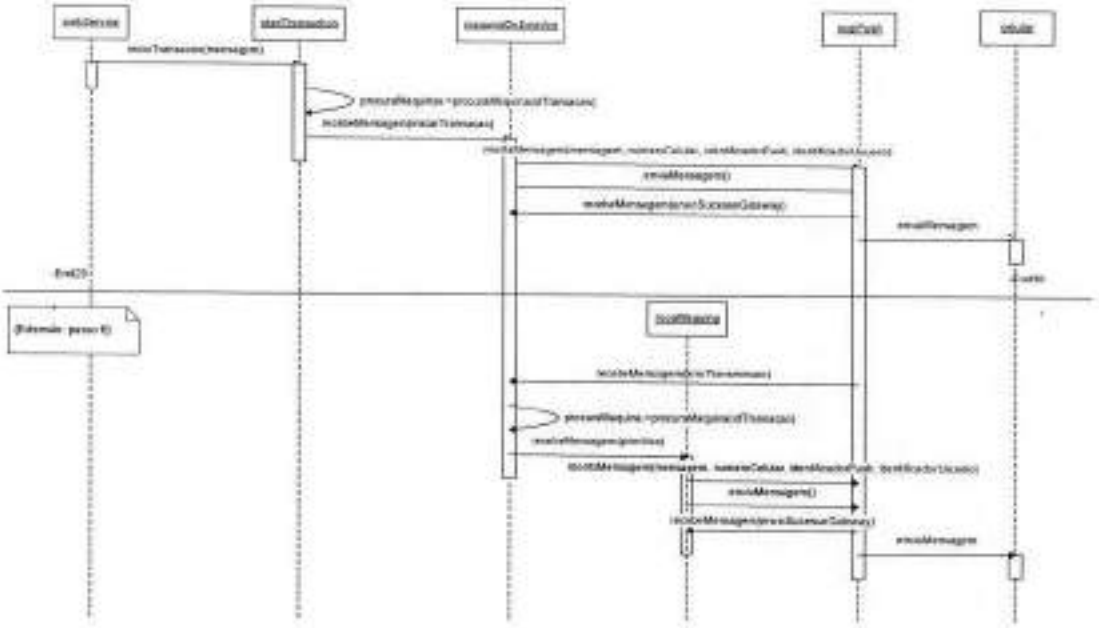

Figura 7: Caso de Uso Iniciar Transação

Figura 8: Caso de Uso Time-out da Máquina de Estados

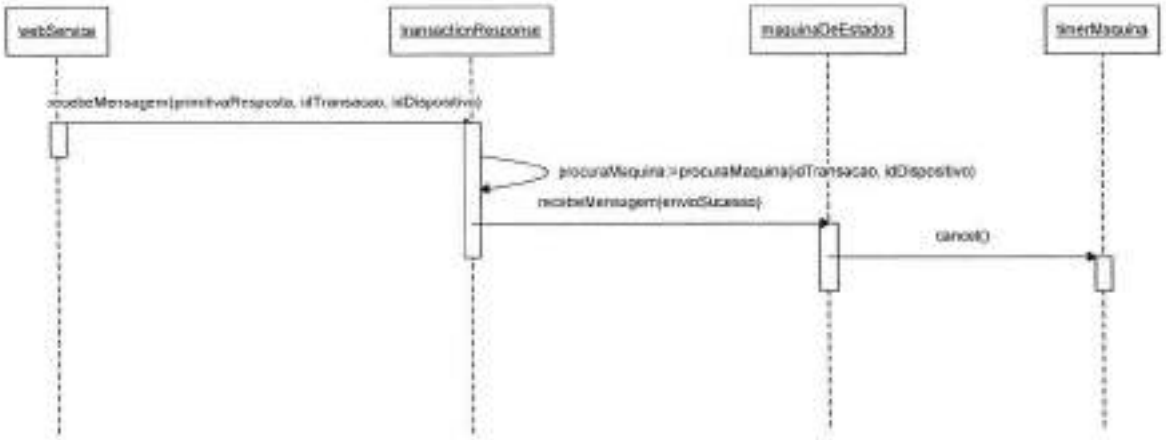

Figura 9: Caso de Uso Receber mensagem celular

APÊNDICE B – Especificação de Projeto de Software

1 Objetivo do Documento

O objetivo deste documento é dar seqüência ao estudo estabelecido pela Especificação de Requisitos de Software (Anexo A) e estudar a implementação do projeto, que é descrita, por sua vez, no Anexo C. Com base nos requisitos levantados através do estudo apresentado no Anexo A, é proposta uma solução técnica, uma plataforma que a viabiliza e mecanismos de engenharia que serão implementados para realizar estas visões.

2 Solução técnica

2.1 Web-services

Permite atender ao requisito de interoperabilidade através da comunicação por um padrão de dados, o formato XML. Além disso, permite dividir o sistema em três canais de comunicação, sendo um de entrada e outro de saída para os serviços legados e um de entrada para os clientes com mobilidade.

Web services são definidos pelo World Wide Web Consortium (W3C) – principal órgão de definição de padrões para a World Wide Web (W3) – como "[...] um sistema de software projetado para prover inter-operação máquina-máquina através de uma rede."[1].

Uma descrição detalhada pode ser encontrada no capítulo 2.3.1.

[1] WEB SERVICE. Wikipedia, the free encyclopedia
Disponível em: <http://en.wikipedia.org/wiki/Web_service>.
Acessado em: 03 de dezembro de 2006.

2.2 Messaging

Permite atender ao requisito de escalabilidade, de desacoplamento e de assincronismo. A técnica de *messaging* permite que componentes conversem entre si através de um canal comum, sem precisarem de referências entre si. A plataforma se encarrega de alocar estes componentes e o canal de comunicação, realizando inclusive balanceamento de carga.

O conceito de *messaging* está atrelado ao conceito de *message-oriented middleware* (MOM), que se define como um software de comunicação intra-aplicações e que se baseia no mecanismo de passagem de mensagens assíncronas, ao invés do mecanismo mais comum de requisição seguida de resposta[1]. Ele se originou como uma alternativa para a integração de sistemas legadas e também para viabilizar o processamento distribuído, especialmente em sistemas interligados através de uma rede.

Uma descrição detalhada pode ser encontrada no capítulo 2.3.2.

2.3 Object-relational mapping

Permite acessar dados sobre o ponto de vista de objetos, abstraindo a implementação relacional de um banco de dados típico e mantendo a aplicação totalmente orientada a objetos. Permite ainda realizar o controle de transações concorrentes no banco de dados de forma a controlar a integridade das informações.

O *object/relational mapping* ou mapeamento entre objetos e dados relacionais é uma técnica de programação que estabelece uma ligação direta entre a representação de dados da

[1] MESSAGE ORIENTED MIDDLEWARE, Wikipedia, the free encyclopedia
Disponível em: <http://en.wikipedia.org/wiki/Message_Oriented_Middleware>.
Acessado em 03 de dezembro de 2006.

orientação a objetos e a representação de dados dos bancos de dados relacionais, criando o efeito de um banco de dados de objetos virtual[1].

Uma descrição detalhada pode ser encontrada no capítulo 2.3.3.

2.4 Wap *push*

Permite atender ao requisito de assincronismo, já que possibilita o contato com clientes de mobilidade assim que uma transação está disponível. Desta forma, o cliente não precisa fazer uma requisição ao servidor e a aplicação ganha agilidade.

O Wap Push é ama tecnologia utilizada para o envio de mensagens assincronamente para dispositivos móveis. A mensagem recebida contém uma Unified Resource Identifier (URI) para alguma página na Internet, sendo que o usuário a acessa apenas aceitando essa mensagem. Essa tecnologia é baseada no modelo cliente-servidor, embora não exista uma requisição feita explicitamente pelo cliente ao servidor.

Uma descrição detalhada pode ser encontrada no capítulo 2.3.5.

3 Plataforma

3.1 Servidor de aplicação

A plataforma de desenvolvimento utilizada é a linguagem Java, utilizando-se ainda a plataforma J2EE para a implementação de mecanismos de troca de mensagens entre componentes *web services* e o componente Gerenciador de Máquinas de Estados do servidor.

[1] OBJECT-RELATIONAL MAPPING, Wikipedia, the free encyclopedia
Disponível em: <http://en.wikipedia.org/wiki/Object-relational_mapping>.
Acessado em: 03 de dezembro de 2006.

Para a implementação e integração com o Sun Application Server (disponível no J2EE), é utilizado o ambiente de desenvolvimento NetBeans.

3.2 Servidor de banco de dados

É utilizado ainda o Hibernate como um *framework* para mapeamento de banco de dados relacionais em classes. O banco de dados utilizado nesse projeto é o Microsoft SQL Server 2005.

3.3 Bibliotecas de comunicação com clientes com mobilidade

Para conexão com os dispositivos móveis, é utilizado a implementação do WapPush da Openwave. Como plataforma de desenvolvimento, é utilizada uma biblioteca disponibilizada pela própria Openwave que provê acesso a métodos de geração de mensagens *push*. Além disso, é utilizado um simulador de dispositivo móvel. Ele é conectado ao *gateway* Wap de testes da Openwave para o recebimento das mensagens.

4 Mecanismos de engenharia

Componente: TransactionRequestReceiverWS	**Tipo:** *web-service*
Objetivo: Estabelecer a comunicação entre o servidor legado e o ***Servidor*** através de um canal de entrada. Classe que representa o *web-service* que contém a lógica para recebimento de um novo pedido de criação de transação e que será convertido em mensagem para chegar ao MDB.	
Interfaces: AckPackage setNewTransactionRequest(TransactionRequestReceiverPackage transactionRequestReceiverPackage);	

Componente: TransactionResponseReceiverWS	**Tipo:** *web-service*
Objetivo: Estabelecer a comunicação entre o cliente com mobilidade e o ***Servidor*** através de um canal de entrada. Classe que representa o *web-service* que contém a lógica para recebimento de resposta para uma transação e que será convertida em mensagem para chegar ao MDB.	
Interfaces: AckPackage setNewTransactionResponse(TransactionResponseReceiverPackage transactionResponseReceiverPackage);	

Componente: TransactionResponseSenderWS	**Tipo:** *web-service*
Objetivo: Estabelecer a comunicação entre o servidor legado e o ***Servidor*** através de um canal de saída. Classe que representa o *web-service* que contém a lógica para envio de resposta final da transação para o servidor legado e que irá interagir com o MDB a fim de obter esta informação.	
Interfaces: TransactionResponseSenderAckPackage getTransactionResponse(TransactionResponseSenderPackage transactionResponseSenderPackage);	

Componente: TransactionRequestReceiverMDB	**Tipo:** Message-driven bean
Objetivo: Receber dados do *web-service* associado através das estruturas do Java Message System. Escuta mensagens da estrutura JMS TransactionRequestReceiverTopic.	
Interfaces: onMessage(Message message);	

Componente: TransactionResponseReceiverMDB	**Tipo:** Message-driven bean
Objetivo: Receber dados do *web-service* associado através das estruturas do Java Message System. Escuta mensagens da estrutura JMS TransactionResponseReceiverTopic.	
Interfaces: onMessage(Message message);	

Componente: TransactionResponseSenderMDB	**Tipo:** Message-driven bean
Objetivo: Receber dados do *web-service* associado através das estruturas do Java Message System. Escuta mensagens da estrutura JMS TransactionResponseSenderTopic.	
Interfaces: onMessage(Message message);	

A implementação de todos os MDBs possui interfaces idênticas. A diferença fica por conta da estrutura do Java Message Service com da qual cada MDB recebe mensagens. Detalhes de cada tipo de classe descrito nas interfaces pode ser encontrado no Anexo C.

APÊNDICE C – Descrição de Código Fonte

1 Módulo envio de mensagens

1.1 Pacotes do módulo de envio de mensagens e suas classes

- br.usp.pcs.projetoformatura.messagessender:

Classe	Descrição
ConfigurationException.java	Sinaliza a ocorrência de um erro no módulo de envio de mensagens.
SendMessageInterface.java	Interface para componentes responsáveis pelo envio de mensagens.

- br.usp.pcs.projetoformatura.messagessender.sms:

Classe	Descrição
SMS.java	Classe que implementa o envio de mensagens para dispositivos móveis por meio de *email*
SMSConfiguration.java	Classe responsável pela leitura e interpretação do arquivo de configuração do módulo SMS.

- br.pcs.usp.projetoformatura.messagessender.wappush:

Classe	Descrição
CodigosRespostaPAP.java	Classe que contém todas as classes de códigos retornados pelo protocolo *Push Access Protocol* (PAP)
WapPush.java	Classe que implementa o envio de mensagens para dispositivos móveis por meio da tecnologia *Push*
WapPushConfigutation.java	Classe responsável pela leitura e interpretação do arquivo de configuração do módulo *WapPush*

1.2 Arquivos de configuração do módulo de envio de mensagens

Arquivo	Descrição
SMSConfiguration.xml	Arquivo com dados de configuração para envio de SMS por *email*.
WapPushConfiguration	Arquivo com dados de configuração para envio de mensagens *push*.

2 Módulo *web* para dispositivos móveis

2.1 Pacotes do módulo *web* para dispositivos móveis e suas classes

- br.pcs.usp.projetoformatura.mobileweb:

Classe	Descrição
Autenticacao.java	Classe que realiza a autenticação do usuário do dispositivo móvel no sistema.
ValidacaoRequisicao.java	Classe responsável pelo envio da resposta do usuário à requisição recebida pelo dispositivo móvel.

3 Módulo núcleo

3.1 Pacotes do módulo núcleo e suas classes

- br.pcs.usp.projetoformatura.core:

Classe	Descrição
TransactionRequestReceiverMDB.java	Classe que contém a lógica para recebimento de mensagem através do JMS. Esta mensagem descreverá uma nova transação no sistema, isto é, um novo conjunto de máquinas de estado.
TransactionResponseReceiverMDB.java	Classe que contém a lógica para recebimento de mensagem através do JMS. Esta mensagem descreverá a respostas dada por um aparelho de celular.
TransactionResponseSenderMDB.java	Classe que contém a lógica para recebimento de mensagem através do JMS. Esta mensagem descreverá a resposta final para o servidor legado que originou esta transação.
StateMachineManagerSB.Java	Classe contém a lógica para o roteamento das mensagens entre os diversos celulares.

- br.pcs.usp.projetoformatura.common:

Classe	Descrição
Agent.java	Classe pai da qual herda RequestAgent, ResponseAgent e SubjectAgent. Contém as variáveis comuns entre os três.
RequestAgent.java	Classe representa o agente de requisição de uma transação, isto é, no caso da aplicação de venda para pessoas com necessidades especiais, seria a loja que requisita autorização para compra.
ResponseAgent.java	Classe representa o agente de resposta de uma transação, isto é, os celulares que serão contatados para autorizarem ou negarem a transação.

SubjectAgent.java	Classe representa o sujeito de uma transação, isto é, no caso da aplicação de venda para pessoas necessitadas, a pessoa que está efetivamente fazendo a compra e requer que alguém autorize por ela.
StateMachine.java	Classe representa a máquina de estados associada a um ResponseAgent. Cada ResponseAgent possui um ciclo de aplicação que vai desde o contato até o envio de resposta, passando pelos casos de *time-out*.
Transaction.java	Classe abriga todas as outras e representa uma transação que contém um SubjectAgent, um RequestAgent e n ResponseAgents (cada um associado a uma StateMachine).

Cada uma destas classes possui uma tabela correspondente no banco de dados.

- br.pcs.usp.projetoformatura.webservices:

Classe	Descrição
TransactionRequestReceiverWS.java	Classe que representa o *web-service* que contém a lógica para recebimento de um novo pedido de criação de transação e que será convertido em mensagem para chegar ao MDB.
TransactionResponseReceiverWS.java	Classe que representa o *web-service* que contém a lógica para recebimento de resposta para uma transação e que será convertida em mensagem para chegar ao MDB.
TransactionResponseSenderWS.java	Classe que representa o *web-service* que contém a lógica para envio de resposta final da transação para o servidor legado e que irá interagir com o MDB a fim de obter esta informação.

- br.pcs.usp.projetoformatura.webservices.common:

Classe	Descrição
AckPackage.java	Classe que representa o pacote pai de uma resposta de um *web-service*.
TransactionRequestReceiverPackage.java	Classe que representa o pacote de uma requisição ao *web-service* TransactionRequestReceiverWS.
TransactionResponseReceiverPackage.java	Classe que representa o pacote de uma requisição ao *web*-service TransactionResponseReceiverWS.
TransactionResponseSenderAckPackage.java	Classe que representa o pacote de uma resposta de um *web-service* TransactionResponseSenderWS.
TransactionResponseSenderPackage.java	Classe que representa o pacote de uma requisição de um *web-service* TransactionResponseSenderWS.

3.2 Arquivo de configuração do módulo núcleo

Classe	Descrição
persistence.xml	Arquivo de configuração do Hibernate.
jdbc_ TemporaryPersistenceLayerDataSource .sun-resource	DataSource que aponta para um ConnectionPool com o banco de dados de persistência temporária (persistência só durante o ciclo da aplicação).
MicrosoftSqlServerPool.sun-resource	ConnectionPool que aponta diretamente para o banco de persistência temporária através do driver JDBC.
jms_ TopicConnectionFactory.sun-resource	Configura a estrutura ConnectionFactory do JMS.
jms_ TransactionRequestReceiverTopic .sun-resource	Referem-se a cada uma das estruturas do JMS utilizadas pelos MDBs para receber as mensagens assincronamente e pelos *web-services* para enviar lhes os dados. Cada arquivo configura uma estrutura.
jms_ TransactionResponseSenderTopic .sun-resource	
jms_ TransactionResponseReceiverTopic .sun-resource	

APÊNDICE D – Descrição de Testes

1 Objetivo do Documento

Esse documento possui o objetivo de descrever os testes realizados no sistema para verificação e validação das funcionalidades de cada componente do sistema.

2 Estratégia de testes

Os testes se basearam em desenvolvimento de métodos inicialmente simples, valendo-se de componentes ou métodos *dummy*, para que a funcionalidade principal pudesse ser testada. Uma vez que as funcionalidades estavam testadas e validadas, outros componentes, também testados e validados, iam sendo integrados e os seus métodos iam sendo desenvolvidos para que pudessem implementar todos os requisitos do sistema, conforme as especificações. O desenvolvimento se baseou,.portanto, em provas de conceito que eram gradualmente incrementadas e testadas até atingir-se a solução final.

3 Descrição dos testes

Teste n. 1	**Requisito Testado:** Iniciar transação
Tipo de Teste	☐ Unidade ☒ Integração
Estratégia de testes / Escopo Testar a inicialização de uma máquina de estados a partir do Gerenciador de Máquinas de Estado.	
Ambiente de teste Gerenciador de Máquinas de estado recebendo o inicio de uma transação.	
Resultados Esperados 1. Gerenciador salvando a máquina de estados corretamente 2. Gerenciador efetuando a busca correta pela máquina de estado correspondente a transação. 3. Máquina de Estados salva, alterando o seu estado DESATIVADO para ENVIADO.	
Resultados Obtidos Os resultados obtidos foram condizentes com o esperado.	
Melhorias Implementadas Feito esse teste inicial, foi implementado o mecanismo de salvar várias máquinas de estado, e a busca dessas máquinas de estado. Como a busca de uma máquina foi pensada para várias máquinas, a adição de várias máquinas foi implementada de modo rápido.	

Teste n. 2	**Requisito Testado:** Envio de mensagem para celular.
Tipo de Teste	☒ Módulo ☐ Integração
Estratégia de testes / Escopo O objetivo desse teste é verificar se uma mensagem é enviada a um simulador/celular para que possa ser aprovada ou não pelo agente autorizador, ou seja, dar continuidade à transação. O envio de mensagens *push* para o simulador pode não funcionar devido a problemas no servidor *Push Proxy Gateway*.	
Ambiente de teste Simulador/celular aptos a receber mensagens SMS ou mensagens *push* com os seguintes dados a serem enviados: identificação do simulador/celular, mensagem a ser enviada, endereço do *Push Proxy Gateway*, identificação do usuário e identificação da transação.	
Resultados Esperados Simulador/celular recebe uma mensagem com uma URI e os parâmetros, identificador da transação, identificador de usuário e identificação do simulador/celular.	
Resultados Obtidos Simulador/celular recebe mensagem com uma URI e os parâmetros, identificador da transação, identificador de usuário e identificação do simulador/celular.	

Teste n. 3	**Requisito Testado:** Envio de mensagem para celular
Tipo de Teste	☐ Unidade ☒ Integração
Estratégia de testes / Escopo Verificar a integração dos componentes WapPush e Máquina de estados para o envio de mensagens.	
Ambiente de teste Máquina de estados inicializada (estado ENVIADO)	
Resultados Esperados **a) Envio com sucesso ao gateway WAP** 1. Resposta de sucesso do gateway WAP sobre o envio da mensagem 2. Máquina de estados do estado ENVIADO para CONECTADO_GATEWAY **b) Erro no envio ao gateway WAP** 1. Resposta de erro do gateway WAP sobre o envio da mensagem 2. Máquina de estados do estado ENVIADO para ERRO	
Resultados Obtidos Os resultados obtidos foram conforme os esperados.	
Melhorias Implementadas Como o gateway WAP utiliza números para a identificação de cada conexão, e ela não pode ser duplicada, foi implementado um número aleatório para a geração dessa identificação. No inicio, estava sendo utilizado a própria identificação da máquina de estados.	

Teste n. 4	**Requisito Testado:** *Time-out* da máquina de estados
Tipo de Teste	☒ Unidade ☐ Integração
Estratégia de testes / Escopo Verificar o funcionamento do mecanismo de *time-out* das máquinas de estado	
Ambiente de teste Máquina de estados inicializada (estados ENVIADO ou CONECTADO_GATEWAY), e timer ativado.	
Resultados Esperados No *time-out* da máquina, ela deve ir para o estado TIME_OUT.	
Resultados Obtidos Os resultados obtidos foram conforme os esperados.	
Melhorias Implementadas No *time-out* da máquina, a máquina que será desativada inicia a transação com uma máquina de estados de mesma transação (máquina já salva, no estado DESATIVADO, e de mesma transação). Para isso, foi implementado na Máquina de Estados o mecanismo de busca já utilizado e testado no Gerenciador, e o mecanismo de ativação de Máquinas, também testado e utilizado no Gerenciador.	

Teste n. 5	**Requisito Testado:** Autenticação de usuário pelo celular.
Tipo de Teste	☒ Módulo ☐ Integração
Estratégia de testes / Escopo O objetivo desse teste é verificar se o usuário está autorizado a validar requisições, desde que ele faça parte de uma transação em andamento.	
Ambiente de teste Simulador/celular capaz de acessar uma página *web* para efetuar a autorização. Os dados para efetuar esse teste são: o nome de usuário, senha, identificação da transação e identificação do simulador/celular.	
Resultados Esperados Redirecionamento para página de erro, caso a senha esteja incorreta e redirecionamento para página de detalhamento da transação, caso a senha esteja correta.	
Resultados Obtidos Redirecionamento para página de erro, quando a senha estava incorreta e redirecionamento para página de detalhamento da transação, quando a senha estava correta.	

Teste n. 6	**Requisito Testado:** Validação da requisição pelo celular.
Tipo de Teste	☒ Módulo ☐ Integração
Estratégia de testes / Escopo O objetivo desse teste é verificar se a resposta do usuário é enviada à máquina de estado correta.	
Ambiente de teste Simulador/celular capaz de acessar uma página *web* para efetuar a validação. Os dados necessários para a realização do teste são: o nome de usuário, senha, identificação da transação, identificação do simulador/celular e resposta do usuário.	
Resultados Esperados Redirecionamento para página de erro, caso algum dos dados estejam incorretos, por exemplo, senha errada, ou redirecionamento para página de confirmação de envio da resposta, caso todos os dados estejam corretos.	
Resultados Obtidos Redirecionamento para página de erro, quando existe algum dado incorreto, e redirecionamento para página de confirmação de envio de resposta quando todos os dados estavam corretos.	

Teste n. 7	**Requisito Testado:** Operação dos *web-services*
Tipo de Teste	☒ Unidade ☐ Integração
Estratégia de testes / Escopo Verificar o funcionamento dos *web-services* e dos mecanismos de empacotamento e mapeamento de dados em suas interfaces.	
Ambiente de teste *Web-services* instalados no Sun Application Server e chamados por aplicações clientes *dummy*.	
Resultados Esperados Correto mapeamento dos dados transportados em XML através do envelope SOAP para objetos Java.	
Resultados Obtidos Os resultados obtidos foram conforme os esperados.	

Teste n. 8	**Requisito Testado:** Operação dos Message Driven Beans e dos *containers* de mensagens do Java Messaging System (estruturas Topic e Queue)
Tipo de Teste	☒ Unidade ☐ Integração
Estratégia de testes / Escopo Verificar o funcionamento e tempo de resposta da operação composta pelo envio de uma mensagem para uma estrutura do JMS e conseqüente recebimento por um MDB através da alocação dinâmica controlada pelo Sun Application Server.	
Ambiente de teste MDBs instalados e estruturas configuradas no Sun Application Server. Mensagens são enviadas às estruturas por clientes *dummy*.	
Resultados Esperados Correta entrega de mensagens para os MDBs. O teste também serve como observação de comportamento como diferenças entre Topic e Queue e de tempo de resposta da operação.	
Resultados Obtidos Os resultados obtidos foram conforme os esperados, no entanto, a observação do comportamento do sistema em caso de erros originados do processamento da mensagem dentro do MDB incentivou a configuração fina das estruturas de mensagem.	
Melhorias Implementadas Através da observação dos resultados, foi possível configurar parâmetros avançados de operação do JMS a fim de adequar o funcionamento aos requisitos de operação.	

Teste n. 9	**Requisito Testado:** Operação com o banco de dados através das interfaces do EJB 3.0 e através da biblioteca Hibernate
Tipo de Teste	☒ Unidade ☐ Integração
Estratégia de testes / Escopo Verificar o funcionamento das operações de inserção, edição e remoção de dados, bem como o controle de transações concorrentes que garante a integridade das informações.	
Ambiente de teste EJBs *dummy* instalados no Sun Application Server acessando o banco de dados.	
Resultados Esperados Dados de exemplo *hard-coded* nos EJBs *dummy* corretamente persistidos e recuperados do banco de dados.	
Resultados Obtidos Os testes realizados permitiram configurar o mapeamento de classes e a configuração da conexão com o banco de dados até obter-se uma solução funcional. Uma vez com essa solução final, os resultados foram conforme o esperado.	
Melhorias Implementadas A observação dos resultados permitiu realizar a configuração fina do mapeamento entre tabelas relacionais e objetos, bem como dos parâmetros de configuração da conexão com o banco de dados. Todo o mapeamento foi feito a mão, já que as ferramentas de automação existentes não otimizam a solução e nem mapeiam a solução de forma precisa.	

Teste n. 10	**Requisito Testado:** Integração entre *web-services*, MDBs e acesso às classes de persistência.
Tipo de Teste	☐ Unidade ☒ Integração
Estratégia de testes / Escopo Verificar o funcionamento dos componentes em conjunto.	
Ambiente de teste Clientes *dummy* enviando mensagens que simulam a operação da aplicação real. Componentes de servidor devidamente instalados e configurados no Sun Application Server.	
Resultados Esperados Espera-se que o ciclo de vida da aplicação termina com sucesso.	
Resultados Obtidos Após ajustes, os resultados obtidos foram conforme o esperado.	