UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MODEL DRIVEN RICHUBI – PROCESSO DIRIGIDO A MODELOS PARA A CONSTRUÇÃO DE INTERFACES RICAS DE APLICAÇÕES UBÍQUAS SENSÍVEIS AO CONTEXTO

CARLOS EDUARDO CIRILO

ORIENTADOR: PROF. DR. ANTONIO FRANCISCO DO PRADO

CO-ORIENTADORA: PROFª. DRª. LUCIANA APARECIDA MARTINEZ ZAINA

São Carlos – SP
Maio/2011

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MODEL DRIVEN RICHUBI – PROCESSO DIRIGIDO A MODELOS PARA A CONSTRUÇÃO DE INTERFACES RICAS DE APLICAÇÕES UBÍQUAS SENSÍVEIS AO CONTEXTO

CARLOS EDUARDO CIRILO

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Engenharia de Software.
Orientador: Dr. Antonio Francisco do Prado
Co-Orientadora: Drª. Luciana Ap. M. Zaina

São Carlos – SP
Maio/2011

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

C578md

Cirilo, Carlos Eduardo.
Model Driven RichUbi – processo dirigido a modelos para a construção de interfaces ricas de aplicações ubíquas sensíveis ao contexto / Carlos Eduardo Cirilo. -- São Carlos : UFSCar, 2011.
197 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2011.

1. Engenharia de software. 2. Computação ubíqua. 3. Computação ciente de contexto. 4. Desenvolvimento orientado por modelos. 5. Interfaces gráficas. I. Título.

CDD: 005.1 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Model Driven RichUbi – Processo Dirigido a Modelos para a Construção de Interfaces Ricas de Aplicações Ubíquas Sensíveis ao Contexto"

CARLOS EDUARDO CIRILO

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

Membros da Banca:

Prof. Dr. Antonio Francisco do Prado
(Orientador - DC/UFSCar)

Profa. Dra. Luciana Aparecida Martinez Zaina
(DC/UFSCar/Sorocaba)

Prof. Dr. Wanderley Lopes de Souza
(DC/UFSCar)

Prof. Dr. Luís Ferreira Pires
(Universidade de Twente/Holanda)

São Carlos
Maio/2011

De maneira geral, dedico este trabalho à minha família,
que sempre esteve do meu lado me dando apoio tanto nos momento bons
quanto nos momentos mais difíceis desta incrível jornada.
De forma especial, dedico tudo isto aos meus pais, Teresinha & Zezinho,
que batalharam para me proporcionar uma vida digna e honesta.
Também dedico este trabalho à minha avó, Sílvia (madrinha),
que, pela sua luta diária, mostrou-me que a perseverança em Deus
é a melhor forma de superar as adversidades
e manter-se firme diante das coisas que não podemos controlar.

AGRADECIMENTOS

Foram muitos os que, direta ou indiretamente, me ajudaram a concluir este trabalho. Meus mais sinceros agradecimentos...

... a Deus, que me abençoou nesta vida, permitindo que eu chegasse até aqui, sendo que sem Sua ajuda nada teria sido possível. Deus, obrigado por me guiar nesta jornada e me fortalecer nos momentos de dificuldade. Sempre que precisei e supliquei por Seu auxílio, e acreditei que ele viria, ele realmente veio, de alguma forma. Muito obrigado, Senhor!

... à minha família, meu porto seguro, que, apesar da distância, sempre estiveram presentes em pensamentos e orações. Em especial, aos meus pais, Teresinha e Zezinho; aos meus avós, Silvia e Mário; aos meus tios, Lúcia e Laércio, e Carmen e Carlos; à minha irmã, Cidinha (dei trabalho pra você, hein? ☺); e ao Luciano e à Sônia. Gente, obrigado mesmo!

... ao meu orientador, professor Prado, pela oportunidade e confiança, e também por toda ajuda e ensinamentos ao longo desses anos.

... à minha orientadora, professora Luciana, pelos valiosos conselhos, orientações e amizade nesses anos derradeiros.

... aos meus companheiros e amigos mais próximos, com quem sempre pude contar, e com os quais pude compartilhar as dores e alegrias deste percurso: Randal, Kelly e dona Rosa. Meus amigos, obrigado de coração!

... aos amigos e colegas que fiz neste mestrado, pelas horas de descontração e por terem vivenciado mais intensamente comigo os momentos desta jornada: Lu, Le, Vi, Van, Walter e Mah. Valeu, pessoal!

... a todos os colegas do LaBDES e do PPG-CC, pelo companheirismo, receptividade e eventuais apoios. Galera, obrigado!

... aos professores, pelos ensinamentos, e à secretaria e funcionários do DC, pelo apoio.

... aos integrantes da banca de qualificação e aos professores debatedores do WTES 2010, pelas sugestões e contribuições que possibilitaram novas reflexões e horizontes a este trabalho.

...ao professor Luís Pires e ao professor Wanderley, pelas orientações, contribuições e sugestões dadas a este trabalho.

... aos alunos da turma de 2010 da disciplina Tópicos em Informática II da UFSCar, pelo interesse e participação no experimento realizado neste trabalho.

... ao Sr. Waldomiro Barioni Júnior, pesquisador e estatístico da Embrapa – Pecuária Sudeste, e à Drª. Cecília Candolo, professora e pesquisadora do Departamento de Estatística da UFSCar, pelas valiosas contribuições nas análises estatísticas desempenhadas neste trabalho.

... à CAPES e ao PPG-CC, pelo apoio financeiro.

Enfim... a todos que, de alguma maneira, contribuíram para a realização deste trabalho e torceram para que tudo desse certo. Que Deus abençoe a todos!

Muito obrigado!

Posso ter defeitos, viver ansioso e ficar irritado algumas vezes,
mas não esqueço de que minha vida é a maior empresa do mundo
e posso evitar que ela vá à falência.
Ser feliz é reconhecer que vale a pena viver
apesar de todos os desafios, incompreensões e períodos de crise.
Ser feliz é deixar de ser vítima dos problemas
e se tornar um autor da própria história.
É atravessar desertos fora de si,
mas ser capaz de encontrar um oásis no recôndito da sua alma.
É agradecer a Deus a cada manhã pelo milagre da vida.
Ser feliz é não ter medo dos próprios sentimentos.
É saber falar de si mesmo.
É ter coragem para ouvir um "não".
É ter segurança para receber uma crítica, mesmo que injusta.
Pedras no caminho?
Guardo todas, um dia vou construir um castelo...
(Fernando Pessoa)

RESUMO

A Web 2.0 permitiu aos usuários maior interatividade com as aplicações Web. As chamadas Aplicações de Internet Ricas (*RIAs – Rich Internet Applications*) transpuseram os limites das interfaces simples construídas apenas em *Hypertext Markup Language (HTML)*. Através da adoção de tecnologias que permitem a criação de interfaces mais avançadas, as RIAs assemelham-se à aparência e comportamento das aplicações *desktop*. Por outro lado, a demanda por software na Computação Ubíqua, onde o acesso às aplicações ocorre de qualquer lugar, a qualquer hora e a partir de diferentes dispositivos, fez surgir novos desafios para a Engenharia de Software. Um desses desafios está relacionado com a adaptação das aplicações acessadas por diferentes dispositivos em contextos distintos. Dada a diversidade de dispositivos, redes de acesso, ambientes e contextos possíveis, prover aplicações que satisfaçam as peculiaridades de cada dispositivo de acesso, ao mesmo tempo em que mantêm um comportamento e aparência coerentes face às mudanças que ocorrem no ambiente ao redor, tornou-se uma difícil tarefa para os engenheiros de software. Nas aplicações que utilizam de interfaces ricas na Web 2.0, para melhorar a interatividade, essa tarefa torna-se mais complexa devido à necessidade de preservar os aspectos de interação que proporcionam aos usuários uma rica experiência com a aplicação. Tal tarefa pode ser facilitada usando um processo de software que oriente o desenvolvedor na construção de uma aplicação ubíqua, considerando os diferentes contextos em que se executa a aplicação. Diante desses desafios e visando a dar suporte ao desenvolvimento de interfaces ricas de aplicações ubíquas que se adaptam quando visualizadas em diferentes dispositivos, neste trabalho é proposto o processo denominado *Model Driven RichUbi*. Com base nas concepções de *Desenvolvimento Dirigido a Modelos* e *Modelagem Específica de Domínio*, são definidos atividades e artefatos que orientam a modelagem e geração parcial de código das interfaces ricas para diferentes contextos. No *Model Driven RichUbi*, também são utilizados adaptadores dinâmicos de conteúdo que refinam as versões produzidas das interfaces para se adequarem às peculiaridades do dispositivo de acesso identificadas do contexto da interação em tempo de execução. O apoio computacional no Domínio de Interfaces Ricas empregado no processo tem a vantagem de poder ser reutilizado no desenvolvimento de interfaces ricas adaptativas em aplicações ubíquas de diferentes domínios, colaborando para a redução de esforços e aumento da produtividade.

Palavras-chave: Processo de Software, Interfaces Ricas Adaptativas, Computação Ubíqua, Desenvolvimento Dirigido a Modelos, Modelagem Específica de Domínio, Sensibilidade ao Contexto.

ABSTRACT

Web 2.0 allowed users more interactivity with Web applications. The so-called *Rich Internet Applications (RIAs)* have transposed the boundaries of simple interfaces built only in *Hypertext Markup Language (HTML)*. Through the adoption of technologies that enable the creation of more advanced interfaces, RIAs resemble the appearance and behavior of *desktop* applications. On the other hand, the demand for software in Ubiquitous Computing, in which access to applications occurs anywhere, at any time and from different devices, has raised new challenges for Software Engineering. One of these challenges is related to the adaptation of the contents of an application to the numerous devices that can access it in distinct contexts. Given the diversity of devices, access networks, environments and possible contexts, providing applications that meet the peculiarities of each access device, while keeping a consistent appearance and behavior in view of the changes occurring in the surrounding environment, has become a difficult task for software engineers. In applications that use rich interfaces in Web 2.0 for improving the interactivity, this task becomes even more complex due to the need of preserving the interaction aspects that afford users a richer experience with the application. This task can be facilitated using a software process that guides developers in building a ubiquitous application, considering the different contexts involved in its execution. Faced with these challenges, this work proposes a software process, named *Model Driven RichUbi*, aiming at supporting the development of rich interfaces for ubiquitous applications that adapt themselves when viewed on different types of devices. Based on the *Model Driven Development* and *Domain-Specific Modeling* conceptions, in the process are defined activities and artifacts that help in modeling and partial code generation of rich interfaces for different platforms. Besides, dynamic content adapters that refine the produced interface versions are also employed in the process, so that the developed interfaces can adapt to the peculiarities of the access device identified from the interaction context at runtime. The computational support focused on the Rich Interfaces Domain employed in the process is advantageous since it can be reused to simplify the development of adaptive rich interfaces for ubiquitous applications of several fields, which contributes to effort reduction and productivity increasing.

Keywords: Software Process, Adaptive Rich Interfaces, Ubiquitous Computing, Model-Driven Development, Domain-Specific Modeling, Context Awareness.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AJAX	*Assynchronous JAvaScript and XML* / JavaScript e XML Assíncronos
ASC	Aplicação Sensível ao Contexto
ASPS	*Ambulance Space Positioning System / Sistema de Posicionamento Espacial de Ambulâncias*
CASE	*Computer-Aided Software Engineering* / Engenharia de Software Assistida por Computador
CMM	*Capability Maturity Model* / Modelo de Maturidade da Capacidade
CSS	*Cascading Style Sheet* / Folha de Estilo em Cascata
DOM	*Document Object Model* / Modelo de Objetos de Documentos
DSL	*Domain Specifc Language* / Linguagem Específica de Domínio
EA	Engenharia de Aplicação
ED	Engenharia de Domínio
DynamiCOS	*Dynamic Composition of Services* / Framework de Composição Dinâmica de Serviços
EC	Elemento Contextual
EICAF	*Extended Internet Content Adaptation Framework* / Framework Estendido de Adaptação de Conteúdo da Internet
EMF	*Eclipse Modeling Framework* / Framework de Modelagem do Eclipse
ERP	*Enterprise Resource Planning* / Planejamento de Recursos Corporativos
GSM	*Global System for Mobile Communications* / Sistema Global para Comunicações Móveis
HTML	*Hypertext Markup Language* / Linguagem de Marcação de Hipertexto
HTTP	*HyperText Transfer Protocol* / Protocolo de Transferência de Hipertexto
ICAF	*Internet Content Adaptation Framework* / Framework de Adaptação de Conteúdo da Internet
ICAP	*Internet Content Adaptation Protocol* / Protocolo de Adaptação de Conteúdo da Internet

IDE	*Integrated Development Environment* / Ambiente de Desenvolvimento Integrado
ISBN	*International Standard Book Number* / Número Padrão Internacional de Livro
JET	*Java Emitter Templates* / Modelos Emissores de Java
JSF	*Java Server Faces* / Faces Servidoras de Java
M2C	*Model to Code* / Modelo para Código
M2M	*Model to Model* / Modelo para Modelo
MDA	*Model-Driven Architecture* / Arquitetura Dirigida a Modelos
MDD	*Model-Driven Development* / Desenvolvimento Dirigido a Modelos
MDE	*Model-Driven Engineering* / Engenharia Dirigida a Modelos
MDSD	*Model-Driven Software Development* / Desenvolvimento de Software Dirigido a Modelos
MD*	Acrônimo que abrange todas as abordagens de desenvolvimento de software dirigidas a modelos
MB-UID	*Model-Based User Interface Development* / Desenvolvimento de Interfaces Dirigido a Modelos
Model Driven RichUbi	*Model Driven Process to Construct Rich Interfaces for Context-Sensitive Ubiquitous Applications* / Processo Dirigido a Modelos para a Construção de Interfaces Ricas de Aplicações Ubíquas Sensíveis ao Contexto
MOF	*Meta Object Facility* / Infra-estrutura de Meta-Objetos
MVCASE	*Multiple-View CASE* / Ferramenta CASE de Múltipla Visão
OMG	*Object Management Group* / Grupo de Gerenciamento de Objetos
OWL	*Web Ontology Language* / Linguagem de Ontologia da Web
OWL-S	*Web Ontology Language for Services* / Linguagem de Ontologia da Web para Serviços
P2P	*Peer-to-Peer* / Arquitetura Par a Par
PC	*Personal Computer* / Computador Pessoal
PDA	*Personal Digital Assistant* / Assistente Pessoal Digital
QIP	*Quality Improvement Paradigm* / Pradigma de Melhoria da Qualidade

RFID	*Radio-Frequency IDentification* / *Identificação* por Rádio Frequência
RIA	*Rich Internet Application* / Aplicação de Internet Rica
SADT	*Structured Analysis and Design Technique* / Técnica de Projeto e Análise Estruturada
SDK	*Software Development Kit* / Kit de Desenvolvimento de Software
SCOUT	*Semantic COntext-aware Ubiquitous scouT* / Observador Ubíquo-Semântico Sensível ao Contexto
SOAP	*Simple Object Access Protocol* / Protocolo Simples de Acesso a Objetos
UbiCon	*Ubiquitous Context Framework* / *Framework* de Contexto Ubíquo
UIG	*User Interface Generator* / Gerador de Interface do Usuário
UIML	*User Interface Markup Language* / Linguagem de Marcação de Interface do Usuário
UML	*Unified Modeling Language* / Linguagem de Modelagem Unificada
URI	*Uniform Resource Identifier* / Identificador Uniforme de Recursos
UsiXML	*USer Interface eXtensible Markup Language* / Linguagem Extensível de Marcação de Interface do Usuário
W3C	*World Wide Web Consortium* / Consórcio da *World Wide Web*
WML	*Wireless Markup Language* / Linguagem de Marcação Móvel
WSDL	*Web Services Description Language* / Linguagem de Descrição de Serviços Web
WURFL	*Wireless Universal Resource File* / Arquivo Universal de Recursos Móveis
XHTML	*eXtensible Hypertext Markup Language* / Linguagem de Marcação Extensível de Hipertexto
XIML	*Extensible Interface Markup Language* / Linguagem de Marcação Extensível de Interfaces
XMI	*XML Metadata Interchange* / Intercâmbio de Metadados XML
XML	*eXtensible Markup Language* / Linguagem de Marcação Extensível

SUMÁRIO

Capítulo 1

INTRODUÇÃO

1.1 Visão Geral

Nos últimos anos, mudanças ocorridas na forma com que usuários e empresas utilizam a Web para fornecer serviços, acessar conteúdo e interagir com outros usuários, evidenciadas, sobretudo, após o "estouro da bolha" em 2001, levaram ao surgimento do termo Web 2.0 (O'REILLY, 2005). Desde então, esse termo tem sido utilizado para referir-se a uma nova geração de aplicações Web projetadas especialmente para apoiar a colaboração e o compartilhamento dos conteúdos gerados pelos usuários (NORRIE, 2008).

Dentre as características marcantes das aplicações de Web 2.0 destaca-se o uso de interfaces ricas que possibilitam aos usuários uma experiência mais significativa com as aplicações. Nesse contexto, as chamadas Aplicações de Internet Ricas (*RIAs – Rich Internet Applications*) transpuseram os limites das interfaces simples construídas apenas com *Hypertext Markup Language (HTML)* básico. Através do uso de tecnologias que permitem a criação de interfaces mais atrativas e com maior sensibilidade, tais como comunicação assíncrona com o servidor, recursos de "clique e arraste", captura e exibição de vídeos, mapas, etc., as RIAs aproximam-se da aparência, comportamento e usabilidade das aplicações *desktop* (DEITEL & DEITEL, 2009).

A miniaturização dos dispositivos computacionais de uso pessoal aliada aos avanços recentes das tecnologias de comunicação sem fio e ao amadurecimento da Computação Ubíqua (WEISER, 1999) ampliaram significativamente as possibilidades de acesso a uma vasta gama de aplicações de diversas áreas (FORTE et al., 2008).

Até pouco tempo atrás, havia poucas maneiras para acessar o conteúdo *online*, dentre as quais o principal meio de acesso era através de computadores pessoais (*Personal Computers - PCs*). Contudo, esta situação tem mudado rapidamente. Hoje, por exemplo, é possível ler *emails*, efetuar transações financeiras, compartilhar recursos (hardware, dados e software) e usufruir de uma série de outras aplicações através de um pequeno celular ou de um sofisticado *smartphone*, quer o usuário esteja parado ou em movimento, quer esteja em casa, na rua ou no trabalho. Neste cenário, a visão de ubiquidade computacional, introduzida por Weiser há cerca de duas décadas, tem sido impulsionada por novos desenvolvimentos que viabilizam o fácil acesso à informação a qualquer hora, de qualquer lugar e por meio de qualquer dispositivo à disposição do usuário (ARAÚJO, 2003; HANSMANN et al., 2003).

A natureza dinâmica dos ambientes de Computação Ubíqua, contudo, impõe uma série de desafios e requisitos adicionais ao desenvolvimento de software (GARLAN & SCHMERL, 2001; SPÍNOLA et al., 2007). Um dos aspectos críticos no desenvolvimento de aplicações que operam nesses ambientes é a premissa de que as mesmas devem ser capazes de executar e adaptar seu comportamento e conteúdo à diversidade dos dispositivos computacionais do usuário, bem como ao ambiente no qual ele está imerso (GAJOS & WELD, 2004). Dada a diversidade de dispositivos, redes de acesso, ambientes e contextos possíveis, prover aplicações que satisfaçam as peculiaridades de cada dispositivo de acesso, ao mesmo tempo em que mantêm um comportamento e aparência coerentes face às mudanças que ocorrem no ambiente ao redor, tornou-se uma tarefa de difícil acompanhamento para os engenheiros de software (EISENSTEIN et al., 2000; PATERNÒ et al., 2008; SINGH, 2004).

Com relação à adaptação baseada no dispositivo, o principal desafio refere-se à heterogeneidade de configurações (VIANA & ANDRADE, 2008). Cada dispositivo apresenta um perfil específico, o qual compreende características como poder de processamento, capacidades gráficas, resolução de tela, suporte para determinada linguagem de marcação, dentre outras. Além disso, novas plataformas e dispositivos com configurações distintas são lançados quase que diariamente no mercado. Como resultado, construir e manter versões específicas da aplicação que atendam às particularidades de cada plataforma e dispositivo existentes acabaram tornando-se tarefas não gerenciáveis. Dentre outros problemas, tal abordagem, além de requerer altos investimentos financeiros, demanda grandes esforços de desenvolvimento e, ainda assim, pode resultar em versões inconsistentes da aplicação. Além disso, a

existência de várias versões da aplicação dificulta sua manutenção, pois as modificações e evoluções terão que ser gerenciadas em diferentes versões (EISENSTEIN et al., 2000; SINGH, 2004).

Apesar desses problemas, é necessário que as aplicações desenvolvidas mantenham um comportamento coerente e apresentem-se de forma compatível ao perfil do dispositivo utilizado, preservando suas características originais de modo que a interação não seja prejudicada. Ainda, no caso das aplicações de Web 2.0, isso se torna mais complexo, visto a necessidade de preservar suas características de interação que proporcionam aos usuários uma rica experiência com a aplicação. Dessa maneira, uma solução automatizada que permita às aplicações adaptarem-se de forma apropriada faz-se necessária (EISENSTEIN et al., 2000; SINGH, 2004).

A adaptação de conteúdo surge como alternativa para aliviar a necessidade de se construir múltiplas versões da aplicação (SOUZA et al., 2011). Na adaptação de conteúdo, os conteúdos da interface gráfica da aplicação, originalmente projetados para um cenário de uso particular, são automaticamente convertidos para adequarem-se a outro cenário, mantendo a compatibilidade com as capacidades do dispositivo de acesso, as preferências do usuário, as características da rede e outros fatores relevantes (BUCHHOLZ et al., 2003; SINGH, 2004). Para alcançar tal objetivo, porém, é necessário identificar dinamicamente os recursos disponíveis e as variações contextuais no dispositivo, ambiente, pessoas e objetos circundantes, de modo que a aplicação ajuste-se apropriadamente. Diante disso, o conceito de contexto exerce um papel fundamental para o desenvolvimento de aplicações adaptativas (COUTAZ et al., 2005).

Contexto envolve informações que se referem a vários aspectos associados ao funcionamento da aplicação, tais como o usuário, o dispositivo de acesso, o ambiente, a rede, dentre outros (DEY, 2001). Aplicações sensíveis ao contexto são aquelas que usam contexto para fornecer informações e serviços relevantes aos usuários, e executam tarefas de forma automática sem a necessidade de intervenção explícita dos mesmos (BALDAUF et al., 2007; VIEIRA et al., 2009). Neste sentido, a sensibilidade ao contexto torna-se essencial para que uma aplicação de Web 2.0 forneça adaptabilidade e flexibilidade de forma que, dentre outras possibilidades, consiga adaptar suas interfaces ricas às condições variantes a que está sujeita em um ambiente ubíquo, em conformidade com o perfil do dispositivo de acesso.

Para facilitar a construção de aplicações ubíquas sensíveis ao contexto, abordagens de Desenvolvimento Dirigido a Modelos (*Model-Driven Development - MDD*) (FRANCE & RUMPE, 2007) podem ser empregadas (SERRAL et al., 2010). O MDD foca na modelagem e na transformação de modelos em artefatos de implementação com o intuito de prover maior automação e produtividade aos processos de desenvolvimento de software. No MDD, os modelos que especificam a aplicação são usados para gerar código, total ou parcialmente, para diferentes tecnologias, de forma que as tarefas repetitivas de codificação para as diversas plataformas fiquem encapsuladas nas transformações, economizando tempo e esforço dos desenvolvedores. Na Computação Ubíqua, onde o universo de dispositivos é grande, o MDD pode ser bastante útil para tornar mais ágil o desenvolvimento de aplicações para as variadas plataformas e dispositivos.

Assim, visando a prover suporte ao desenvolvimento de aplicações de Web 2.0 na Computação Ubíqua, este trabalho define um processo de desenvolvimento dirigido a modelos, denominado *Model Driven RichUbi,* com foco na construção de interfaces ricas que se adaptam conforme o perfil do dispositivo recuperado do contexto da interação. Com base nas concepções de MDD e Modelagem Específica de Domínio (*Domain Specific Modeling – DSM*) (KELLY & TOLVANEN, 2008), o processo estabelece atividades e artefatos que auxiliam na modelagem e geração parcial de código das interfaces ricas para diferentes plataformas. No *Model Driven RichUbi*, também faz-se uso de adaptadores dinâmicos de conteúdo que refinam as versões produzidas das interfaces para se adequarem às peculiaridades do dispositivo de acesso em tempo de execução. Esses artefatos, construídos previamente no processo numa etapa de *Engenharia de Domínio (ED)*, por estarem associados ao Domínio de Interfaces Ricas – um domínio transversal aos domínios de aplicações – podem ser reutilizados durante a etapa de *Engenharia de Aplicação (EA)* para simplificar o desenvolvimento das interfaces ricas adaptativas de aplicações ubíquas de diversas áreas, o que colabora para a redução de esforços e aumento da produtividade.

1.2 Motivação e Objetivos

Com a Internet tornando-se onipresente, o uso de aplicações e serviços em rede tem sido cada vez maior e deverá aumentar nos próximos anos, ao mesmo tempo em que dispositivos móveis com rápida conexão de dados se proliferam rapidamente no mercado (GONÇALVES DA SILVA et al., 2010). Estima-se que, entre 2013 e 2014, mais de três bilhões de pessoas ao redor do planeta serão capazes de realizar transações eletrônicas na Internet por meio de tecnologia móvel ou de computadores pessoais. A tendência é a de que o número de clientes móveis acessando aplicações *online* supere o de computadores estacionários. Até 2014 prevê-se que a taxa de penetração de *smartphones* e telefones celulares dotados com capacidades para acesso à Internet no mercado global seja de 90%, excedendo 1,82 bilhões de unidades, ao passo que o número de computadores pessoais ativos chegue em torno de 1,78 bilhões (GARTNER, 2010). Essa realidade tem levado à necessidade premente por aplicações que forneçam suporte à mobilidade e a diferentes tipos de dispositivos e usuários em múltiplos contextos de uso (GARTNER, 2010; VIEIRA et al., 2011).

A questão da adaptação das aplicações conforme o contexto não é recente e tem despertado o interesse de diversos pesquisadores ao longo dos anos. Diferentes soluções têm sido propostas pela comunidade acadêmica. Algumas dessas soluções objetivam a definição de *frameworks* que disponibilizam funcionalidades de adaptação de comportamento e conteúdo de aplicações ubíquas (FORTE et al., 2008; VAN WOENSEL, 2009). Outras soluções concentram-se na elaboração de ferramentas que facilitam o desenvolvimento de aplicações adaptativas (PATERNÒ et al., 2008; VIANA & ANDRADE, 2008). Há também pesquisas que têm foco na definição de processos de software que orientam o desenvolvimento de aplicações sensíveis ao contexto (SANTOS, 2008; VIEIRA et al., 2011).

Para ilustrar a importância que tem sido dada à pesquisa com foco no desenvolvimento de aplicações adaptativas, em 2006 um comitê organizado pela *Sociedade Brasileira de Computação (SBC)*, composto por pesquisadores de diferentes subáreas da Computação, definiu os cinco grandes desafios da pesquisa em Ciência da Computação no Brasil para os 10 anos seguintes (SBC, 2006). A adaptação da aplicação (conteúdo, informação e comportamento) com base em

informações contextuais é mencionada explicitamente em três desses desafios, mais especificamente: a recuperação de informações, a partir de grandes volumes de dados, que sejam mais apropriadas às necessidades e preferências do usuário; a produção de conteúdos e interfaces flexíveis e adaptáveis que se ajustam conforme o contexto sociocultural do usuário, com o intuito de propiciar o acesso participativo e universal ao conhecimento a todos os cidadãos; e o desenvolvimento de aplicações ubíquas que possam ser acessadas de qualquer lugar, qualquer hora e a partir de qualquer dispositivo.

Nesses desafios, o contexto aparece como fator chave para permitir o desenvolvimento de aplicações adaptativas. O contexto possibilita que as aplicações refinem a informação disponível em informação relevante, escolham ações apropriadas a partir de uma lista de possibilidades, ou determinem o melhor método para a disponibilização da informação (SANTOS, 2008). Predições apontam que por volta de 2015 o contexto será tido como a principal fonte de informações para que as aplicações forneçam conteúdos personalizados e se adaptem em diferentes dispositivos e situações de forma automatizada (GARTNER, 2010).

Motivados em contribuir com esses desafios, este trabalho teve por objetivo geral definir um processo dirigido a modelos que prescreve atividades e artefatos para apoiar a construção de interfaces ricas de aplicações ubíquas que se adaptam quando visualizadas em diferentes dispositivos de acordo com seus perfis extraídos do contexto da interação. Pretende-se que o processo contribua para tornar mais ágil o desenvolvimento dessas aplicações, colaborando para o aumento da produtividade e redução dos esforços de desenvolvimento.

Para reduzir o número de versões de interfaces construídas em tempo de desenvolvimento, no *Model Driven RichUbi* é empregada uma abordagem de adaptação híbrida, combinando adaptação estática (construção de várias versões em tempo de desenvolvimento) e dinâmica (adaptação de todo o código em tempo de execução). Essa abordagem combina geração de código a partir da modelagem em tempo de desenvolvimento, com a geração de código em tempo de execução. Dessa forma, versões mais genéricas da interface são construídas, cada qual adequada a um determinado grupo de dispositivos (adaptação estática). Os adaptadores dinâmicos de conteúdo reutilizados no processo, no acesso à aplicação, completam, quando necessário, o restante da adaptação conforme as peculiaridades do dispositivo (adaptação dinâmica). Assim, o desenvolvimento torna-

se simplificado, uma vez que um número menor de versões podem ser desenvolvidas.

Em síntese, o objetivo geral deste trabalho pode ser desmembrado nos seguintes itens:

- Formular um processo com base em MDD e DSM para simplificar a construção de interfaces ricas de aplicações ubíquas sensíveis ao contexto;
- Estabelecer atividades com diretrizes para o desenvolvimento de artefatos reutilizáveis e para a construção de interfaces ricas adaptativas com reúso desses artefatos;
- Disponibilizar os artefatos construídos para reúso no desenvolvimento de interfaces ricas adaptativas, compreendendo: um metamodelo do Domínio de Interfaces Ricas para suporte à modelagem; transformações para geração de código; e adaptadores dinâmicos de conteúdo;
- Definir uma abordagem de adaptação híbrida de interfaces ricas que combine geração de código em tempo de desenvolvimento (construção de versões estáticas da interface) e geração de código em tempo de execução (refinamento dinâmico conforme o perfil do dispositivo).
- Disponibilizar um *framework* que dê suporte à adaptação híbrida, o qual fornece serviços contextualizados de adaptação de conteúdo de interfaces ricas baseados na combinação de perfis de diferentes entidades contextuais presentes num ambiente ubíquo (e.g. dispositivo, usuário, rede de acesso).

1.3 Organização do Trabalho

Esta dissertação está organizada em outros seis capítulos, além deste capítulo introdutório. A Figura 1.1 ilustra a organização geral do trabalho. Os conteúdos de cada capítulo são detalhados a seguir.

- O **Capítulo 2** apresenta uma revisão da literatura sobre interfaces ricas na Web 2.0, desenvolvimento dirigido a modelos, aplicações sensíveis ao

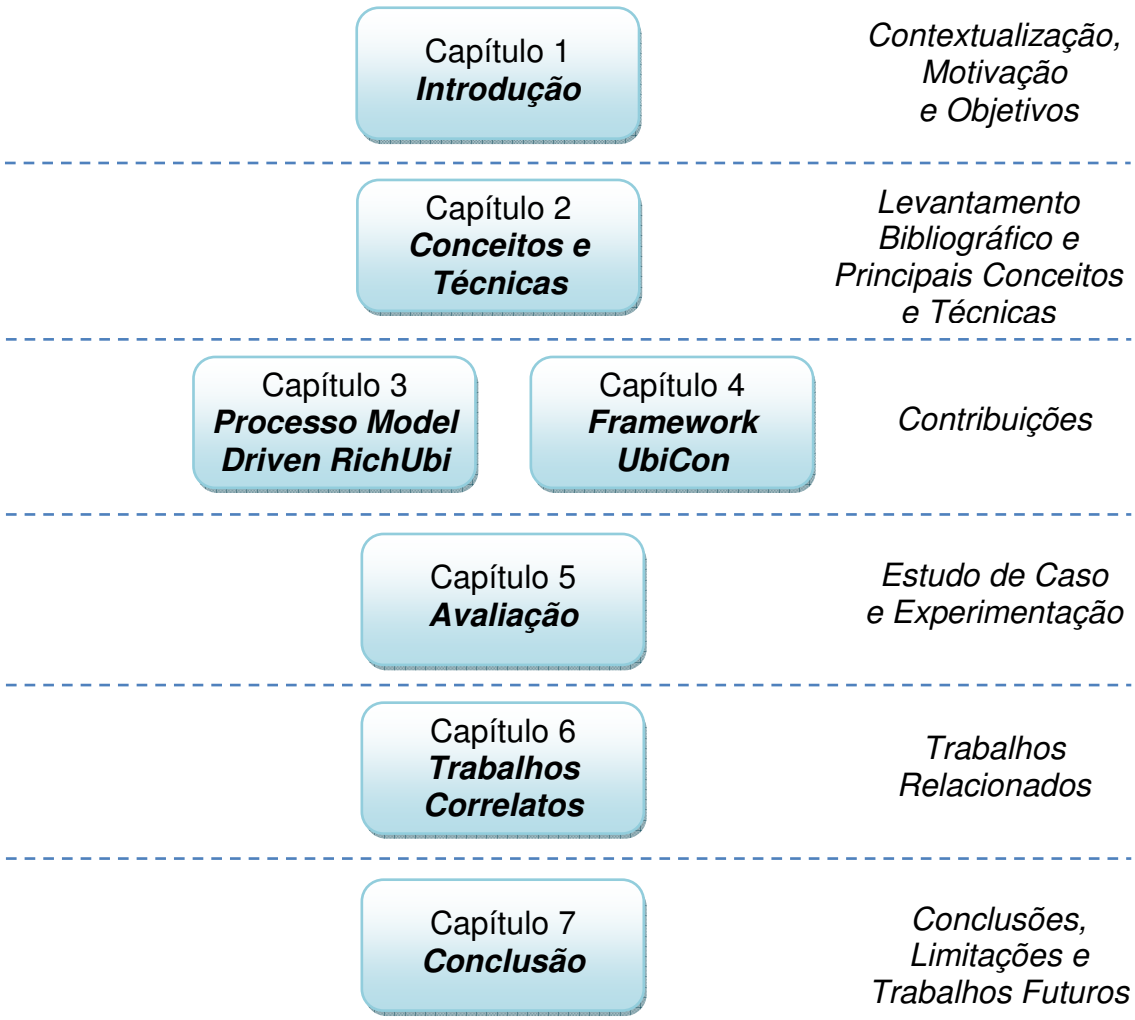


Figura 1.1 – Organização da Dissertação.

contexto e estratégias de adaptação de interfaces, discutindo os principais conceitos e técnicas envolvidos no desenvolvimento deste trabalho;

- O **Capítulo 3** apresenta o processo *Model Driven RichUbi* proposto neste trabalho para suporte à construção de interfaces ricas de aplicações ubíquas sensíveis ao contexto. As principais ideias em torno do processo são discutidas e suas atividades são detalhadas, apresentando as entradas, saídas, controles e os mecanismos que apoiam os desenvolvedores na construção de suas aplicações;
- O **Capítulo 4** apresenta o desenvolvimento de um *framework* para suporte ao processo, chamado *Ubiquitous Context Framework* (*UbiCon)*, que encapsula as funcionalidades de aquisição e manipulação do contexto e fornece serviços contextualizados de adaptação de conteúdo de interfaces ricas de forma híbrida;
- O **Capítulo 5** aborda a avaliação da proposta.

- O **Capítulo 6** discute alguns trabalhos encontrados na literatura que relacionam-se ao trabalho apresentado nesta dissertação, apresentando, também, uma comparação entre os mesmos.
- O **Capítulo 7**, por fim, apresenta algumas conclusões, incluindo as contribuições e limitações deste trabalho, além de possibilidades de trabalhos futuros.

Capítulo 2

CONCEITOS E TÉCNICAS

2.1 Considerações Iniciais

Este capítulo apresenta os principais conceitos e técnicas nos quais este trabalho se baseia. Considerando que o processo proposto objetiva apoiar o desenvolvimento de interfaces ricas adaptativas de aplicações interativas, a Seção 2.2 apresenta os conceitos relacionados a Web 2.0 e interfaces ricas. As Seções 2.3 a 2.5 introduzem, respectivamente, as concepções de desenvolvimento dirigido a modelos, sensibilidade ao contexto e estratégias de adaptação de interfaces, amplamente empregadas neste trabalho.

2.2 Web 2.0 e Interfaces Ricas

A Web está, sem dúvida, entre as grandes aplicações de sucesso surgidas no fim do milênio passado. Quando o navegador *Mosaic* foi apresentado em 1993, a Web teve uma explosão de popularidade. Nos anos que sucederam, ela continuou a crescer de forma intensa, atraindo muitas companhias e investidores em busca de rendimentos promissores, período que ficou conhecido como “bolha ponto-com”. Em 2001, muitas dessas empresas entraram em colapso financeiro, o que resultou no “estouro da bolha”, marcando o fim daquele período e uma virada na forma de encarar a Web. Percebeu-se, então, que havia uma mudança no modo com que as

pessoas e as empresas, que haviam sobrevivido à crise, estavam usando a Web e desenvolvendo aplicações nela baseadas. O termo Web 2.0 foi criado para referir-se a essa aparente mudança e popularizou-se pela conferência *O'Reilly Media Web 2.0 Summit*, inaugurada em 2004 (DEITEL & DEITEL, 2009; O'REILLY, 2005).

2.2.1 Princípios da Web 2.0

Geralmente, as empresas que desenvolvem aplicações para Web 2.0 usam a Web como uma plataforma parar criar sites colaborativos, baseados em comunidades, como os sites de redes sociais, *blogs* e *wikis* (NORRIE, 2008). Conforme ilustra a Figura 2.1, a ideia é fazer com que o ambiente *on-line* se torne mais dinâmico e que os usuários colaborem para a organização do conteúdo, diferentemente da Web tradicional (Web 1.0) onde o conteúdo era produzido apenas pelo *Webmaster* e disponibilizado aos usuários.

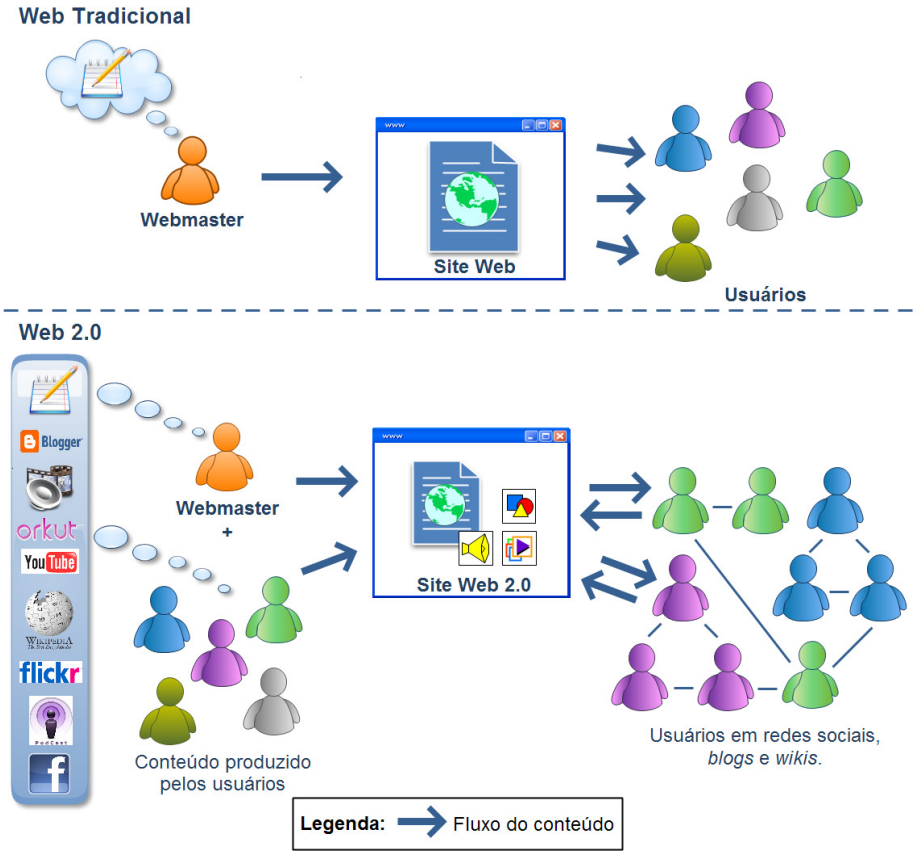


Figura 2.1 – Web tradicional *versus* Web 2.0.

A Web 2.0 engloba um conjunto de princípios. Aplicações classificadas como de Web 2.0 não necessariamente atendem a todos esses princípios simultaneamente, mas a um subconjunto deles. A Figura 2.2 ilustra os princípios da Web 2.0.

Figura 2.2 – Princípios da Web 2.0 (adaptado de GASPAR et al., 2009).

De acordo com a Figura 2.2, os sete princípios que giram em torno das aplicações de Web 2.0 são (O'REILLY, 2005):

- **A Web como plataforma:** A Web tornou-se a plataforma de execução de diversas aplicações tidas tradicionalmente como *desktop*, tais como jogos, planilhas, editores de texto, agendas, clientes de *email*, até Sistemas de Planejamento de Recursos Corporativos (*ERP – Enterprise Resource Planning*). Neste sentido, a Web deixou de ser principalmente um repositório de informações para tornar-se uma plataforma de apoio para realização de tarefas complexas (GASPAR et al., 2009);
- **Uso da inteligência coletiva**: O princípio central das empresas sobreviventes à crise de 2001 é o de armazenar e usar informações produzidas coletivamente. Neste contexto, há uma evolução na forma de participação do usuário. Mais do que apenas consumir a informação disponibilizada, o usuário agora passa a fazer parte do processo de elaboração do conteúdo. As informações produzidas pelos usuários são usadas pelas aplicações de Web 2.0 para melhorar a qualidade do seu serviço à medida que essas informações são coletadas. Por exemplo, a *Wikipédia*[1] fica mais completa a cada edição; redes sociais, como o *Orkut*[2]

[1] http://www.wikipedia.org/.

e o *Facebook*[3], melhoram a cada novo usuário; as ferramentas de idioma da *Google*[4] ficam mais otimizadas a cada sugestão de tradução feita pelos usuários, etc;

- **Dado é o mais importante (*Data is the next "Intel inside"*)**: A informação é hoje um dos ativos mais preciosos no domínio de tecnologia da informação. As informações ficam ainda mais valorizadas se combinadas aos dados coletados de uma massa de usuários. O site da *Amazon*[5], por exemplo, usa um catálogo de publicações *International Standard Book Number (ISBN)* fornecido por uma determinada empresa. No entanto, ao longo de anos de uso, os usuários do *Amazon* foram incluindo comentários, revisões e notas às publicações do catálogo. Hoje o maior vendedor de catálogo de publicação não é a empresa que originalmente forneceu catálogos de ISBN à *Amazon*, e sim a própria *Amazon* (GASPAR et al., 2009);
- **Betas perpétuos e desenvolvimento ágil:** Aplicações de Web 2.0 são disponibilizadas como serviços e não em versões. O usuário não instala um software em seu dispositivo, ao invés disso ele acessa um serviço pela Internet. Com isso, as aplicações estão em contínuo desenvolvimento e novas funcionalidades podem ser disponibilizadas com muito mais frequência em questões de meses, semanas e até mesmo dias ou horas. Se os usuários não responderem bem às novas funcionalidades, as mesmas são retiradas e outras são adicionadas. Dessa forma, aplicações podem possuir a característica de "*Beta"* indefinidamente. Por exemplo, o *Gmail*[6] da *Google* desde seu lançamento possui o rótulo de *Beta*, uma vez que está em constante aprimoramento. Com o *Google Labs*[7], funcionalidades experimentais podem ser adicionadas e testadas pelos usuários, cujo *feedback* é usado pela Google para aprimorar as funcionalidades disponibilizadas;
- **Experiência rica do usuário:** As interfaces gráficas das aplicações de Web 2.0 vão além do *Hypertext Markup Language (HTML)* básico e contam

[2] http://www.orkut.com/.

[3] http://www.facebook.com/.

[4] http://www.google.com.br/language_tools/.

[5] http://www.amazon.com/.

[6] http://www.gmail.com/.

[7] http://www.googlelabs.com/.

agora com recursos como *arrastar-e-soltar*, menus deslizantes, mapas, componentes de captura e exibição de vídeos, etc. Com a evolução da tecnologia, parte do processamento pode ser feita no cliente e, com a comunicação assíncrona com o servidor introduzida com *Assynchronous JAvaScript and XML (AJAX)* (ZAKAS et al., 2007), é possível desenvolver uma interface muito mais rica para o usuário, que oferece a sensibilidade, recursos e funcionalidades que se aproximam das aplicações *desktop* (*webtops*) (DEITEL & DEITEL, 2009);

- **Modelos leves de programação:** O modelo de programação de aplicações de Web 2.0 é voltado para a simplicidade, com uso de protocolos de comunicação simples como o *Representational State Transfer (REST)* (FIELDING & TAYLOR, 2000), linguagens de programação menos complexas e dinâmicas como *Ruby*[8] e *PHP*[9], *frameworks* que favorecem a produtividade como o *Grails*[10], metodologias de desenvolvimento ágeis e composição de serviços (*mashups*). Esses serviços devem ser modulares e fracamente acoplados, de modo que um grande número de aplicações possam reutilizá-los. Por exemplo, o *Google Maps*[11] possibilita inserir um mapa com marcadores dentro de uma aplicação Web, de forma simples e desacoplada; e
- **Software em mais de um dispositivo:** A Web 2.0 não se limita à plataforma PC. As aplicações podem ser executadas em diversos tipos de dispositivos com diferentes sistemas operacionais. Qualquer pessoa no mundo que tenha um navegador Web disponível em seu dispositivo pode acessar a aplicação, seja de um celular ou *desktop*, seja utilizando Linux ou Windows, ou qualquer outra plataforma. A plataforma é a Web e não o sistema do usuário. Na verdade, toda aplicação Web já satisfaz este requisito, uma vez que requerem apenas um servidor e um cliente com um navegador Web. Porém, na Web 2.0 este conceito vai mais além, de forma que as aplicações não se restringem somente à arquitetura cliente-servidor

[8] http://www.ruby-lang.org/pt/.

[9] http://www.php.net/.

[10] http://www.grails.org/.

[11] http://maps.google.com/.

convencional, mas podem também possuir outras arquiteturas, como a arquitetura *Peer-to-Peer (P2P)* (GASPAR et al., 2009).

2.2.2 Aplicações de Internet Ricas

No contexto da Web 2.0, as chamadas Aplicações de Internet Ricas (*RIAs – Rich Internet Applications*) são aplicações Web que se aproximam da aparência, comportamento e usabilidade de aplicações *desktop* (DEITEL & DEITEL, 2009; NORRIE, 2008). O avanço das aplicações Web foi propiciado principalmente pela evolução dos meios de transmissão de dados para acesso à Internet e pela evolução dos navegadores Web com relação ao suporte para renderização gráfica, captura de câmera, atualizações assíncronas de informações, suporte a *plugins*, dentre outras funcionalidades (GASPAR et al., 2009).

As RIAs possuem dois atributos fundamentais: **desempenho** e **interface rica** (DEITEL & DEITEL, 2009). Dentre as características das RIAs que englobam esses atributos destacam-se:

- **Componentes de interface rica:** Recursos como painéis de efeito sanfona, componentes de *arrastar-e-soltar*, componentes de captura e exibição de vídeos, mapas, campos dinâmicos de formulários, planilhas e editores de textos *on-line* são exemplos de elementos de interface rica que proporcionam uma interface mais atraente e com maior interatividade, melhorando a experiência geral dos usuários com a aplicação (DEITEL & DEITEL, 2009). Alguns *frameworks* e bibliotecas, como o *Google Web Toolkit (GWT)*[12] e a biblioteca *jQuery*[13], fornecem esses elementos como componentes que podem ser reutilizados em diferentes aplicações, o que facilita o desenvolvimento de RIAs. A Figura 2.3 apresenta um exemplo de interface rica de uma aplicação de Web 2.0 com alguns componentes de interface rica, como painel de abas, campo de texto com funcionalidade *auto-completar*, *player* de música, dentre outros;
- **Processamento distribuído entre cliente e servidor:** A robustez oferecida pelos navegadores Web atuais, como o suporte para *plugins* e

[12] http://code.google.com/intl/pt-BR/webtoolkit/.

[13] http://jqueryui.com/.

Figura 2.3 – Exemplo de interface rica.

JavaScript, permite que a carga de processamento possa ser compartilhada entre o servidor e o cliente. Dessa maneira, uma vez que parte do processamento pode ser executado na máquina do usuário, os servidores podem distribuir sua capacidade de processamento para atender a um número maior clientes (GASPAR et al., 2009); e

- **Comunicação assíncrona:** Nas aplicações Web tradicionais é empregado um modelo de *comunicação síncrona*, onde o cliente realiza uma requisição e aguarda o resultado do processamento retornado pelo servidor. Ao receber a resposta, toda a página Web do cliente é atualizada e então o usuário pode continuar a interagir com a aplicação (Figura 2.4(a)). Nas RIAs a comunicação entre o cliente e o servidor ocorre de forma *assíncrona*, quebrando o ciclo "clique-aguarde-recarregue". Assim, enquanto as requisições do cliente são processadas pelo servidor, o usuário pode continuar interagindo com a aplicação normalmente. Quando a resposta é retornada pelo servidor ao cliente, apenas as partes da página Web que contém informação nova são recarregadas, ao invés da página toda (Figura 2.4(b)). Essa capacidade é possível graças ao uso de AJAX, o qual combina tecnologias

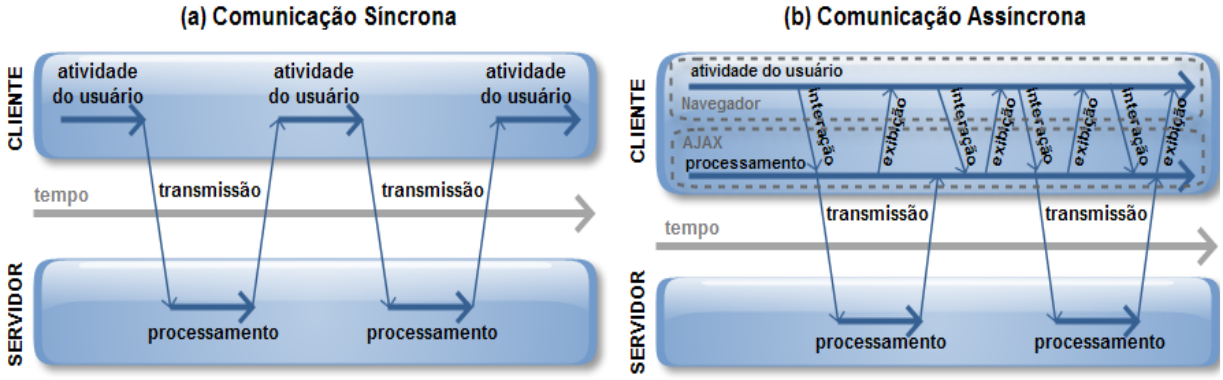


Figura 2.4 – Modelo de comunicação síncrona e assíncrona.

existentes como *eXtensible Hypertext Markup Language (XHTML)*, *Cascading Style Sheets (CSS)*, *Document Object Model (DOM)*, *eXtensible Markup Language (XML)* e *JavaScript*, e utiliza o objeto *XMLHttpRequest*[14] para atualizar partes específicas das páginas Web, permitindo que os usuários continuem interagindo com a aplicação enquanto o servidor processa as requisições (DEITEL & DEITEL, 2009; HOLDENER, 2008).

2.3 Desenvolvimento de Interfaces Dirigido a Modelos

A principal ideia em torno do Desenvolvimento de Interfaces Dirigido a Modelos (*Model-Based User Interface Development – MB-UID*) é identificar abstrações úteis que permitem descrever a interface independentemente das tecnologias de implementação, permitindo que os desenvolvedores se concentrem na definição conceitual da camada de apresentação da aplicação ao invés dos detalhes técnicos de implementação (EISENSTEIN et al., 2000).

Para uma melhor compreensão de como a abordagem dirigida a modelos é aplicada ao desenvolvimento de interfaces, faz-se necessário compreender os conceitos relacionados ao Desenvolvimento Dirigido a Modelos (MDD)[15] de uma forma geral. Esses conceitos, apresentados nas próximas seções, abrangem tópicos específicos, como os componentes que constituem o MDD, e tópicos abrangentes, como Linguagens Específicas de Domínio (*Domain Specific Languages – DSLs*) e metamodelagem.

[14] http://www.w3.org/TR/XMLHttpRequest/.

[15] Outros termos encontrados na literatura, tais como MDE (*Model-Driven Engineering*) (SCHMIDT, 2006), MDSD (*Model-Driven Software Development*) (VÖLTER & GROHER, 2007), ou simplesmente MD* (VÖLTER, 2008), são usados como sinônimos de MDD.

2.3.1 Conceitos do Desenvolvimento Dirigido a Modelos

As abordagens de MDD propõem que a modelagem e transformação de modelos em artefatos de implementação são a melhor base para o desenvolvimento e manutenção de sistemas de software ao invés de codificação pura (FRANCE & RUMPE, 2007; MELLOR et al., 2004; MELLOR et al., 2003; SCHMIDT, 2006). Um modelo de um sistema/software, nesse contexto, é uma descrição ou especificação abstrata desse sistema/software, bem como de seu ambiente, para um determinado propósito, sendo geralmente representado como uma combinação de elementos gráficos e textuais (OMG, 2003).

Conforme ilustrado na Figura 2.5, no MDD o engenheiro de software não necessita interagir manualmente com *todo* o código-fonte, mas pode concentrar-se em modelos de maior nível de abstração, ficando protegido das complexidades requeridas para implementação nas diferentes plataformas (FRANCE & RUMPE, 2007; LUCRÉDIO, 2009). Mecanismos de transformação são empregados para gerar código e outros artefatos de implementação de forma (semi)automática a partir dos modelos. Neste cenário, os modelos não apenas guiam as tarefas de desenvolvimento e manutenção, mas são parte integrante do software assim como o código-fonte, servindo como entrada para ferramentas de geração de código e reduzindo os esforços dos desenvolvedores (BITTAR et al., 2009; KLEPPE et al., 2003).

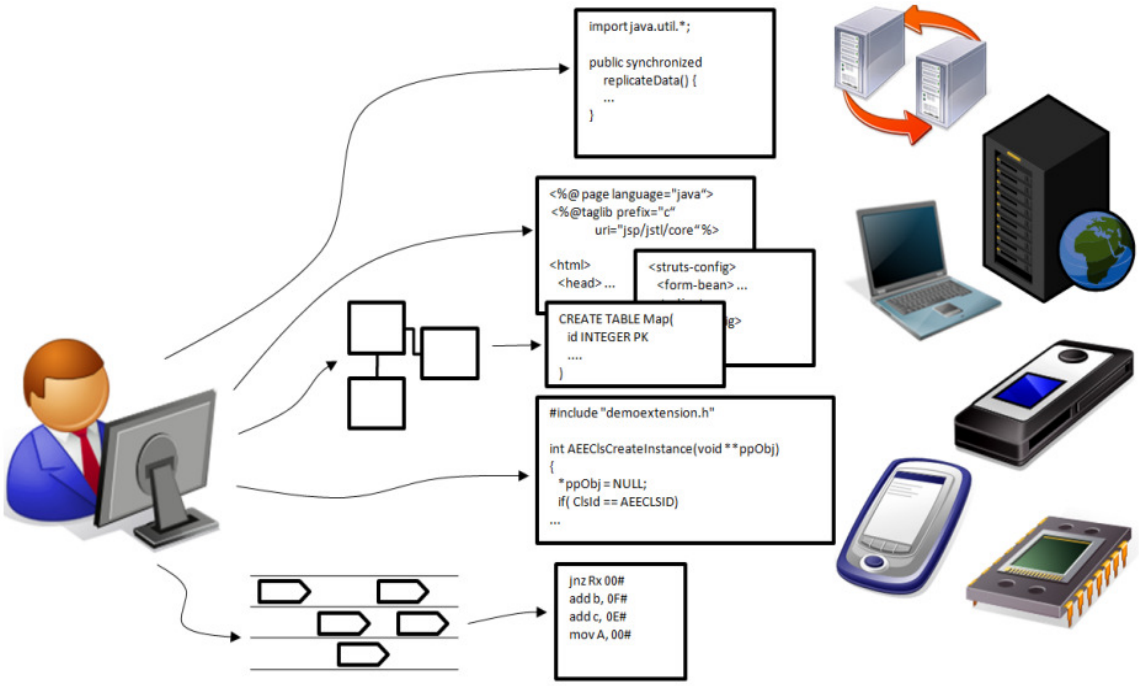


Figura 2.5 – Ilustração do processo de criação de software no MDD (LUCRÉDIO, 2009).

No desenvolvimento de software não dirigido a modelos, artefatos de alto nível (e.g. modelos, diagramas) são produzidos antes da codificação e costumam ser úteis apenas para facilitar a análise de um problema nas etapas iniciais do ciclo de desenvolvimento. Porém, à medida que o desenvolvimento avança, os modelos logo tornam-se inúteis devido ao fato de que quando mudanças são necessárias os desenvolvedores usualmente as fazem diretamente no código, principalmente por restrições de tempo. Dessa forma, os modelos rapidamente perdem a consistência, tornando-se incapazes de refletir a realidade, o que faz com que o tempo e os esforços gastos na construção desses artefatos não sejam diretamente aproveitados na produção do software (BITTAR et al., 2009; LUCRÉDIO, 2009).

Por outro lado, no MDD o foco nos modelos busca simplificar o processo de desenvolvimento, visto que apenas o uso de tecnologias centradas em código-fonte requer um esforço maior para construir o software (FRANCE & RUMPE, 2007; KLEPPE et al., 2003). Apesar de encapsular o conhecimento de uma forma concreta retratando computacionalmente as regras de negócio que são importantes e, por vezes, vitais para o sucesso da organização, o código-fonte possui uma linguagem que é demasiadamente densa e codificada, de tal maneira que é difícil identificar, extrair e reutilizar o conhecimento apenas pela leitura desse código. Além disso, o conhecimento expresso no código-fonte está saturado por trechos altamente associados à plataforma de implementação, de maneira que a reutilização de uma determinada lógica de negócio em uma plataforma ou linguagem diferentes requer um trabalho meticuloso de engenharia. Os modelos, por outro lado, são formas mais intuitivas para representação do conhecimento e menos dependentes do código-fonte (LUCRÉDIO, 2009), de forma que podem ser reutilizados facilmente em diferentes projetos.

A Tabela 2.1 apresenta uma compilação das principais vantagens advindas da prática de MDD (BAHNOT et al., 2005; DEURSEN & KLINT, 1998; KLEPPE et al., 2003; LUCRÉDIO, 2009; MERNIK et al., 2005).

Em síntese, as vantagens do MDD resultam da capacidade de evitar que o desenvolvedor precise executar tarefas repetitivas necessárias para a transformação de modelos para código executável, o que é alcançado por meio da automação dessas transformações. O tempo despendido nessas tarefas, que no desenvolvimento convencional, i.e., não dirigido a modelos, desestimula a execução do ciclo completo dos requisitos aos testes, é significativamente reduzido graças às transformações automatizadas, o que faz com que mesmo atividades urgentes,

como correção de erros, possam ser executadas sem deixar os modelos inconsistentes, mantendo-os atualizados constantemente.

Tabela 2.1 – Principais vantagens da prática de MDD.

Vantagem	Caracterização
Produtividade	– Automatização da geração de código a partir de modelos através do uso de ferramentas de transformação, incentivando os desenvolvedores a retornarem às etapas iniciais de requisitos e análise; – Tarefas repetitivas de codificação são implementadas nas transformações, poupando tempo e esforço que podem ser despendidos em tarefas mais importantes.
Portabilidade	– A partir de um mesmo modelo pode-se gerar código para diferentes plataformas.
Interoperabilidade	– Cada parte do modelo pode ser transformado em código para uma plataforma diferente, o que resulta em um software que pode ser executado em um ambiente heterogêneo, mas que mantém sua funcionalidade global.
Manutenção e documentação	– Alterações são realizadas diretamente nos modelos, mantendo-os consistentes com o código-fonte, o qual é gerado automaticamente a partir de transformações aplicadas nesses modelos; – A documentação permanece atualizada, o que facilita as tarefas de manutenção.
Comunicação	– Uma vez que os modelos são mais abstratos que o código-fonte, o que não exige conhecimento técnico associado à plataforma de implementação para sua compreensão, os especialistas do domínio podem utilizar diretamente os modelos para identificar mais facilmente as questões associadas ao negócio; – Os especialistas de tecnologia da informação podem identificar os elementos técnicos usando os mesmos modelos.
Reúso	– O reúso é feito em nível de modelos ao invés de em nível de código-fonte; – O código pode ser automaticamente regenerado para novos contextos através de ferramentas de transformação apropriadas.
Verificação e Otimizações	– Os modelos facilitam a análise por verificadores semânticos, conforme a sintaxe de seu metamodelo, e a execução de otimizações automáticas; – Minimização da ocorrência de erros semânticos, o que fornece implementações mais eficientes.
Correção	– Ferramentas de transformação evitam a introdução de erros acidentais, tais como erros de digitação e de sintaxe; – Erros conceituais podem ser identificados em um nível mais alto de abstração.

2.3.1.1 Elementos do MDD

Para que o MDD ocorra de forma efetiva, alguns elementos devem estar presentes no processo de desenvolvimento, conforme ilustrado na Figura 2.6.

A criação dos modelos é feita através de uma **ferramenta de modelagem**. Essa ferramenta deve ser intuitiva, de forma que o engenheiro de software produza facilmente os modelos que descrevem os conceitos do domínio de aplicação. Os modelos produzidos devem ser semanticamente corretos e completos, de forma que um computador seja capaz de processá-los e interpretá-los para geração de artefatos de implementação. Geralmente essas características são implementadas através de uma Linguagem Específica de Domínio (DSL), apresentada nas próximas

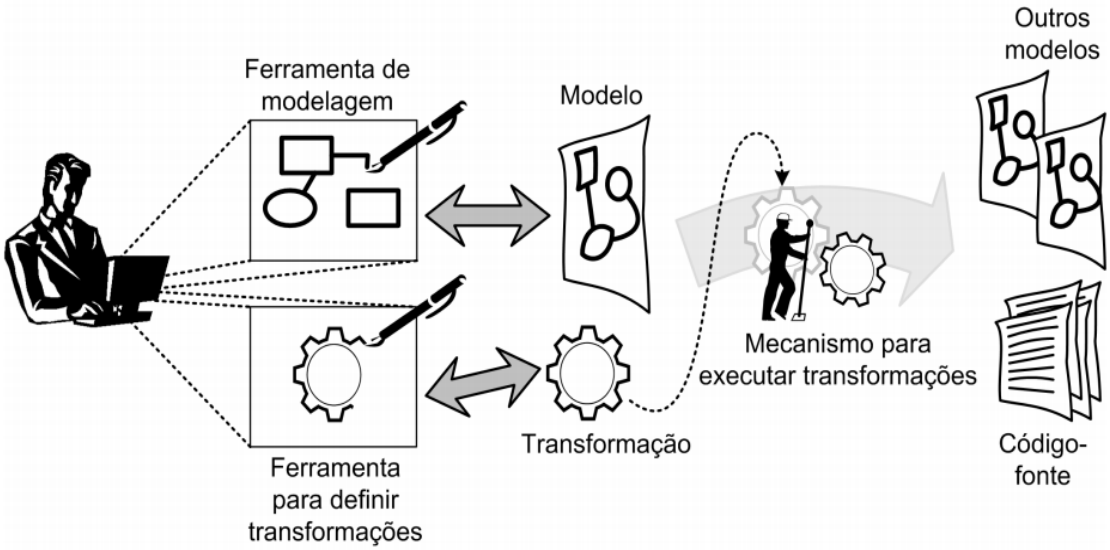


Figura 2.6 – Principais elementos do MDD (LUCRÉDIO, 2009).

subseções. **Ferramentas para definir transformações** permitem que o engenheiro de software construa regras de transformação modelo-para-modelo e modelo-para-texto (e.g. código-fonte, arquivos de configuração). Um **mecanismo** é usado para **executar as transformações** produzidas, recebendo como entrada os modelos criados por meio das ferramentas de modelagem e gerando como saída outros modelos ou código-fonte de acordo com as regras definidas nas transformações (LUCRÉDIO, 2009).

2.3.1.2 Metamodelagem

Para dar suporte a diferentes linguagens de modelagem, ajudar a garantir que os modelos construídos estejam semanticamente corretos e completos, bem como possibilitar a definição e execução de transformações, as principais abordagens de MDD baseiam-se no conceito de metamodelagem, que é a atividade onde se constroem metamodelos (LUCRÉDIO, 2009).

Um metamodelo trata-se de um modelo que contém elementos para a descrição de outros modelos. Embora ambos – modelo e metamodelo – constituam-se como modelos, um é expresso em termos do outro, ou seja, um modelo é uma instância ou está em conformidade com o seu respectivo metamodelo (GRONBACK, 2009).

No âmbito do MDD, os metamodelos desempenham um importante papel, uma vez que: (i) definem a sintaxe abstrata de linguagens de modelagem; (ii) facilitam o entendimento dos conceitos envolvidos durante a definição de transformações; e (iii) são empregados por mecanismos que executam

transformações para mapear um modelo de entrada para outro modelo ou código.

A Figura 2.7 apresenta uma arquitetura de metamodelagem em quatro camadas amplamente utilizada pela comunidade de desenvolvimento (OMG, 2006).

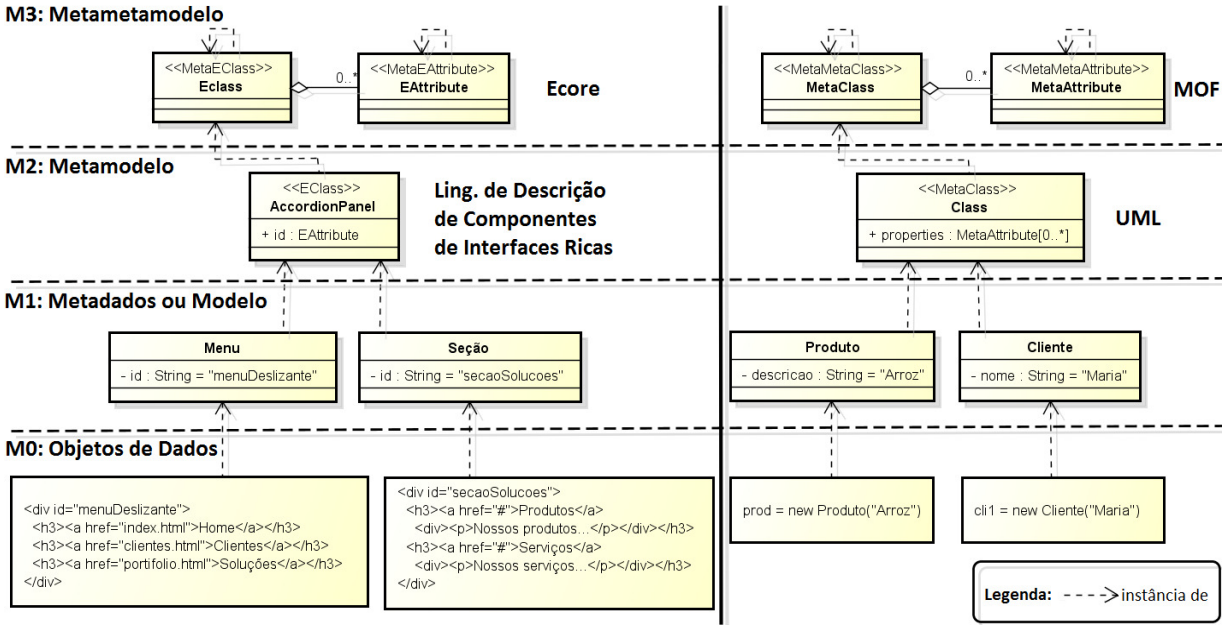


Figura 2.7 – Arquitetura clássica de metamodelagem.

A camada de **Metametamodelo (M3)** define uma linguagem para descrever metamodelos. Não existe uma camada superior a esta, pois usualmente um meta-metamodelo é uma instância de si próprio. Por exemplo, na Figura 2.7 o *Ecore* é a linguagem de metamodelagem (ou metametamodelo) utilizada pelo *Eclipse Modeling Framework (EMF)* (STEINBERG et al., 2008) que especifica as construções básicas para a criação de metamodelos nesse *framework* (e.g. *Eclass*, *EAttribute*). Já o *Meta Object Facility (MOF)* (OMG, 2006) é o metametamodelo definido pelo consórcio *Object Management Group (OMG)*[16]*,* o qual serve de base para a definição de todas as linguagens de modelagem utilizadas na *Model Driven Architecture (MDA)* (OMG, 2003).

Na camada de **Metamodelo (M2)** é definida uma linguagem para criação de modelos. Por exemplo, na Figura 2.7 a *Linguagem de Descrição de Componentes de Interfaces Ricas* trata-se do metamodelo que define os elementos que permitem criar modelos nessa linguagem específica. A especificação da *Unified Modeling Language (UML)* (OMG, 2009) também é um exemplo de metamodelo que define os elementos para a criação dos diagramas dessa linguagem.

A camada de **Metadados ou Modelo (M1)** especifica uma linguagem para descrever informações do domínio de aplicação. São os modelos propriamente ditos.

[16] http://www.omg.org/.

Por exemplo, conforme ilustrado na Figura 2.7, pode-se criar um modelo de interface rica que define um *Menu* e uma *Seção* de conteúdo dentro da interface tendo como base a metaclasse *AccordionPanel* da *Linguagem de Descrição de Componentes de Interfaces Ricas*. Em um diagrama de classes UML que especifica os requisitos de uma aplicação, pode-se criar classes para descrever os produtos (*Produto*) e clientes (*Cliente*) do domínio da aplicação com base na metaclasse *Class* da especificação da UML.

Por fim, a camada de **Objetos de Dados (M0)** engloba as instâncias dos modelos que correspondem aos dados efetivamente usados na aplicação. No exemplo da Figura 2.7, o *Menu* modelado na *Linguagem de Descrição de Componentes de Interfaces Ricas* pode ser mapeado para XHTML. Já a classe UML nomeada como *Produto* pode ser instanciada em um objeto Java (SUN, 2010b) chamado *prod* que representa o produto “Arroz”, por exemplo.

2.3.1.3 Linguagens Específicas de Domínio

Uma Linguagem Específica de Domínio (DSL) é uma linguagem projetada para ser útil para um conjunto específico de tarefas em um dado domínio (GRONBACK, 2009; SADILEK, 2008). As DSLs se baseiam na noção de “domínio”, um conceito tido como vago e com significados distintos em diferentes áreas. No entanto, esse conceito de domínio pode ser mais bem definido de acordo com determinados critérios. Para a compreensão dos critérios que caracterizam o domínio de que trata uma DSL, a Figura 2.8 apresenta um exemplo de uma situação comumente vivenciada durante a Análise de Sistemas.

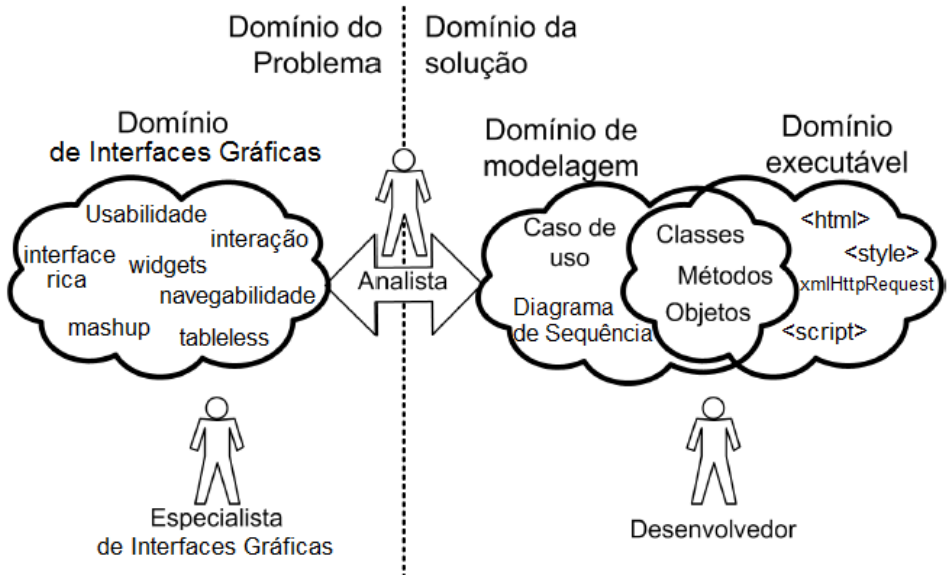


Figura 2.8 – Exemplo de uma situação comum na Análise de Sistemas (adaptado de LUCRÉDIO, 2009).

Durante o desenvolvimento de sistemas, o conhecimento de um especialista de alguma área (no exemplo, um especialista de interfaces gráficas) é capturado pelo analista de sistemas, o qual traduz os termos e conceitos familiares ao especialista de interfaces gráficas para termos e conceitos de computação conhecidos pelo desenvolvedor. Neste contexto, o domínio refere-se a uma determinada área de competência e conhecimento que possui terminologia e conceitos particulares. No exemplo, os termos associados às interfaces gráficas, tais como "Usabilidade", "interface rica" e "widgets" estão dentro da área de conhecimento do especialista de interfaces gráficas, o que configura o **Domínio de Interfaces Gráficas**.

Os termos e conceitos relacionados ao desenvolvimento de software, tais como "Casos de uso", "Classes", "Métodos", as marcações "<html>", "<script>", "<style>", o objeto *xmlHttpRequest*, dentre outros, fazem parte da área de competência e conhecimento do desenvolvedor. De acordo com a Figura 2.8, alguns desses termos enquadram-se no **domínio de modelagem**, outros no **domínio executável**, e outros ainda enquadram-se em ambos (LUCRÉDIO, 2009).

Conforme observa-se na Figura 2.8, uma outra distinção pode ser feita para o termo domínio. No âmbito do desenvolvimento de sistemas, o domínio para o qual se está construindo um sistema é chamado de **domínio do problema**, uma vez que trata-se do problema que se quer solucionar. Por outro lado, o domínio no qual se está construindo o sistema é chamado **domínio da solução**, visto que este engloba a solução computacional que está sendo desenvolvida (LUCRÉDIO, 2009). Essa mesma classificação, por vezes, aparece também como **domínio vertical** e **domínio horizontal**, respectivamente (KELLY & TOLVANEN, 2008).

A partir dessas definições, é possível afirmar que uma DSL é uma linguagem qualquer que representa os termos e conceitos do domínio (SADILEK, 2008). Por exemplo, XHTML e *JavaScript* são linguagens de um domínio executável, UML é uma linguagem de um domínio de modelagem, e assim por diante. Contudo, ultimamente quando se fala em DSLs, normalmente está se referindo a linguagens específicas de um **domínio do problema** (LUCRÉDIO, 2009).

Neste sentido, tem-se a seguinte definição (DEURSEN et al., 2000):

> *"Uma linguagem específica de domínio é uma linguagem de programação ou linguagem de especificação executável que oferece, através de notações e*

> *abstrações apropriadas, poder expressivo com foco em, e usualmente restrito a, um* ***domínio do problema*** *particular".*

Assim, com base nessa definição, uma linguagem projetada para a descrição de componentes de interface rica, por exemplo, é uma DSL, enquanto que XHTML, *JavaScript* e UML não o são.

Uma DSL pode ser textual (permitindo especificar programas) ou visual (permitindo especificar modelos ou diagramas), e normalmente possui uma *sintaxe abstrata*, uma *sintaxe concreta* e uma *sintaxe de serialização*.

A **sintaxe abstrata** define os conceitos do domínio, bem como as relações e as restrições que se aplicam a esses conceitos. Em linguagens de modelagem, por exemplo, a sintaxe abstrata corresponde ao metamodelo que fornece a estrutura para os modelos que podem ser criados (GRONBACK, 2009; GUIZZARDI et al., 2002).

A **sintaxe concreta** permite representar os conceitos do domínio na forma de texto (sintaxe concreta textual) ou através de notação de diagramas (sintaxe concreta gráfica). Pode ser constituída de símbolos ou ícones gráficos com características visuais que representam diferentes atributos, como cor, posição, tamanho e forma (GRONBACK, 2009; GUIZZARDI et al., 2002). Por se tratar de uma representação superficial, que usualmente difere da representação interna expressa pela sintaxe abstrata, uma DSL pode possuir múltiplas sintaxes concretas (KLEPPE, 2007).

A **sintaxe de serialização** (GRONBACK, 2009) permite persistir os modelos de forma que possam ser interpretados, processados e intercambiados entre diferentes ferramentas de modelagem. Por exemplo, o *XML Metadata Interchange (XMI)* (OMG, 2007) é um padrão do OMG utilizado tanto na MDA quanto no EMF para representar modelos em formato XML.

A principal característica de uma DSL é o fato de ela possuir seu poder expressivo focado em um domínio de problema. Isto contribui para reduzir os esforços de tradução dos conceitos desse domínio para o domínio da solução, uma vez que os termos da linguagem estão próximos dos termos reais conhecidos pelo especialista do domínio considerado. Dessa maneira, as DSLs são normalmente pequenas, consistindo de um conjunto de abstrações e notações restritos ao domínio considerado, de forma que especialistas desse domínio possam trabalhar nessa linguagem facilmente (LUCRÉDIO, 2009).

A Tabela 2.2 resume as vantagens da utilização de DSLs (DEURSEN et al., 2000).

Tabela 2.2 – Vantagens da utilização de DSLs.

Vantagem	Caracterização
Comunicação	– As soluções são expressas no idioma e no nível de abstração do domínio do problema; – Os especialistas do domínio podem compreender, validar, modificar e até mesmo desenvolver seus próprios programas (em se tratando de uma DSL programática).
Eficiência	– Aumento da produtividade, confiabilidade, manutenibilidade e portabilidade.
Reúso	– Conservação e reutilização do conhecimento sobre o domínio incorporado pela DSL.
Otimização	– Validação e otimização são realizadas no nível do domínio.
Testes	– Facilitação dos testes das aplicações.

2.3.2 MDD Aplicado ao Desenvolvimento de Interfaces

O desenvolvimento de interfaces dirigido a modelos (MB-UID) explora a ideia de utilizar modelos declarativos da interface que permitem definir seus diferentes aspectos de uma forma abstrata, i.e., sem ater-se aos detalhes da plataforma de implementação. O objetivo é facilitar a transformação dos componentes de interação abstratos representados nos modelos em componentes concretos das plataformas-alvo (DA SILVA, 2000; PUERTA & EISENSTEIN, 1999; VELLIS, 2009).

A abordagem dirigida a modelos, portanto, foi introduzida ao desenvolvimento de interfaces para apoiar a especificação e o projeto de sistemas interativos em um nível semântico, contextual e abstrato, como uma alternativa para lidar com as questões de baixo nível de implementação logo nas fases iniciais do ciclo de vida de desenvolvimento. Dessa forma, os projetistas podem concentrar-se na definição conceitual da interface ao invés dos detalhes técnicos de implementação (EISENSTEIN et al., 2000). Como resultado, a crescente complexidade das interfaces com o usuário pode ser mais facilmente gerenciada. Além disso, a arquitetura da interface é simplificada, permitindo uma melhor compreensão e rastreabilidade para manutenções futuras para diferentes contextos de uso (AHMED; ASHRAF, 2006).

No MB-UID, a modelagem de interfaces com o usuário envolve a criação de bases de conhecimento expressas em uma série de modelos que descrevem os vários aspectos da interface, tais como a apresentação, o diálogo, a estrutura de tarefas do usuário, dentre outros (DA SILVA, 2000; EISENSTEIN et al., 2000). Esses modelos, por não estarem relacionados a uma plataforma específica, permitem a reutilização das especificações da interface em diferentes fases de um projeto de aplicação, bem como fornecem uma infraestrutura para a construção de métodos e ferramentas para a geração automática da apresentação final da interface (VIANA & ANDRADE, 2008).

As principais vantagens de se adotar a abordagem dirigida a modelos para o desenvolvimento de interface são listadas na Tabela 2.3 (DA SILVA, 2000).

Tabela 2.3 – Vantagens do emprego de MDD no desenvolvimento de interfaces.

Vantagem	Caracterização
Compreensão do domínio	– Modelos declarativos fornecem uma descrição mais abstrata da interface do que o código-fonte.
Produtividade e reúso	– Modelos facilitam a criação de métodos para o projeto e implementação de interfaces em uma forma sistemática, uma vez que fornecem capacidades para: (1) modelar as interfaces em diferentes níveis de abstração; (2) refinar os modelos incrementalmente; e (3) reutilizar as especificações da interface em outros projetos similares.
Automação do processo	– Uma vez que possuem uma sintaxe e semântica expressas pela linguagem que os define, os modelos fornecem a infraestrutura necessária para automatizar tarefas relacionadas ao projeto e implementação de interfaces, como por exemplo, geração de código.

No MB-UID, o projeto de interface é um processo onde se cria e refina modelos. Conforme ilustra a Figura 2.9, ferramentas de modelagem são utilizadas durante o projeto para construir os modelos. Assistentes de modelagem, que desempenham funções como validação dos modelos, podem ser utilizados para apoiar os desenvolvedores. Essas ferramentas usualmente fornecem um ambiente gráfico que facilita a criação dos modelos. Alternativamente, ambientes que permitem a criação de modelos de interface em uma notação textual também podem ser empregados (DA SILVA, 2000).

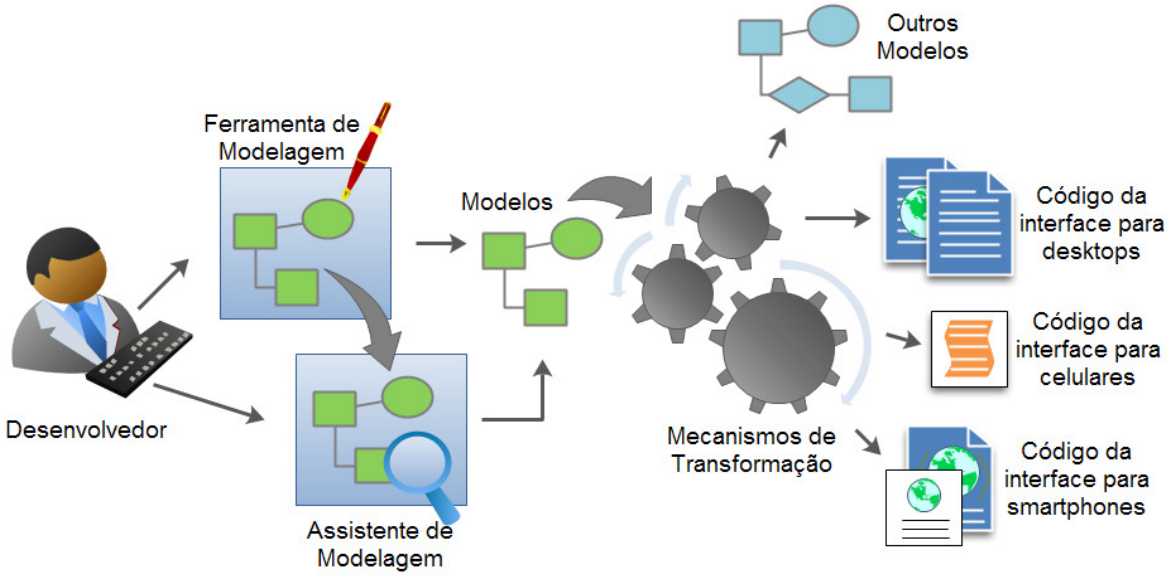


Figura 2.9 – Ilustração do processo de desenvolvimento de interfaces dirigido a modelos.

Neste contexto, os desenvolvedores podem criar, editar e validar os modelos usando as ferramentas disponíveis, até que todos os detalhes relevantes da interface estejam modelados. Além disso, os desenvolvedores podem, a qualquer

momento, alterar os modelos para correção de erros ou inclusão de novos componentes, mesmo depois de iniciada a implementação (DA SILVA, 2000). Conforme também pode ser observado na Figura 2.9, para alcançar a automação no processo de desenvolvimento, mecanismos de transformação modelo-para-modelo (*Model to Model – M2M*) e modelo-para-código (*Model to Code – M2C*) são empregados para gerar automaticamente modelos mais refinados ou código executável para diferentes plataformas.

2.3.3 Modelagem Específica de Domínio

Um modelo de interface com o usuário deve ser expresso em uma determinada linguagem de modelagem. Algumas linguagens, como UIML (ABRAMS et al., 1999), UsiXML (LIMBOURG et al., 2005), XIML (PUERTA & EISENSTEIN, 2002), TERESA XML (MORI et al., 2003), dentre outras, foram criadas com o propósito de permitir a descrição da interface dissociando-a dos detalhes de implementação. No entanto, essas linguagens geralmente são rígidas e não seguem um padrão. Praticamente, essas abordagens diferem-se umas das outras em relação ao poder de expressão, ao conjunto de componentes disponíveis e às técnicas de mapeamento da descrição abstrata para a apresentação final da interface, necessitando, usualmente, de diferentes ferramentas de autoria para produzir as interfaces (CHAVARRIAGA & MACÍAS, 2009).

Uma abordagem que concentra-se no uso de DSLs, por meio do uso de metamodelos, que permitem definir uma linguagem de modelagem mais especializada, como, por exemplo, uma linguagem para descrever componentes de interfaces ricas, colabora para o aumento da produtividade em comparação com linguagens de propósito geral, uma vez que os modelos de alto nível resultantes são mais próximos ao domínio do problema (CHAVARRIAGA & MACÍAS, 2009; SADILEK, 2008). Abordagens que seguem nessa direção são conhecidas como Modelagem Específica de Domínio (*Domain Specific Modeling - DSM*) (KELLY & TOLVANEN, 2008).

Processos baseados na DSM descrevem os conceitos e terminologias específicas de um determinado domínio num nível de abstração que permite a definição de modelos específicos do domínio do problema para apoiar o projeto e a implementação de diferentes aplicações. Seguindo na mesma direção do MDD, a DSM busca elevar o nível de abstração nos processos de desenvolvimento através

do uso da modelagem, porém especificando as soluções diretamente com os conceitos do domínio do problema considerado. Os produtos finais (e.g. aplicação, interfaces, documentação) podem então ser gerados de forma semiautomática a partir dessas definições de alto nível com o uso de geradores de código automatizados (KELLY & TOLVANEN, 2008).

Na prática, na DSM as soluções focam em domínios bem delimitados, uma vez que um foco mais estreito do problema oferece maiores possibilidades para automação, além de facilitar sua definição. Dessa forma, as soluções de DSM são geralmente aplicadas a um produto particular, uma linha de produtos, um ambiente específico ou uma plataforma (LUOMA et al., 2004).

A DSM possui dois objetivos principais, que são (LAFORCADE et al., 2008):

- Aumentar o nível de abstração no processo de desenvolvimento de software para facilitar a implementação das aplicações de um determinado domínio de problema, especificando uma solução em um alto nível de abstração.
- Gerar programas executáveis a partir dos modelos baseados nas especificações do domínio. Usualmente o código gerado é complementado por componentes de software que implementam funcionalidades comuns às aplicações do domínio considerado.

Dessa forma, o emprego de DSM no desenvolvimento de interfaces permite que, a partir da modelagem de uma aplicação e aplicando transformações M2C apropriadas, grande parte do código de suas interfaces possa ser gerado. Por não estarem relacionados a uma plataforma específica, os modelos permitem descrever a interface de uma forma abstrata, o que facilita a transformação dos componentes de interação representados nesses modelos em componentes concretos das plataformas-alvo (VELLIS, 2009).

2.4 Aplicações Sensíveis ao Contexto

A sensibilidade ao contexto (ou ciência de contexto) está relacionada com a adaptação da aplicação de acordo com sua localização de uso, as pessoas ou objetos circundantes, bem como as mudanças que ocorrem nesses objetos ao longo

do tempo (ABOWD et al., 1999; COUTAZ et al., 2005; DEY, 2001; SCHILIT & THEIMER, 1994). Isso inclui, por exemplo, o dispositivo de acesso, o usuário, a rede de acesso, dentre outros. Uma Aplicação Sensível ao Contexto (ASC)[17] é capaz de adaptar suas operações ao contexto atual sem a necessidade de intervenção explícita do usuário, fornecendo informações e/ou serviços que são relevantes para o usuário realizar suas tarefas levando em consideração as informações extraídas do contexto da interação (BALDAUF et al., 2007; DEY, 2001).

Neste sentido, o contexto desempenha um papel primordial para permitir que as aplicações refinem a informação disponível em informação relevante, escolham ações apropriadas a partir de uma lista de possibilidades, ou determinem o melhor método para a disponibilização da informação. O contexto, portanto, guia as variações do comportamento da ASC, enriquecendo a interação do usuário tanto por influenciar recomendações ou por executar adaptações de qualquer tipo (SANTOS, 2008). Por exemplo, entre as ações realizadas por uma ASC pode-se mencionar a *adaptação de conteúdo*, na qual um conteúdo original é convertido em uma série de novos formatos compatíveis com o contexto de entrega. Essa adaptação pode ser, por exemplo, uma transformação, onde a linguagem de marcação de uma interface Web é modificada para ajustar-se mais apropriadamente ao perfil do dispositivo de acesso, e/ou uma *transcodificação*, pela qual imagens são convertidas ou redimensionadas (FORTE et al., 2008).

Contudo, desenvolver uma aplicação sensível ao contexto pode ser uma tarefa complexa e cara. Ao projetar uma aplicação sensível ao contexto deve-se lidar com questões associadas com qual tipo de informação pode ser considerado como contexto, como representar essa informação, como adquirir e processar essa informação uma vez que pode ser oriunda de fontes heterogêneas, e como integrar o uso do contexto na aplicação. Além disso, por ser uma área de estudo recente dentro da Ciência da Computação e ainda não haver um consenso dos conceitos associados ao contexto (DOURISH, 2001; SANTOS, 2008), existe uma dificuldade entre os desenvolvedores para distinguir os aspectos relacionados aos requisitos de

[17] Na literatura também são encontrados outros termos, como *aplicação ciente de contexto* (SCHILIT & THEIMER, 1994), *aplicação orientada a contexto* (DESMET et al., 2007) e *aplicação baseada em contexto* (KASHYAP & SHETH, 1996), para referir-se às aplicações que adaptam seu comportamento e conteúdo de acordo com o contexto. Nesta pesquisa, porém, prevalece o uso do termo aplicação sensível ao contexto (SATO, 2004; VIEIRA et al., 2009) para designar esse tipo de aplicação.

negócio da aplicação daqueles referentes à manipulação do contexto. Essas observações indicam a ausência de suporte de Engenharia de Software (e.g. processos, métodos, ferramentas) que apoie os projetistas de ASCs no desenvolvimento de suas aplicações (VIEIRA et al., 2009, 2011).

2.4.1 Contexto Computacional

Na tentativa de auxiliar os desenvolvedores de software a incluir o conceito de contexto em suas aplicações, muitas definições operacionais para contexto em termos computacionais já foram propostas. Para se ter uma ideia, Bazire e Brézillon (2005) catalogaram mais de 150 definições de contexto e concluíram que sua definição varia fortemente à medida em que se transita entre diferentes domínios. Apesar da variedade de definições, os pesquisadores, em geral, concordam que: o contexto está sempre associado com alguma entidade (e.g. tarefas, agente, interação); o contexto é um conjunto de itens associados a uma entidade (e.g. conceitos, regras, proposições); e um item é considerado como sendo parte do contexto se for útil para apoiar na realização da tarefa que se quer desempenhar (SANTOS, 2008; VIEIRA et al., 2011).

Segundo o *Dicionário Merriam-Webster*[18], contexto é definido como "condições inter-relacionadas nas quais alguma coisa existe ou ocorre". O contexto é o que fundamenta a habilidade de identificar o que é e o que não é relevante em um dado momento (SANTOS, 2008). Uma definição operacional de contexto amplamente referenciada declara que o contexto é qualquer informação que pode ser usada para caracterizar a situação de uma entidade. Essa entidade pode ser uma pessoa, um lugar, ou um objeto que é considerado relevante para a interação do usuário com a aplicação, inclusive eles próprios (DEY, 2001; DEY et al., 2001).

De maneira similar, Brézillon (1999) considera o contexto como um conjunto de condições relevantes e influências que possibilitam a compreensão de uma situação, onde tais condições e influências atuam diretamente sobre entidades do domínio considerado. Além disso, Brézillon & Pomerol (1999) introduzem a noção de *foco*, declarando que contexto não é um conceito isolado, mas que está sempre relacionado a um foco que determina o que deve ser considerado como relevante

[18] http://www.merriam-webster.com/.

em dada situação. Segundo esses autores, o foco pode representar uma tarefa a ser executada ou uma etapa na resolução de um problema ou tomada de decisão.

Uma definição mais recente, derivada das definições acima apresentadas, propõe que para um melhor entendimento do que venha a ser contexto e como utilizá-lo efetivamente nas aplicações, deve-se estabelecer a distinção entre *contexto* e *Elemento Contextual (EC)*. Essa definição declara que um EC é qualquer dado ou informação que permite caracterizar uma entidade em um domínio. Já o contexto da interação entre um agente (humano ou software) e uma aplicação, com *foco* em alguma tarefa, é o conjunto de ECs instanciados que são necessários para apoiar a tarefa a ser executada (VIEIRA et al., 2009). Essa definição torna mais fácil para um desenvolvedor enumerar o contexto de um dado cenário de aplicação. Neste caso, se determinada informação que caracteriza uma entidade em uma interação for útil para apoiar a tarefa a ser executada (foco), então essa informação é um EC, que por sua vez compõe o contexto daquela interação.

Por exemplo, suponha uma aplicação Web que adapta sua interface de acordo com o perfil do dispositivo de acesso para proporcionar ao usuário uma interação amigável, fornecendo interfaces que não se distorcem quando visualizadas em diferentes dispositivos. A tarefa a ser desempenhada, neste caso, é *adaptar a interface da aplicação*. Para este foco, se for considerada a *largura de tela do dispositivo*, percebe-se que tal informação é útil, pois fornece subsídios para redimensionar o conteúdo da página Web de forma que se encaixe na tela do dispositivo do usuário. Logo, esse EC faz parte do contexto atual. Por outro lado, se for considerada a temperatura do ambiente em que o usuário se encontra atualmente, descobre-se que tal informação, apesar de fazer parte da situação, não é relevante para a tarefa em questão e, portanto, não compõe o contexto atual.

2.4.2 Arquitetura de uma Aplicação Sensível ao Contexto

Aplicações sensíveis ao contexto podem ser implementadas de diversas maneiras. A arquitetura da aplicação depende de requisitos e condições especiais, dentre os quais o método de aquisição dos elementos contextuais exerce influência considerável na definição do estilo arquitetural da ASC (BALDAUF et al., 2007).

Chen (2004) apresenta três diferentes abordagens para aquisição dos ECs:

- **Acesso direto às fontes de contexto**: nesta abordagem, a aplicação cliente obtém os elementos contextuais diretamente das fontes de contexto disponíveis. Não há uma camada adicional para tratamento do contexto, de forma que o código para aquisição e processamento dos elementos contextuais fique atrelado ao código das regras de negócio da aplicação;
- **Infraestrutura intermediária:** esta abordagem incorpora uma arquitetura intermediária às ASCs que encapsula a manipulação do contexto, ocultando os detalhes de baixo nível de obtenção dos elementos contextuais de suas fontes de origem. Esta abordagem favorece a extensibilidade, uma vez que o código da aplicação cliente não necessita ser modificado caso haja alterações na forma de aquisição dos ECs.
- **Servidor de contexto:** esta abordagem distribuída estende a arquitetura baseada em infraestrutura intermediária, introduzindo um componente remoto de gerenciamento de acesso às fontes de contexto. A obtenção dos ECs é realizada em um servidor, chamado *servidor de contexto*, para facilitar múltiplos acessos simultâneos. Esta abordagem alivia as aplicações clientes das operações intensivas para obtenção dos ECs de suas fontes de origem, o que é um aspecto importante considerando uma ASC que executa em dispositivos com recursos limitados.

Entre essas abordagens, o uso da infraestrutura intermediária para gerenciar o contexto introduz uma arquitetura em camadas à ASC. Isto possibilita ocultar os detalhes de baixo nível para extração dos ECs de suas fontes de contexto originais em uma camada projetada especificamente para gerenciar o contexto. Ao contrário da abordagem de acesso direto às fontes de contexto, a infraestrutura intermediária, além de encapsular as funcionalidades relacionadas à obtenção do contexto favorecendo a extensibilidade, também possibilita o reúso da camada de gerenciamento de contexto em diferentes aplicações (BALDAUF et al., 2007), uma vez que essa camada pode ser considerada como independente de domínio (SANTOS, 2008).

A Figura 2.10 apresenta um modelo conceitual de arquitetura em camadas para ASCs, chamada *Arquitetura de Contexto* (SANTOS, 2008). A camada de *Gerenciamento de Contexto* exerce o papel da infraestrutura intermediária, sendo posicionada entre as fontes e os consumidores de contexto, de modo a fornecer os ECs adquiridos a partir dessas fontes aos consumidores interessados. Nessa camada,

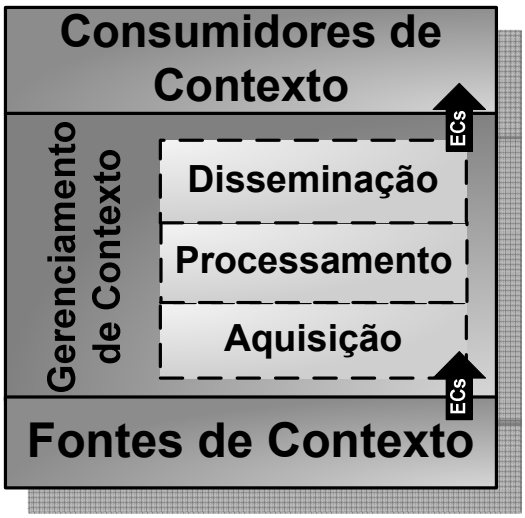


Figura 2.10 – Modelo conceitual de arquitetura em camadas para uma ASC (adaptado de SANTOS, 2008).

o *Módulo de Aquisição* gerencia as fontes de contexto e recupera os ECs por meio do uso de componentes de software apropriados que permitem acesso às funcionalidades internas de cada fonte de contexto. O *Módulo de Processamento* é responsável por raciocinar e interpretar os ECs, transformando-os em um formato ou nível de abstração adequados para que sejam utilizados pelos consumidores de contexto. Além disso, esse módulo também pode realizar a combinação ou agregação de diferentes ECs provenientes de múltiplas fontes de contexto (BALDAUF et al., 2007). O *Módulo de Disseminação*, por sua vez, organiza os ECs e os disponibiliza aos consumidores de contexto. As *Fontes de Contexto* são elementos de software (e.g. perfis armazenados, bases de dados externas, drivers de câmeras e sensores) que fornecem informações atualizadas sobre as entidades consideradas no domínio da ASC. Por fim, os *Consumidores de Contexto* são elementos de software que usam os ECs relevantes para desempenhar algum comportamento sensível ao contexto, como por exemplo, adaptação de conteúdo. Tanto as fontes quanto os consumidores de contexto estão ligados à camada de gerenciamento de contexto através de interfaces bem definidas (SANTOS, 2008).

A maioria das arquiteturas em camadas de ASC existentes diferem entre si em aspectos relacionados às funcionalidades fornecidas, à localização e nomeação das camadas, dentre outras questões arquiteturais. Apesar disso, é possível identificar aspectos comuns na arquitetura das ASCs modernas quando se analisa as diferentes abordagens de projeto (BALDAUF et al., 2007). Conforme ilustrado na Figura 2.10, a camada de *Gerenciamento de Contexto* inclui módulos com funções para adquirir os ECs de suas fontes de contexto, bem como para processá-los e disseminá-los aos

consumidores de contexto. Essas mesmas funcionalidades podem aparecer em diferentes ASCs baseadas camadas em módulos com nomes distintos, subdivididos ou aglutinados a outros módulos. Uma descrição mais detalhada a respeito de outras arquiteturas de ASCs e *frameworks* podem ser encontrados em Baldauf et al. (2007).

Independentemente da organização dos módulos de uma ASC baseada em camadas, nota-se que a camada de *Gerenciamento de Contexto* é uma entidade que envolve a definição de soluções que permitem separar as operações relacionadas à manipulação de contexto das funcionalidades de negócio do domínio de aplicação (SANTOS, 2008).

2.5 Estratégias de Adaptação de Interfaces do Usuário

A crescente disponibilidade de dispositivos móveis tem levado à necessidade de ferramentas e técnicas para adaptar o grande número de aplicações Web existentes originalmente desenvolvidas para sistemas *desktop* em versões que são acessíveis e utilizáveis por dispositivos móveis, permitindo aos usuários interagir adequadamente com essas aplicações, independentemente do dispositivo de acesso (PATERNÒ et al., 2008). Como a Computação Ubíqua hoje em dia se torna mais complexa e oferece uma variedade de dispositivos de acesso para múltiplos contextos de uso, uma série de desafios para o projeto e o desenvolvimento de interfaces do usuário surgem para os engenheiros de software (EISENSTEIN et al., 2000).

Aplicações interativas e suas interfaces, tais como as desenvolvidas para a Web 2.0, geralmente devem ser executadas em diferentes plataformas computacionais, desde uma estação de trabalho poderosa a um pequeno telefone celular. Cada dispositivo de acesso apresenta um perfil específico, o qual compreende características como poder de processamento, memória, capacidades gráficas, dimensões de tela, formas de interação suportadas (e.g. tátil, vocal, gestual), recursos especiais fornecidos (e.g. acelerômetro, reconhecimento facial, bússola, GPS), dentre outras. Alguns dispositivos, por exemplo, apresentam capacidades gráficas sofisticadas (e.g. *desktops*, *notebooks*, *tablets*), enquanto outros possuem limitações na resolução de tela (e.g. *smartphones*, telefones celulares); alguns são equipados com vários tipos de periféricos de entrada/saída

(e.g. teclado, mouse, *trackball*), ao passo que outros são restritos a um pequeno teclado ou a uma tela sensível ao toque. Eles também possuem diferentes formas de conectividade, variando desde redes sem fio de baixa velocidade até as mais velozes redes cabeadas. As limitações de cada dispositivo impõem restrições na forma com que os usuários interagem com as aplicações e, consequentemente, na interface do usuário que deverá ser implementada (EISENSTEIN et al., 2000).

Neste sentido, é importante considerar as principais características dos dispositivos no desenvolvimento das interfaces gráficas das aplicações (PATERNÒ et al., 2008). Para responder aos desafios que surgem a partir desse escopo, a adaptação de conteúdo toma lugar (BUCHHOLZ & SCHILL, 2003). Em geral, existem duas estratégias de geração de código que podem ser empregadas para adaptar o conteúdo de interfaces do usuário desenvolvidas para a Web (VIANA & ANDRADE, 2008):

- **Geração de código em tempo de desenvolvimento (adaptação estática):** diferentes versões da interface, cada qual com conteúdos apropriados para um determinado dispositivo ou um conjunto de dispositivos, são construídas durante o processo de desenvolvimento (i.e., antes da execução da aplicação). Mais tarde, de acordo com o contexto da interação, a versão mais apropriada é escolhida para o dispositivo de acesso. Assim, é possível conduzir implementações que aproveitem ao máximo as funcionalidades específicas de determinado conjunto de dispositivos. No entanto, essa abordagem levanta diversos problemas: altos custos e maior tempo de desenvolvimento devido à necessidade de se construir uma versão especializada da interface para cada tipo de dispositivo; custos de manutenção elevados, dado que uma simples mudança nos requisitos da interface requer alterações isoladas em todas as versões construídas; processo de desenvolvimento interminável, pois à medida que novos dispositivos surgem outras versões devem ser construídas; dentre outros (EISENSTEIN et al., 2000).
- **Geração de código em tempo de execução (adaptação dinâmica):** o código da interface é gerado durante a execução da aplicação a partir de descrições abstratas constantes em um modelo da interface. Essa abordagem é usada, sobretudo, para adaptação de sistemas Web baseados em requisição/resposta, nos quais as interfaces são mapeadas

para tecnologias como *Wireless Markupt Language (WML)*, XHTML e *VoiceXML*, dependendo do dispositivo que faz a requisição. Assim, o processo de desenvolvimento é simplificado uma vez que não há necessidade de implementar e gerenciar inúmeras versões da interface. No entanto, visto que a definição da interface deve ser genérica o suficiente para contemplar todos os possíveis dispositivos, essa estratégia restringe o acesso a funcionalidades específicas dos dispositivos (VIANA & ANDRADE, 2008). Além disso, uma vez que todo o código da interface deve ser gerado no momento da execução para contemplar as especificidades do contexto da interação atual, o tempo de processamento do código pode impactar negativamente no desempenho da execução da aplicação.

De forma geral, a estratégia mais adotada é a adaptação estática, com interfaces sendo desenvolvidas para cada contexto de uso. Contudo, uma vez que, para manter a consistência, as interfaces devem ser implementadas para várias dispositivos, o processo repetitivo de desenvolvimento multi-plataforma torna-se dispendioso, complexo e sujeito a erros, já que os dispositivos apresentam um conjunto de restrições próprio a serem satisfeitas. Além disso, é quase inevitável que esse processo de desenvolvimento multi-plataforma será realizado por vários indivíduos, os quais podem ter diferentes habilidades e experiências, o que reforça a dificuldade de manter a compatibilidade e a coerência entre as diferentes versões desenvolvidas para a interface (EISENSTEIN et al., 2000). Por outro lado, a estratégia de adaptação dinâmica pode produzir resultados inutilizáveis com elementos ilegíveis e componentes inadequados para certos dispositivos (PATERNÒ et al., 2008).

2.6 Considerações Finais

Este capítulo apresentou uma revisão da literatura a respeito dos principais conceitos e técnicas envolvidos no trabalho elaborado. Foram discutidos os conceitos relacionados às interfaces ricas e Web 2.0, bem como aqueles referentes ao desenvolvimento de interfaces dirigido a modelos, aplicações sensíveis ao contexto e estratégia de adaptação de interfaces do usuário, aplicados neste trabalho.

Capítulo 3

PROCESSO DIRIGIDO A MODELOS PARA A CONSTRUÇÃO DE INTERFACES RICAS DE APLICAÇÕES UBÍQUAS SENSÍVEIS AO CONTEXTO

3.1 Considerações Iniciais

Processo de software é definido como um roteiro com passos previsíveis e diretrizes relacionadas ao desenvolvimento de sistemas computacionais. Compreende a abordagem que é empregada quando o software é elaborado, bem como as tecnologias que o constituem, tais como métodos técnicos e ferramentas automatizadas. Seu objetivo é apoiar a criação de produtos de software com alta qualidade dentro dos prazos e orçamento previstos (PRESSMAN, 2004; PFLEEGER, 2005).

Conforme apontado no capítulo introdutório desta dissertação, a Computação Ubíqua tem demandado uma série de requisitos adicionais ao desenvolvimento de software. Dentre esses requisitos destaca-se a necessidade de adaptar o conteúdo das aplicações aos diferentes dispositivos computacionais do usuário que podem acessá-las em contextos distintos (GAJOS & WELD, 2004). Contudo, diante da diversidade de dispositivos e plataformas existentes, construir e manter versões específicas da aplicação para cada tipo de dispositivo tornaram-se tarefas de difícil acompanhamento para os engenheiros de software, especialmente no que concerne

à construção das interfaces gráficas pelas quais os usuários irão interagir com a aplicação. Ainda, no caso de aplicações que utilizam de interfaces ricas na Web 2.0, essas tarefas tornam-se mais complexas em vista da necessidade de preservar os aspectos de interação que proporcionam aos usuários uma rica experiência com a aplicação. Neste sentido, é importante fornecer aos desenvolvedores um processo de software adequado que possa guiá-los através do estabelecimento de atividades e artefatos que apoiem no atendimento aos requisitos de adaptação demandados por uma aplicação ubíqua, considerando-se os diferentes contextos em que se executa a aplicação.

Neste capítulo é apresentado um *Processo Dirigido a Modelos para apoio à Construção de Interfaces Ricas de Aplicações Ubíquas Sensíveis ao Contexto* (*Model Driven Process to Construct Rich Interfaces for Context-Sensitive Ubiquitous Applications – Model Driven RichUbi*). Tendo em vista a infinidade de dispositivos e plataformas existentes na Computação Ubíqua, no processo proposto são combinadas as concepções de Desenvolvimento Dirigido a Modelos (MDD) e Modelagem Específica de Domínio (DSM), focando-se na modelagem de interfaces e na transformação de modelos em código-fonte para diferentes tecnologias de implementação. Com isso, parte da complexidade envolvida na codificação das interfaces para os diversos dispositivos fica encapsulada nas transformações, o que contribui para a redução de esforços, diminuição da incidência de erros, aumento da produtividade e melhoria da qualidade das interfaces produzidas. Além disso, com base nas concepções de *Sensibilidade ao Contexto* e *Adaptação de Interfaces*, no processo também se estabelece a construção e reutilização de componentes de software responsáveis pela manipulação do contexto para detecção do perfil do dispositivo de acesso e adaptação das interfaces ricas em conformidade com as características do dispositivo identificado. O uso do contexto para a adaptação das interfaces em tempo de execução permite que o número de versões construídas seja reduzido e, consequentemente, o processo de desenvolvimento seja simplificado.

Este capítulo está organizado da seguinte forma: a Seção 0 fornece uma visão geral do *Model Driven RichUbi*, apresentando seus principais elementos e etapas, cujas atividades são detalhadas nas seções subsequentes (Seções 3.2.1 e 3.2.2); e a Seção 3.3 encerra o capítulo apresentando algumas considerações finais.

3.2 Model Driven RichUbi

Considerando as motivações e desafios existentes no desenvolvimento de aplicações de Web 2.0 adaptativas para a Computação Ubíqua, este trabalho apresenta um processo de software dirigido a modelos, denominado *Model Driven RichUbi*, com o intuito de simplificar a construção de interfaces ricas de aplicações ubíquas que se adaptam quando visualizadas em diferentes dispositivos.

O domínio de aplicações ubíquas como um todo pode ser muito abrangente para ser abordado de forma integral por este trabalho. Além do requisito de adaptabilidade às condições dinâmicas do ambiente em que operam, aplicações ubíquas também requerem uma série de outros requisitos especiais se comparadas com aplicações tradicionais, tais como: onipresença de serviços, pervasibilidade, interoperabilidade, privacidade, invisibilidade, disponibilidade, autonomia, confiabilidade, portabilidade, descoberta e composição de serviços, dentre outros (SPÍNOLA et al., 2007; GARLAN & SCHMERL, 2001). A adaptabilidade também é um termo bastante abrangente que engloba a adaptação tanto do comportamento quanto do conteúdo da aplicação considerando-se diversas características contextuais, como, por exemplo, o perfil do dispositivo, as preferências e condições do usuário, as características da rede de acesso e outros fatores relevantes. Além disso, essas aplicações podem estar disponibilizadas em diferentes plataformas, seja na Web, embarcadas em um microcontrolador, ou instaladas em algum dispositivo específico. Dessa forma, a contribuição deste trabalho se focou em um aspecto pontual da construção de uma aplicação ubíqua. No processo proposto restringiu-se o domínio de trabalho a aplicações ubíquas desenvolvidas para a Web 2.0 que adaptam seu conteúdo, i.e., suas interfaces ricas, com base no perfil do dispositivo de acesso recuperado do contexto da interação. Ainda assim, não se teve a pretensão de abordar todos os aspectos do desenvolvimento dessas aplicações (e.g. modelagem de dados, implementação da lógica de negócios, manutenção). Porém, mostrou-se uma maneira de simplificar a construção das interfaces ricas adaptativas de forma sistematizada através de um conjunto de atividades e artefatos, os quais podem ser ampliados futuramente para possibilitar um maior alcance do processo.

Conforme apresentado no diagrama *Structured Analysis and Design Technique (SADT)* (ROSS, 1977) da Figura 3.1, o processo *Model Driven RichUbi* é realizado em duas etapas principais: *Engenharia de Domínio (ED)* e *Engenharia de Aplicação (EA)*. Nesse diagrama, os retângulos simbolizam as etapas ou atividades do processo. As setas que incidem pelo lado esquerdo dos retângulos representam as entradas de dados/artefatos, e as setas que saem pelo lado direito representam as saídas geradas em cada atividade. As setas do lado superior representam os controles que orientam a execução de cada etapa/atividade. Já as setas do lado inferior representam os participantes, ferramentas e mecanismos que executam ou automatizam a execução das atividades.

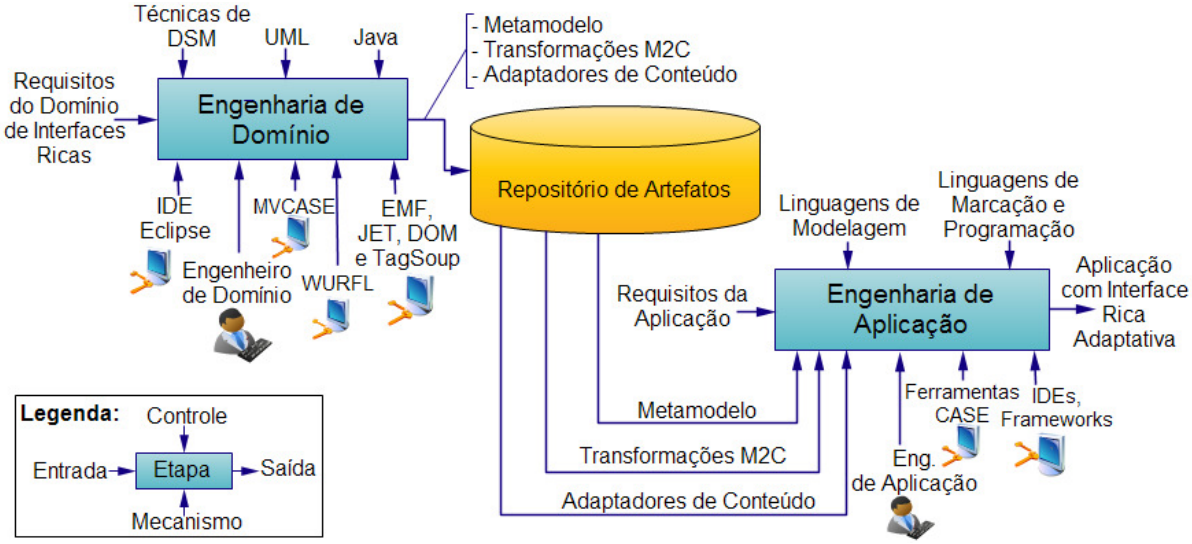


Figura 3.1 – Visão geral em alto nível do processo *Model Driven RichUbi*.

A ED engloba as atividades para a construção dos artefatos reutilizáveis que apoiam o desenvolvimento das aplicações ubíquas com interfaces ricas adaptativas. Esses artefatos compreendem um metamodelo do Domínio de Interfaces Ricas que expressa a sintaxe abstrata de uma Linguagem Específica de Domínio (DSL) para apoio à modelagem, transformações M2C para geração de código e adaptadores de conteúdo das interfaces. Conforme as concepções da DSM, o metamodelo permite reutilizar o conhecimento do Domínio de Interfaces Ricas em diferentes projetos, bem como fornece uma infraestrutura para automatizar grande parte da geração de código das interfaces. As transformações M2C e os adaptadores de conteúdo, por sua vez, atuam como mecanismo de apoio no desenvolvimento das interfaces. Os artefatos provenientes da etapa de ED são armazenados no repositório de artefatos para que se sejam posteriormente reusados.

Diferentemente da ED, cujo enfoque é o desenvolvimento **para reúso**, a EA visa ao desenvolvimento **com reúso** dos artefatos produzidos na ED. Nessa etapa, as aplicações ubíquas com interfaces ricas adaptativas são produzidas instanciando-se o metamodelo para a modelagem das interfaces. As transformações M2C são reutilizadas para a geração parcial do código das interfaces a partir dos modelos para diferentes dispositivos, e os adaptadores de conteúdo são empregados para conferir às interfaces desenvolvidas uma característica adaptativa.

Dentre os mecanismos estabelecidos para apoio à execução do processo, tem-se o *Integrated Development Environment (IDE)* Eclipse[19] e a *Multiple-View CASE (MVCASE)* (LUCRÉDIO et al., 2003). A MVCASE é uma ferramenta *Computer-Aided Software Engineering (CASE)* que dá suporte à modelagem e geração de código, sendo disponibilizada como um *plugin* no IDE Eclipse. Para a especificação do metamodelo foi definida a utilização da linguagem de metamodelagem *Ecore* do *Eclipse Modeling Framework (EMF)* (STEINBERG et al., 2008). Além disso, estabeleceu-se o uso do *framework Java Emitter Templates (JET)* (ECLIPSE DOCUMENTATION, 2010) para a criação das transformações M2C. As *Application Programming Interfaces (APIs)* Java *Document Object Model (DOM)* (JAVA, 2010) e *TagSoup* (TAGSOUP, 2010) são utilizadas para implementar os adaptadores de conteúdo. Para guiar a adaptação das interfaces, as informações acerca dos perfis dos dispositivos são consumidas a partir da base de dados XML pública chamada *Wireless Universal Resource File (WURFL)*[20]. Os principais controles definidos para orientar a execução da etapa de ED incluem a semântica e sintaxe da UML e da linguagem de programação Java, bem como técnicas de DSM.

No *Model Driven RichUbi*, tanto na ED como na EA, as atividades para construção dos artefatos estendem as disciplinas tradicionais de *Análise*, *Projeto*, *Implementação* e *Testes* da Engenharia de Software, sendo estas adaptadas para atender às necessidades da DSM na construção de interfaces ricas adaptativas das aplicações ubíquas.

Um aspecto importante do processo *Model Driven RichUbi* é a abordagem empregada para adaptação das interfaces ricas. Buscando superar as deficiências individuais das estratégias de adaptação de interfaces puramente estática

[19] http://www.eclipse.org/.

[20] http://wurfl.sourceforge.net/.

(construção de várias versões em tempo de desenvolvimento) e puramente dinâmica (adaptação de todo o código em tempo de execução) (VIANA & ANDRADE, 2008), bem como reduzir o número de versões construídas, o processo emprega uma abordagem de **adaptação híbrida**. Essa abordagem combina geração de código a partir da modelagem em tempo de desenvolvimento (facilitada pela geração parcial de código a partir dos modelos), com a geração de código em tempo de execução (facilitada pelo reúso de adaptadores de conteúdo). Dessa forma, durante a EA, apenas algumas poucas versões mais genéricas da interface serão construídas, cada qual adequada a um determinado grupo de dispositivos (adaptação estática). Os adaptadores de conteúdo permitem que, no acesso à aplicação, a versão que melhor se encaixa ao perfil do dispositivo recuperado do contexto seja selecionada e os trechos do código que necessitam ser refinados sejam adequados às peculiaridades do dispositivo (adaptação dinâmica).

A abordagem híbrida empregada no processo contribui para a redução dos esforços de desenvolvimento, já que um número menor de versões da interface podem ser desenvolvidas. Além disso, como apenas partes do código são adaptadas durante a execução, e não todo o código, o impacto no desempenho da execução da aplicação é reduzido. Considerando ainda que os conteúdos das versões genéricas são refinados dinamicamente conforme o perfil do dispositivo identificado do contexto, há um melhor aproveitamento dos recursos disponíveis no dispositivo e, consequentemente, uma melhor adaptação das interfaces ricas.

3.2.1 Engenharia de Domínio (ED)

A ideia de desmembrar o desenvolvimento de software em etapas distintas, uma voltada para construção de artefatos que apoiam o desenvolvimento de aplicações de um domínio particular, e outra focada no reúso sistematizado desses artefatos para construção das aplicações, não é nova. Essa distinção entre desenvolvimento para/com reutilização provém das atuais abordagens de Engenharia de Software conhecidas como Linhas de Produto de Software *(Software Product Lines – LPS)* (LINDEN et al., 2007). Nessas abordagens as aplicações de um domínio são desenvolvidas considerando-se uma *família de programas*, onde primeiro estuda-se e implementa-se o que os programas de uma mesma família têm em comum para depois estender o que os diferencia. Assim, ao invés de desenvolver software

pensando apenas nos requisitos de uma única aplicação, em LPS tenta-se desenvolver software que atenda aos requisitos de um conjunto de aplicações similares, de maneira que seja possível reaproveitar o que é comum e desenvolver apenas o que é inteiramente novo. Na terminologia da área de LPS, a construção das partes comuns (desenvolvimento *para* reúso) é chamada de *Engenharia de Domínio*, e a construção de uma aplicação da família (desenvolvimento *com* reúso) é chamada de *Engenharia de Aplicação* (CLEMENTS & NORTHROP, 2002; LINDEN et al., 2007).

Apesar de não focar-se nas metodologias envolvidas em LPS, neste trabalho a Engenharia de Domínio possui o mesmo sentido, ou seja, concentra-se no desenvolvimento de artefatos de software para posterior reutilização. De forma mais geral, a ED constitui-se como um processo de identificação e organização do conhecimento acerca de uma classe de problemas – o domínio do problema – para apoiar sua descrição e solução. Seu objetivo é sistematizar a criação de modelos do domínio, arquiteturas e conjuntos de artefatos de software para dar suporte à construção de aplicações num domínio de problema particular (BLOIS et al., 2005).

Usando a notação de diagramas SADT, a Figura 3.2 apresenta as atividades definidas para a ED no *Model Driven RichUbi*, que compreendem: *1) Especificar*, *2) Projetar* e *3) Implementar Metamodelo dos Componentes de Interface Rica*; *4) Construir Transformações Modelo para Código*; e *5) Construir Adaptadores de Conteúdo dos Componentes de Interface Rica*. Conforme se observa na Figura 3.2, a ED inicia-se com os requisitos do Domínio de Interfaces Ricas para construir um metamodelo de componentes de interfaces ricas para apoio à modelagem. Também na ED, com base no metamodelo implementado, são construídas as transformações M2C e os adaptadores de conteúdo dos componentes de interface, que servem como mecanismos de apoio na execução da EA.

A seguir, são detalhadas as tarefas realizadas em cada atividade da ED.

3.2.1.1 Especificar Metamodelo dos Componentes de Interface Rica

O objetivo desta atividade é identificar, a partir dos requisitos do Domínio de Interfaces Ricas, os componentes de interface que são úteis para a construção das aplicações ubíquas de Web 2.0. Esses componentes são elicitados, especificados, analisados e representados num metamodelo de componentes de interfaces ricas. Uma das formas de realizar essa identificação é através do estudo dos componentes de interface disponíveis em diversos ambientes de desenvolvimento Web, como, por

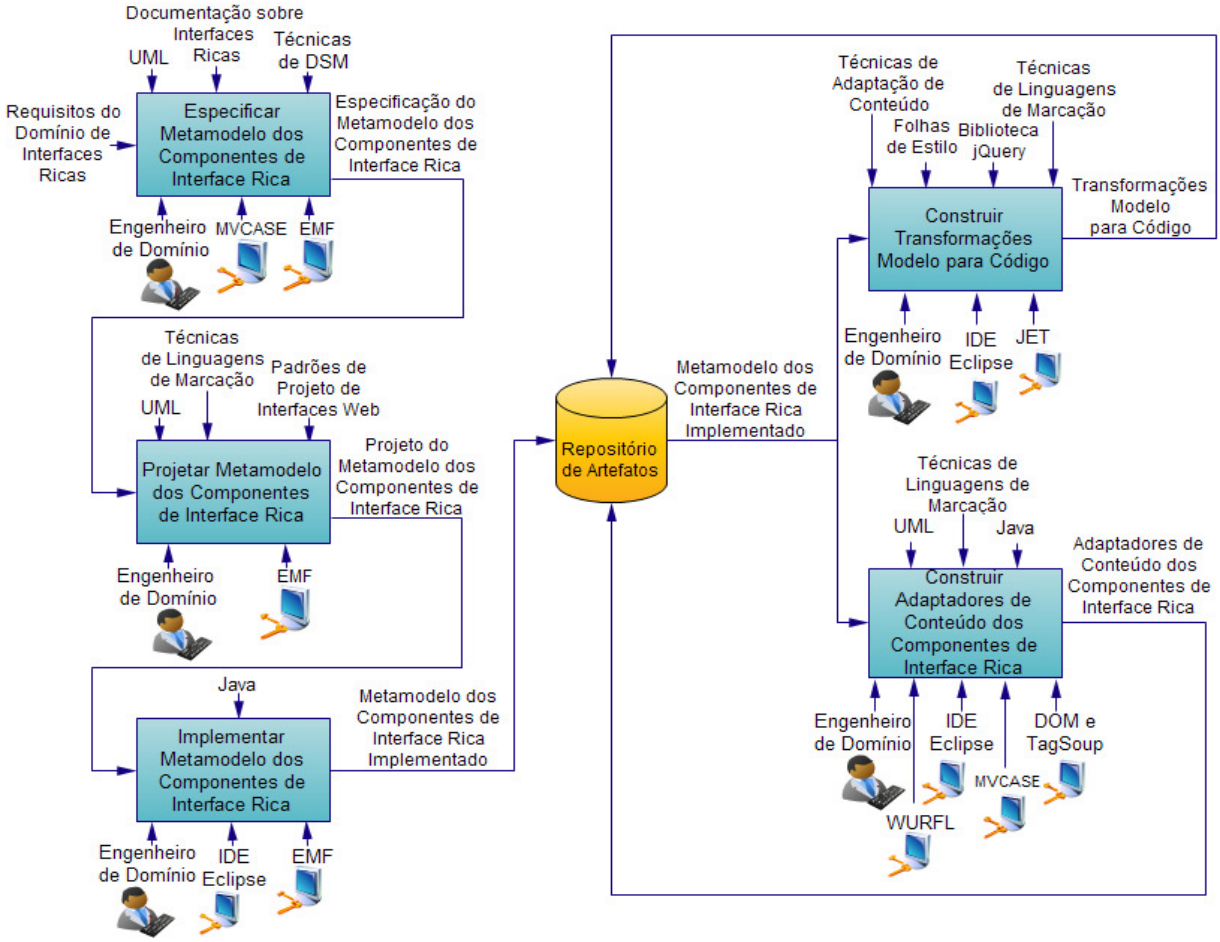


Figura 3.2 – Engenharia de Domínio no *Model Driven RichUbi*.

exemplo, o *Adobe Dreamweaver*[21] e o *MS Visual Studio*[22], em conjunto com a análise de documentações a respeito de interfaces ricas[23, 24, 25]. Por meio desses estudos, o Engenheiro de Domínio pode identificar os componentes comumente utilizados na construção de interfaces ricas na Web e modelar suas similaridades estruturais e comportamentais. No processo, a UML é utilizada para apoiar na modelagem e especificação desses componentes.

A Figura 3.3 mostra, por exemplo, um diagrama de classes que especifica alguns componentes de controle de formulários, como *Botão* (`Button`), *Campo de Texto* (`TextField`) e *Caixa de Seleção* (`Select`), bem como componentes avançados de interfaces ricas, como *Painel de Abas* (`TabbedPanel`), *Painel Deslizante*

[21] http://www.adobe.com/products/dreamweaver/.

[22] http://www.microsoft.com/visualstudio/en-us/.

[23] http://www.jboss.org/richfaces/docs.html.

[24] http://jqueryui.com/demos/.

[25] http://www.asp.net/ajax/AjaxControlToolkit/Samples/.

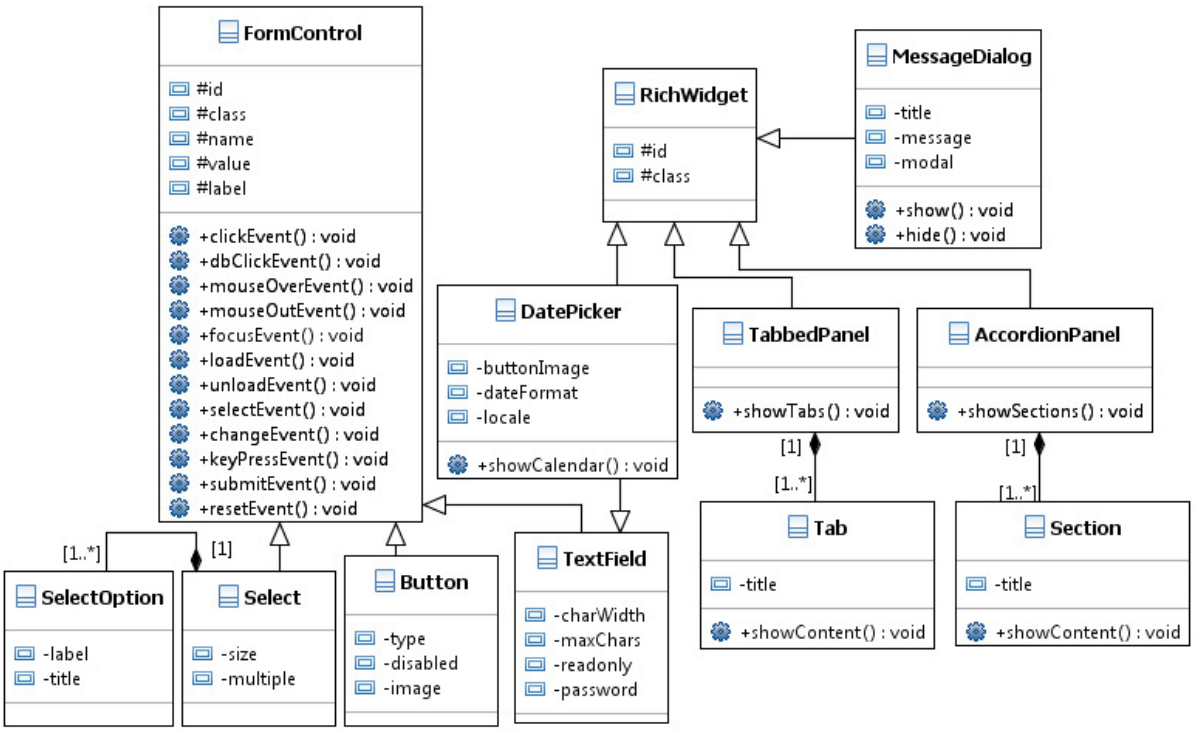


Figura 3.3 – Especificação dos componentes de interface rica.

(`AccordionPanel`), *Caixa de Mensagem* (`MessageDialog`) e *Seletor de Data* (`DatePicker`) identificados nesta atividade. Durante as tarefas de modelagem desta atividade, o Engenheiro de Domínio é auxiliado pela ferramenta MVCASE.

A partir das representações dos componentes de interface especificados nos diagramas de classe, o Engenheiro de Domínio parte para a especificação do metamodelo dos componentes de interface rica, definindo suas metaclasses, meta-atributos, e meta-relacionamentos. Conforme as concepções e técnicas da DSM, esse metamodelo deve refletir o conhecimento acerca do Domínio de Interfaces Ricas, de forma a facilitar a construção das aplicações ubíquas com interfaces ricas. No *Model Driven RichUbi*, o metamodelo é construído utilizando-se o metametamodelo *Ecore* do *framework* EMF, que fornece uma metalinguagem para definição de metamodelos no IDE Eclipse.

Por exemplo, as classes `FormControl`, `Button`, `TextField`, `RichWidget`, `TabbedPanel`, `Tab` , `DatePicker`, `MessageDialog`, `AccordionPanel` e `Section` do diagrama da Figura 3.3 foram mapeadas para as metaclasses de mesmos nomes destacadas no trecho do metamodelo na Figura 3.4. Os atributos `id` e `class` da classe `FormControl` foram mapeados para meta-atributos homônimos e separados, respectivamente, nas metaclasses `IdentifiableComponent` e

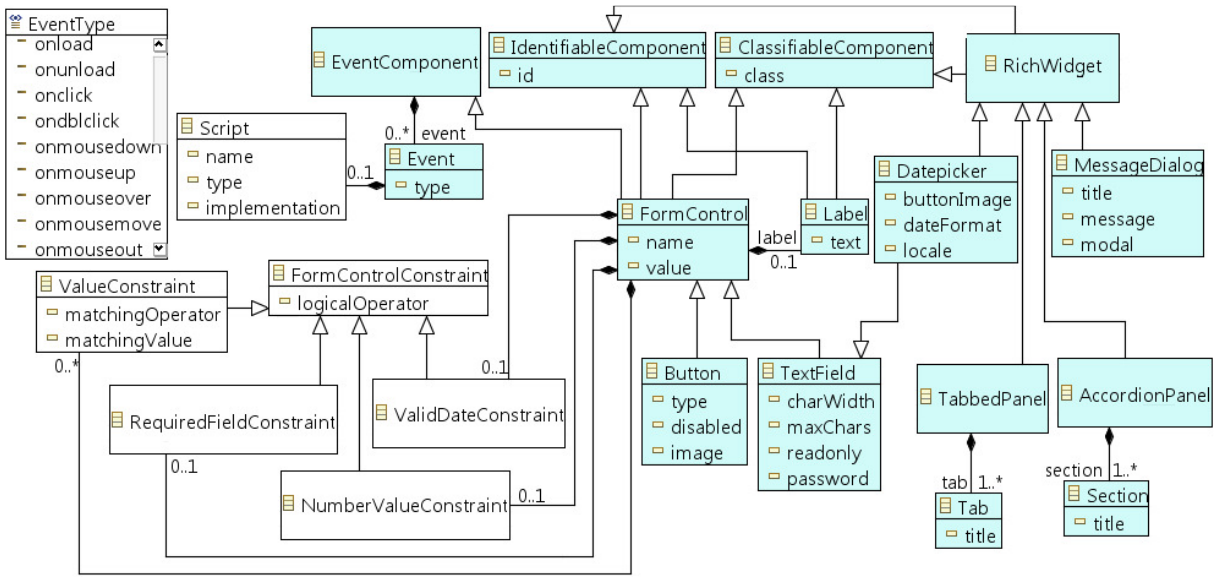


Figura 3.4 – Metamodelo dos componentes de interface rica.

`ClassifiableComponent`. O atributo `label` da classe `FormControl` deu origem à metaclasse `Label`. Além disso, os métodos de tratamento de eventos da classe `FormControl` originaram a meta-enumeração `EventType` e as metaclasses `Event`, `EventComponent` e `Script`. O metamodelo inclui também metaclasses para tratar os requisitos de restrições de entrada de dados das interfaces ricas, tais como: `FormControlConstraint`, que atua como restrição base para os demais tipos de restrições de controles de formulários; `ValueConstraint`, que representa as restrições de valores de entrada; `NumberValueConstraint`, que representa as restrições para campos de entrada numéricos; `RequiredFieldConstraint`, que representa as restrições para campos de preenchimento obrigatório; e `ValidDateConstraint`, que representa as restrições para campos de preenchimento de datas. Apenas os principais meta-atributos e meta-elementos foram apresentados no metamodelo da Figura 3.4 para facilitar sua compreensão. Métodos acessores, construtores e outros, apesar de não apresentados, fazem parte do metamodelo.

3.2.1.2 Projetar Metamodelo dos Componentes de Interface Rica

O objetivo desta atividade é definir os padrões, tecnologias e plataformas de hardware e software que viabilizam a construção do metamodelo. Através dessas decisões de projeto o Engenheiro de Domínio refina a especificação do metamodelo de interfaces ricas produzida na atividade precedente.

Por exemplo, a Figura 3.5 mostra parte do metamodelo refinado com o emprego do padrão de projeto de interfaces Web chamado *Portal Site*[26]. Esse padrão, representado à direita da Figura 3.5, define as regiões de cabeçalho (*header*), navegação (*navigation*), conteúdo (*content*), pesquisa (*search*) e rodapé (*footer*) de um portal Web. À esquerda da Figura 3.5 tem-se o metamodelo com a inclusão de novas metaclasses (sombreadas), cujos estereótipos indicam a associação das mesmas com as regiões especificadas no padrão *Portal Site*. Por exemplo, a metaclasse `NavigationRegion` identificada com o estereótipo `<<navigation>>` foi incluída ao metamodelo para representar a região de navegação do portal Web. Essa metaclasse possui um meta-atributo chamado `orientation`, que indica a orientação da região de navegação, a qual, conforme especificado no padrão *Portal Site*, pode ser definida tanto verticalmente quanto horizontalmente.

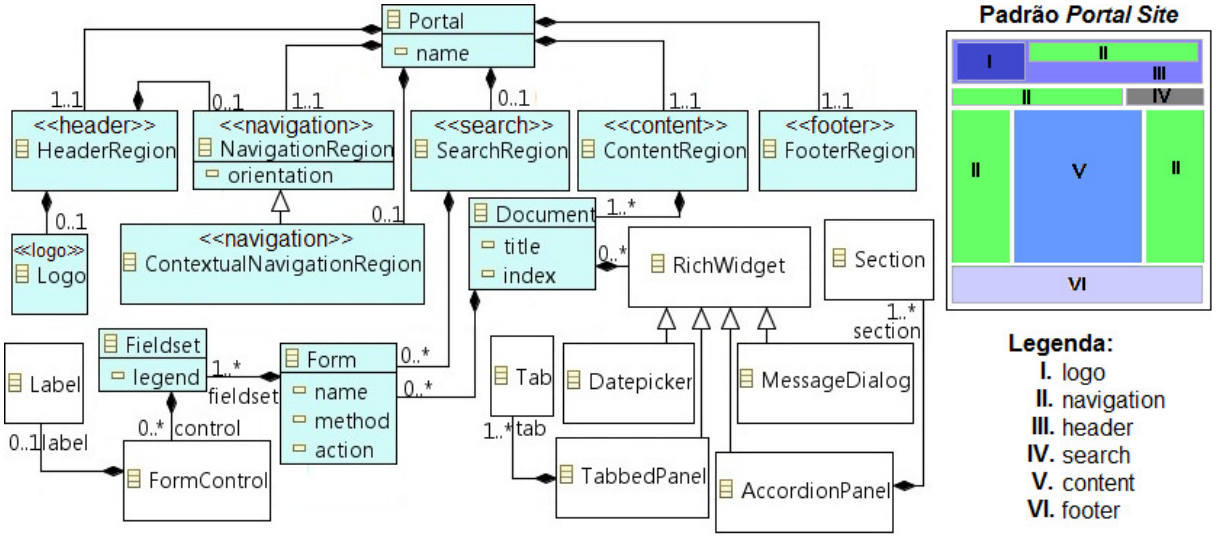


Figura 3.5 – Refinamento do metamodelo.

Além das metaclasses que representam as regiões do portal Web, também foram incluídas ao metamodelo metaclasses para adequá-lo à estrutura dos documentos XHTML (W3C, 2002), como as metaclasses `Document`, `Form` e `Fieldset`. A definição das relações de composição entre as metaclasses `Document` e `Form`, `Form` e `Fieldset`, e `Fieldset` e `FormControl` foi realizada em conformidade com a especificação da XHTML. O refinamento do metamodelo com a inclusão dessas metaclasses e relacionamentos permite definir a forma com que os componentes de interface rica serão dispostos e organizados nos modelos de interface construídos como instância do metamodelo durante a EA.

[26] http://www.welie.com/patterns/showPattern.php?patternID=portals.

3.2.1.3 Implementar Metamodelo dos Componentes de Interface Rica

Nesta atividade o metamodelo é implementado a partir das especificações constantes no projeto do metamodelo resultante da atividade anterior. A implementação do metamodelo visa a obter componentes de software que reflitam as entidades e relacionamentos representados no metamodelo de modo que possam ser instanciados para a criação dos modelos de interface rica na EA.

Para a implementação do metamodelo, utiliza-se o *framework* EMF. O EMF possibilita ao Engenheiro de Domínio gerar o código Java do metamodelo e de um editor de modelos que auxiliará na criação dos modelos das interfaces ricas da aplicação. No editor gerado pelo EMF os modelos são persistidos no formato XMI. Tal formato define uma estrutura de documento XML que considera a relação entre os dados e seus metadados correspondentes, o que facilita o mapeamento dos modelos para código durante a definição de transformações M2C.

Como exemplo, a Figura 3.6 apresenta um trecho do código gerado pelo EMF que corresponde à metaclasse `FormControl`. Conforme observa-se nessa figura, a metaclasse `FormControl` foi traduzida para uma interface Java, a qual possui um relacionamento de herança com as interfaces que representam as metaclasses `IdentifiableComponent`, `ClassifiableComponent` e `EventComponent`, tal como descrito na definição do metamodelo (Figura 3.4). O editor de modelos gerado permite instanciar esses componentes de software para a criação dos modelos que descrevem as interfaces ricas de acordo com a semântica expressa no metamodelo implementado.

```
package br.ufscar.dc.richinterface.metamodel.ric;
import org.eclipse.emf.common.util.EList;

/**
  * @generated
 */
public interface FormControl extends
            IdentifiableComponent,
            ClassifiableComponent,
            EventComponent {
    String getName();
    void setName(String value);
    String getValue();
    void setValue(String value);
    Label getLabel();
    void setLabel(Label value);
    EList<PhraseElement> getAccompanyingPhrase();
    EList<ValueConstraint> getValueConstraints();
    NumberValueConstraint getNumberConstraint();
    void setNumberConstraint(NumberValueConstraint value);
    RequiredFieldConstraint getRequiredFieldConstraint();
    void setRequiredFieldConstraint(RequiredFieldConstraint value);
    ValidDateConstraint getValidDateConstraint();
    void setValidDateConstraint(ValidDateConstraint value);
} // FormControl
```

Figura 3.6 – Implementação do metamodelo.

Durante a execução desta atividade neste trabalho, a implementação do metamodelo juntamente com o editor de modelos foram incorporados como um *plugin* integrado à ferramenta MVCASE, de forma que fosse possível, durante a Engenharia de Aplicação, instanciar o metamodelo para apoio à modelagem das interfaces ricas das aplicações ubíquas.

3.2.1.4 Construir Transformações Modelo para Código

O objetivo desta atividade é construir as transformações que serão aplicadas aos modelos de interface para geração automatizada de código na Engenharia de Aplicação.

Dentre as técnicas existentes para a construção de transformações, destaca-se o uso de *templates* (LUCRÉDIO, 2009). Um *template* é um arquivo de texto qualquer instrumentado com construções de seleção e expansão de código (CZARNECKI & EISENECKER, 2000). Essas construções realizam consultas em uma entrada, que pode ser um arquivo XMI representando um modelo, e usam as informações resultantes dessa consulta como parâmetro para produzir código personalizado, em qualquer linguagem textual (LUCRÉDIO, 2009). Dessa forma, um *template* geralmente é constituído por partes fixas, as quais sempre são incluídas ao código de saída, e partes variáveis, que dependem das informações contidas no modelo de entrada para serem geradas.

A correta implementação das transformações depende do conhecimento da sintaxe da linguagem em que os modelos de entrada são criados, i.e., seu metamodelo. Dessa forma, o metamodelo dos componentes de interface rica é usado como entrada nesta atividade de modo que as transformações produzidas sejam compatíveis com o metamodelo construído. No processo proposto, as transformações são implementadas usando o *framework* JET, o qual disponibiliza uma biblioteca de marcações (*tags*) para metaprogramação, que implementam comandos condicionais, de laços, formatação, dentre outras funções úteis. Assim, através do JET, o Engenheiro de Domínio cria os *templates* que interpretarão os modelos de interface instanciados a partir do metamodelo para gerar o código das interfaces ricas.

Por exemplo, durante a execução da ED neste trabalho, dois tipos de transformações foram construídas de maneira a atender cada componente de interface rica representado no metamodelo elaborado: uma que gera código XHTML para *desktops*, e outra que gera código *XHTML* para *smartphones* aplicando técnicas

de adaptação de conteúdo para dispositivos móveis, tais como *apresentação de conteúdo em coluna única* e *divisão de formulários extensos* (PATERNÒ et al., 2008; VIANA & ANDRADE, 2008). No código de saída dessas transformações foram incorporadas referências a folhas de estilo pré-fabricadas para uma formatação inicial do leiaute das interfaces geradas. Além disso, para a criação de componentes de interface rica mais estilizados adotou-se a biblioteca *JavaScript jQuery*[27], a qual disponibiliza funções prontas para renderização de componentes avançados, tais como painéis deslizantes, painéis de abas, seletores de data e outros.

A Figura 3.7 apresenta um trecho do *template* JET para a geração do código do componente *Painel de Abas* (`TabbedPanel`). Nessa figura também é ilustrado o processo de execução desse *template*, onde cada trecho do mesmo é processado para consultar o modelo de entrada no formato XMI para produzir o código

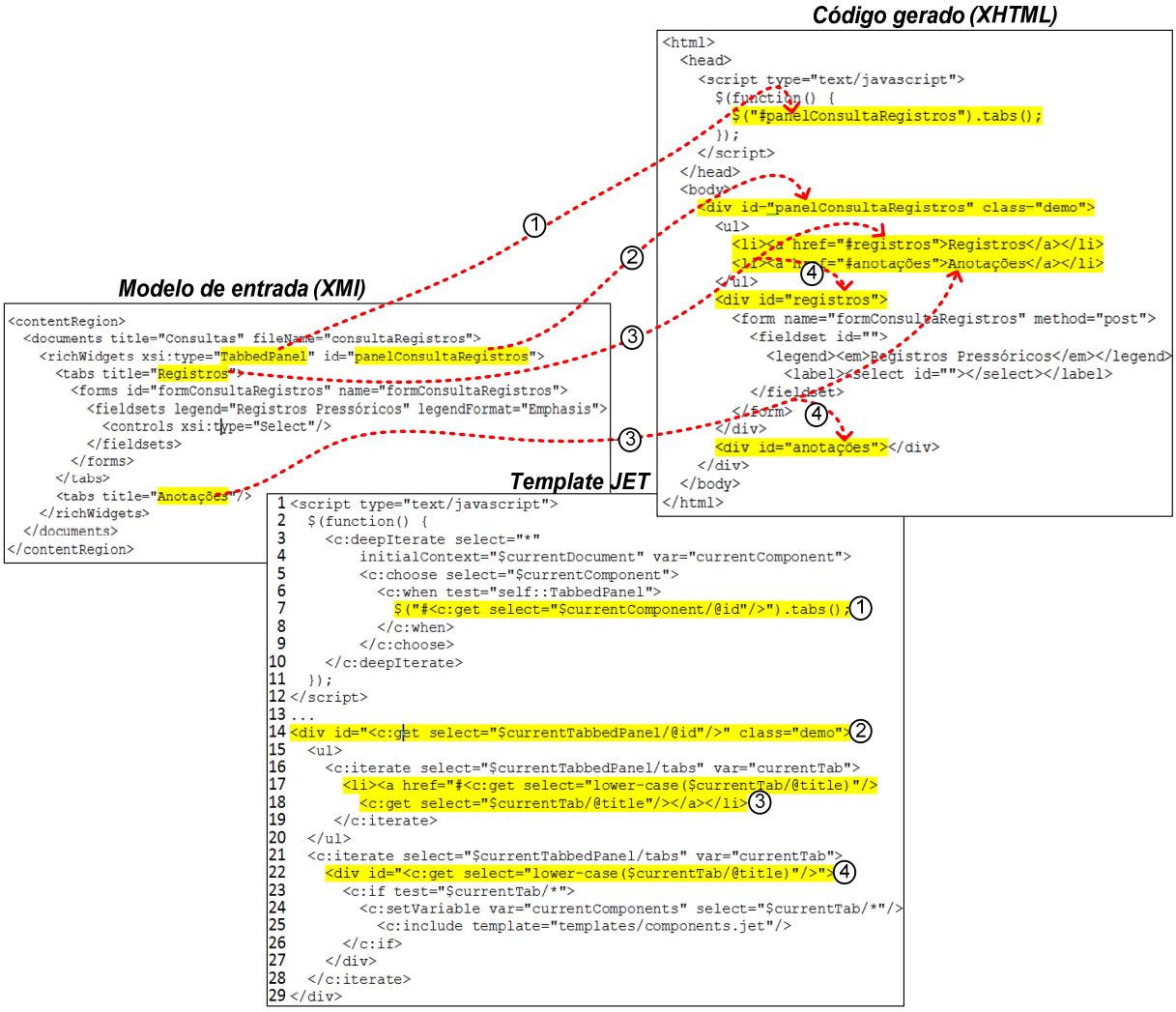


Figura 3.7 – Geração do código do componente *TabbedPanel* baseada em *templates*.

[27] http://jquery.com/.

correspondente. Esse *template* é invocado pelo mecanismo que executa transformações (ou processador de *templates*) quando, durante a leitura do XMI do modelo da interface, o nó que representa o `TabbedPanel` é encontrado. No *template*, as linhas 1-12 geram na saída o código *JavaScript* com a invocação das funções da biblioteca *jQuery* para renderizar o painel de abas na tela do navegador Web. Na linha 14 observa-se a saída de uma marcação `<div>`, que compõem a estrutura do painel de abas, com a referência à classe de estilo denominada `demo`, a qual é definida na folha de estilo pré-fabricada que é copiada ao projeto da aplicação durante a geração de código. Nas linhas 16-19 e 21-28 são feitas as iterações sobre as abas (`tabs`) definidas no modelo para gerar seus conteúdos no código de saída.

Uma vez que os *templates* propiciam a geração de código mais legível, pelo fato de serem parecidos com a saída desejada acrescida de anotações e instruções que consultam a entrada (CLEAVELAND, 2001), sua implementação torna-se mais simples. Porém, é necessário que o Engenheiro de Domínio possua habilidades e conhecimentos associados a tecnologias para o desenvolvimento de interfaces ricas, de modo que possa transcrever nos *templates* o código de saída mais adequado que será reutilizado nas aplicações. A produção dos *templates* com práticas não recomendadas de programação, uso de código não padronizado, implementação inadequada e outras formas de má programação fará com que código sem qualidade se propague nas interfaces produzidas a partir desses *templates*, o que pode dificultar a customização e manutenção das interfaces geradas.

Para apoiar a geração de código na etapa de EA, as transformações foram também incorporadas à ferramenta MVCASE. Devido ao fato dos modelos serem construídos independentemente de plataforma, é possível criar diversos tipos de transformações que geram código a partir de um mesmo modelo para diferentes tecnologias de implementação, tais como *Wireless Markup Language (WML)*, *Voice XML (VXML)*, *Compact Hypertext Markup Language (CHTML)*, e assim por diante. Utilizando esta técnica, as tarefas repetitivas relacionadas à implementação das interfaces para os diferentes dispositivos da Computação Ubíqua são automatizadas, o que economiza tempo e esforços de desenvolvimento. Assim, as transformações em conjunto com os modelos colaboram para simplificar a parte estática da adaptação das interfaces na abordagem híbrida empregada no processo.

3.2.1.5 Construir Adaptadores de Conteúdo dos Componentes de Interface Rica

O objetivo desta atividade é construir os adaptadores de conteúdo que realizarão a parte dinâmica da adaptação das interfaces considerando o perfil do dispositivo recuperado do contexto da interação. Esta atividade recebe como entrada o metamodelo que contém as definições dos componentes de interface a serem utilizados nas aplicações, os quais necessitam ser adaptados quando renderizados em dispositivos com diferentes características.

Antes de partir para a implementação dos adaptadores de conteúdo, o Engenheiro de Domínio deve especificar, para cada componente representado no metamodelo, os requisitos de adaptação demandados quando esses componentes forem visualizados em dispositivos com configurações distintas. Essa especificação não é uma tarefa simples e requer do Engenheiro de Domínio uma boa visão tecnológica a respeito das interfaces, dos dispositivos e suas características que influenciam na adaptação. Para apoiar o trabalho do Engenheiro de Domínio, a UML é utilizada durante a especificação das adaptações. A Figura 3.8 apresenta um exemplo de diagrama de casos de uso construído durante esta atividade para especificar algumas das funcionalidades de adaptação dos componentes de interface `Image`, `Div`, e `TextInput` quando visualizados a partir de um *smartphone*.

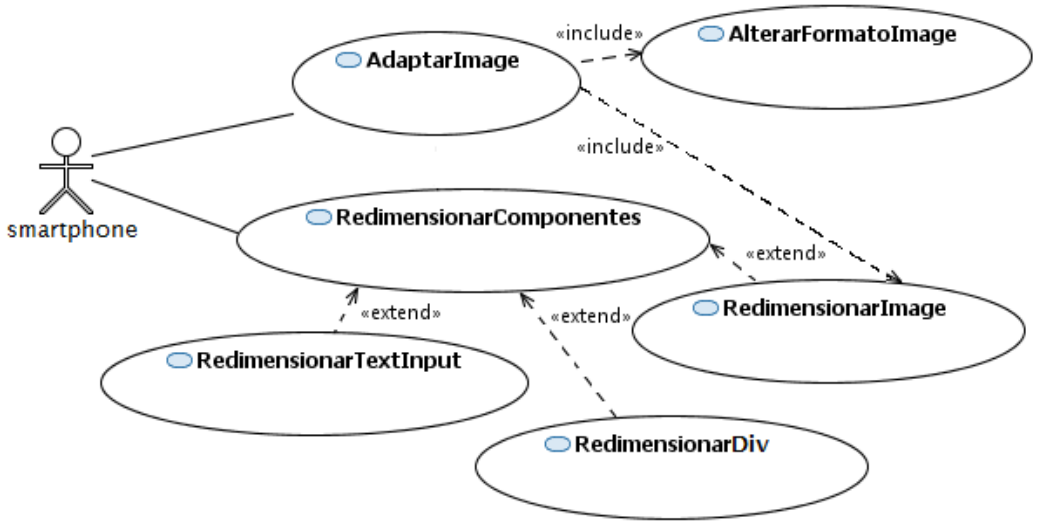


Figura 3.8 – Especificação dos requisitos de adaptação do conteúdo das interfaces.

Em seguida, o Engenheiro de Domínio deve especificar e projetar os componentes de software que efetivamente realizarão as adaptações identificadas. Nesta tarefa, a UML também é empregada para apoiar a modelagem. Por exemplo,

a Figura 3.9 apresenta o diagrama de classes construído para especificar os adaptadores de conteúdo. Cada método da classe `ContentAdapter` especificada nesse diagrama representa um adaptador de conteúdo específico para um determinado componente de interface rica representado no metamodelo. O método `adapt` implementado pela classe `ContentAdapter` realiza a leitura dinâmica do documento Web requisitado e invoca os métodos apropriados para adaptar os diferentes componentes da interface, como controles de formulário, imagens, seções de conteúdo, tabelas, painéis de abas, dentre outros.

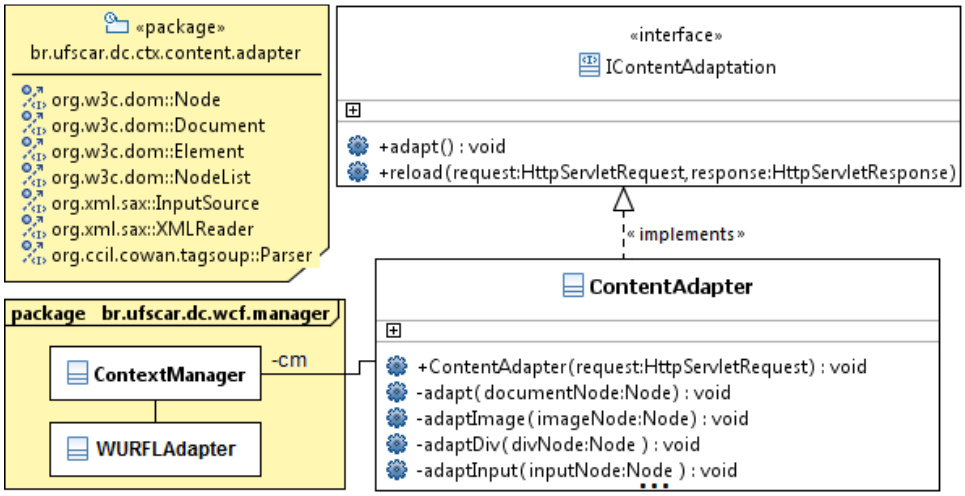


Figura 3.9 – Especificação dos adaptadores de conteúdo.

Para implementar a adaptação dinâmica das interfaces, utiliza-se as APIs Java DOM e *TagSoup*. Essas APIs permitem, em tempo de execução, manipular e transformar documentos XML que suportam a recomendação DOM do *World Wide Web Consortium (W3C)*[28], tais como XHTML e HTML. A API DOM possibilita a leitura e escrita desses documentos na memória, estruturando-os em uma árvore de objetos DOM (e.g. `Node`, `Document`, `Element`), de forma que seja possível analisar e modificar sua estrutura e conteúdos dinamicamente conforme necessário. Uma vez que a API DOM requer documentos bem formatados como entrada, a API *TagSoup* é utilizada para a corrigir possíveis trechos mal escritos dos documentos Web (e.g. adição de *tags* de fechamento ausentes, correção de aninhamento) para que o processamento através da API DOM funcione corretamente.

Para guiar a adaptação das interfaces, a classe `ContentAdapter` consome as informações acerca do perfil do dispositivo de acesso atual e ajusta a interface conforme as peculiaridades do dispositivo identificado a partir do contexto da

[28] http://www.w3.org/.

interação. Conforme apresentado no diagrama de classes da Figura 3.9, essas informações são fornecidas pela classe `ContextManager`, que obtém os perfis dos dispositivos do WURFL utilizando os serviços providos pela classe `WURFLAdapter`. O WURFL armazena os perfis de milhares[29] de dispositivos de diferentes marcas e modelos existentes no mercado e é utilizado por desenvolvedores de software para guiar a criação de soluções apropriadas para dispositivos específicos. Sendo uma base de dados pública, o WURFL recebe atualizações diárias por desenvolvedores espalhados ao redor do mundo interessados em contribuir com a completude e correção das informações nele contidas. Pelo fato de receber contínuas atualizações pela própria comunidade de desenvolvimento, o WURFL foi adotado como fonte de contexto principal para obtenção dos perfis dos dispositivos neste trabalho.

A Figura 3.10 mostra um trecho do código da classe `WURFLAdapter`. A API Java do WURFL é utilizada para a recuperação dos perfis armazenados nessa base de dados. A seleção do perfil do dispositivo é baseada no campo *User-Agent*[30] da requisição HTTP originada pelo dispositivo de acesso. O método `getContextualElement` obtém as informações do perfil armazenadas no WURFL, recebendo como parâmetro um item da enumeração `ContextualElement`, que identifica qual informação sobre o dispositivo deve ser recuperada.

```
package br.ufscar.dc.ctx.sources.adapters;
import net.sourceforge.wurfl.core.DefaultDeviceProvider;

public clas  WURFLAdapter implements ICtxSrcAdapter {
 private String wurflPath = "wurfl.zip";
 private String patchPath = "web_browsers_patch.xml";
 private WURFLManager manager;
 private Device deviceProfile;
 ...
 public WURFLAdapter (HttpServletRequest request) {
   ...
   deviceProfile = manager.getDeviceForRequest(
                        request.getHeader("User-Agent"));
 } //end of constructor
 ...
 @Override
 public String getContextualElement(ContextualElement ce) {
  switch(ce) {
   case DEVICE_DISPLAY_COLUMNS_NUMBER:
       return deviceProfile.getCapability("columns");
   case DEVICE_DISPLAY_RESOLUTION_WIDTH:
       return deviceProfile.getCapability("resolution_width");
   case DEVICE_DISPLAY_RESOLUTION_HEIGHT:
       return deviceProfile.getCapability("resolution_height");
   ...
  }
 } // end of getContextualElement method
 ...
} //end of WURFLAdapter class
```

Figura 3.10 – Código da classe *WURFLAdapter*.

[29] Na última atualização do WURFL disponível em 24/04/2011 foram contabilizados cerca de 14.386 perfis de dispositivos.

[30] http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.43.

Algumas das regras implementadas nos adaptadores de conteúdo para adaptação dos componentes de interface são ilustradas na Figura 3.11. Por exemplo, conforme a primeira regra, a adaptação dos campos de entrada da interface (`inputNode`) só será realizada caso o comprimento dos mesmos (`size`) ultrapasse o número de colunas visíveis na tela do dispositivo. A segunda regra determina que uma imagem contida na interface (`imageNode`) deverá ser adaptada quando sua largura (`width`) ou altura (`height`) extrapolarem a resolução de tela do dispositivo do usuário.

```
Regra 1:
  Condições
    inputNode.size > DEVICE_DISPLAY_COLUMNS_NUMBER
    AND (inputNode.type == "text" OR inputNode.type == "password")
  Ações
    adaptInput(inputNode)
Regra 2:
  Condições
    imageNode.height > DEVICE_DISPLAY_RESOLUTION_HEIGHT
    OR imageNode.width > DEVICE_DISPLAY_RESOLUTION_WIDTH
  Ações
    adaptImage(imageNode)
```

Figura 3.11 – Regras para adaptação de conteúdo.

A Figura 3.12 ilustra o funcionamento da adaptação das interfaces ricas realizada pelos adaptadores de conteúdo, considerando uma aplicação Web desenvolvida com o emprego do *framework JavaServer Faces (JSF)* (SUN, 2010a). De acordo com a Figura 3.12, quando um usuário acessa uma aplicação enviando uma requisição HTTP (①), o *Servlet*[31] da aplicação invoca o método `adapt` da classe `ContentAdapter` (②). O perfil do dispositivo de acesso é recuperado do WURFL (③) e a página Web requisitada é selecionada da versão da interface, armazenada no diretório de páginas Web da aplicação, que melhor se enquadra ao perfil recuperado (④). Em seguida, uma cópia da página escolhida é criada na memória, seus trechos de código mal formados são corrigidos e os componentes que necessitam ser refinados para ajustarem-se ao perfil do dispositivo são identificados aplicando-se as regras de adaptação implementadas nos adaptadores de conteúdo (⑤). As adaptações necessárias são, então, realizadas (⑥), a cópia

[31] http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets.html.

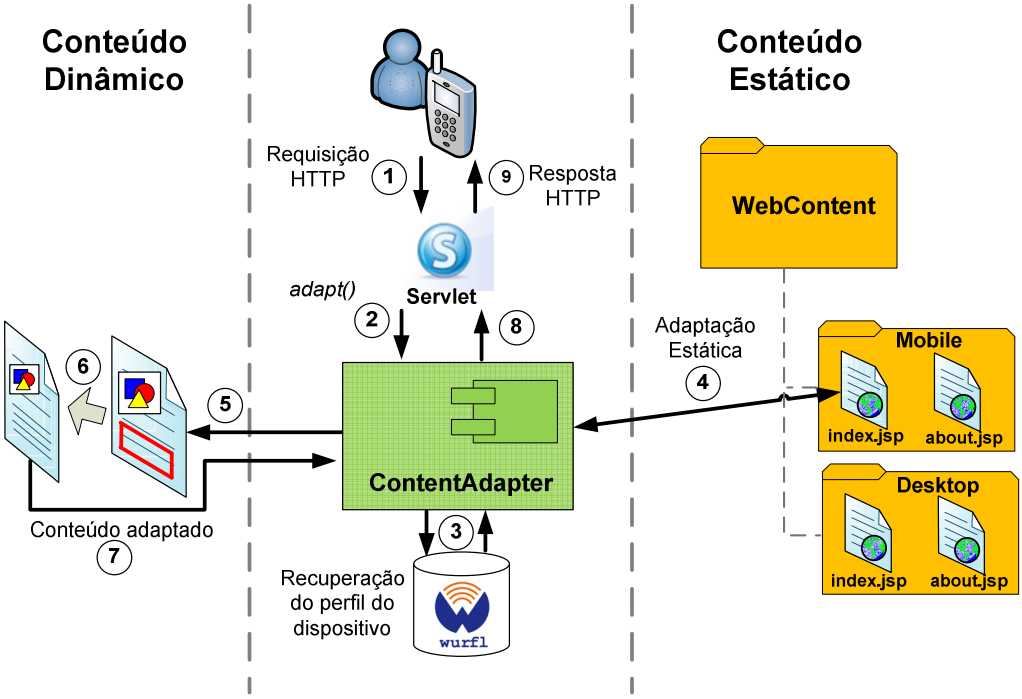


Figura 3.12 – Funcionamento da adaptação das interfaces.

adaptada da página (⑦) é escrita no *buffer* de saída da resposta HTTP (⑧) e finalmente enviada ao dispositivo do usuário por intermédio do *Servlet* (⑨).

Após a execução das atividades da ED do *Model Driven RichUbi*, têm-se os artefatos implementados que apoiarão a construção das aplicações ubíquas com interfaces ricas na EA. Ressalte-se que a ED é uma etapa que produz artefatos para serem reutilizados em diferentes projetos durante a EA, o que contribui para a redução dos esforços de desenvolvimento advinda do reúso dos artefatos. No processo proposto, a ED é executada sempre que for necessária a inclusão de novos componentes de interface rica ao metamodelo (e.g. componente de exibição de vídeo, componentes com comportamento “arrastar e soltar”), a construção de novas transformações M2C necessárias (e.g. *templates* para geração de código WML para celulares) ou a implementação de novos adaptadores de conteúdo.

3.2.2 Engenharia de Aplicação (EA)

Assim como nas abordagens de LPS, o objetivo principal da EA é construir aplicações com foco na reutilização de software. De maneira geral, a EA dedica-se ao estudo das melhores técnicas, processos e métodos para a construção de aplicações, tendo como base o reúso de artefatos. Na EA, os componentes de software previamente desenvolvidos na ED são reutilizados para o desenvolvimento

das aplicações do domínio do problema considerado (GRISS et al., 1998).

Conforme ilustra o diagrama SADT da Figura 3.13, no *Model Driven RichUbi* as atividades definidas para a EA abrangem as disciplinas de *Análise*, *Projeto*, *Implementação e Testes*. Essas disciplinas estendem o ciclo de vida de desenvolvimento dos processos de software convencionais através da inclusão das concepções de MDD e DSM, e concentram-se no desenvolvimento das interfaces ricas adaptativas de aplicações ubíquas com reúso dos artefatos produzidos na ED. O uso do metamodelo dos componentes de interface rica facilita a modelagem das interfaces das aplicações, e as transformações M2C possibilitam automatizar a geração de código tornando grande parte das tarefas do Engenheiro de Aplicação mais ágil. Além disso, os adaptadores de conteúdo fornecem as funcionalidades para adaptação dinâmica das interfaces, permitindo que o foco seja mantido mais no desenvolvimento das funcionalidades relacionadas aos demais requisitos funcionais e não funcionais da aplicação.

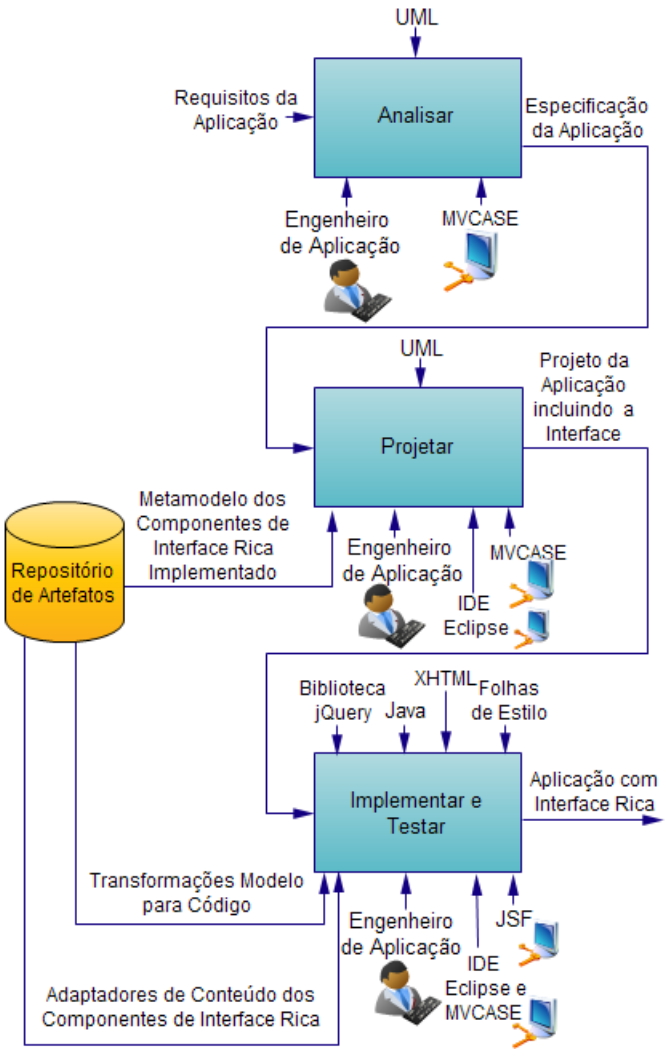


Figura 3.13 – Engenharia de Aplicação no *Model Driven RichUbi*.

Para ilustrar as atividades da EA é construída uma aplicação como exemplo. O metamodelo, as transformações M2C e os adaptadores de conteúdo desenvolvidos na ED foram empregados no desenvolvimento do módulo Web de uma aplicação ubíqua do domínio de Registros Eletrônicos de Saúde (RES), denominado *WebRES*, que permite que médicos cardiologistas acessem os dados pressóricos de seus pacientes utilizando tanto um *desktop* quanto um *smartphone*. A aplicação permite aos médicos analisarem a situação da pressão arterial de seus pacientes de qualquer lugar e a qualquer momento e agir rapidamente em caso de necessidade.

Conforme ilustrado na Figura 3.14, essa aplicação é constituída por três partes: a primeira, que é instalada no dispositivo móvel do paciente, registra e persiste os dados pressóricos informados pelo paciente, e os transmite a um servidor remoto formatados em mensagens *Health Level Seven (HL7) v3*[32], um padrão internacional de protocolo de aplicação utilizado para a troca de dados em ambientes de Cuidado de Saúde; a segunda, que é executada no servidor, recebe as mensagens HL7, trata os dados nelas contidos e persiste os mesmos numa base de dados; a terceira, também executada no servidor, trata-se do módulo Web (*WebRES*) que disponibiliza uma interface para que o profissional de saúde visualize os dados pressóricos de seus pacientes.

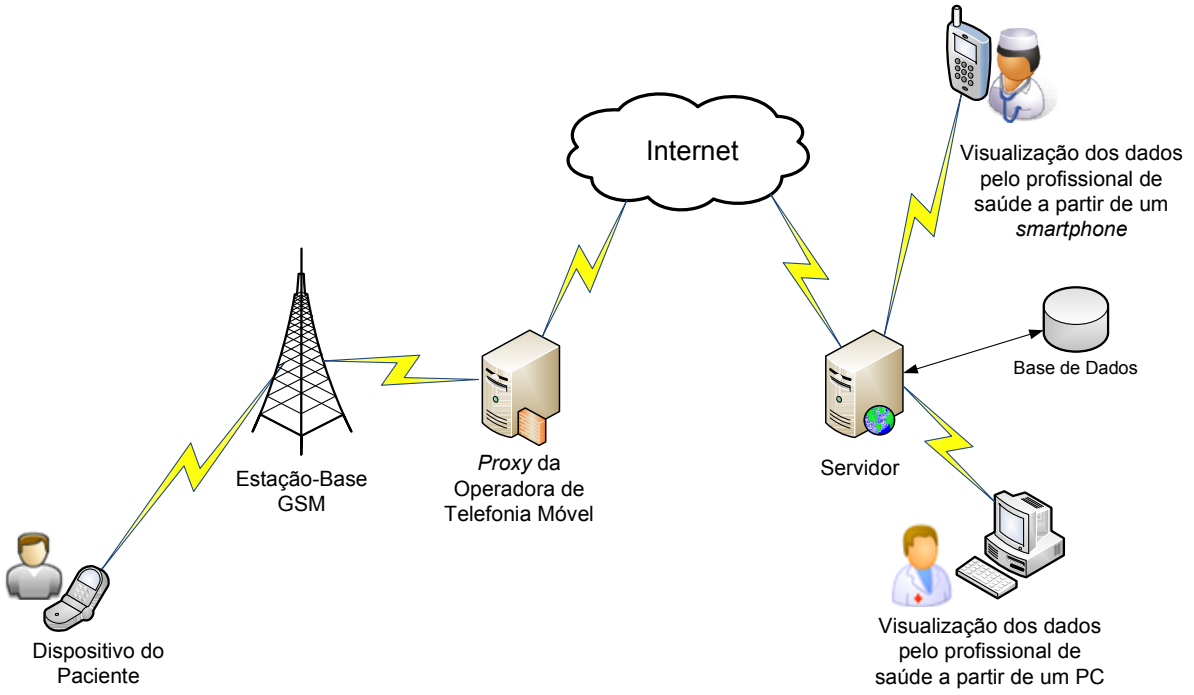


Figura 3.14 – Arquitetura da aplicação *WebRES*.

[32] http://www.hl7.org/.

3.2.2.1 Analisar

Nesta atividade, a aplicação é especificada a partir de seus requisitos. Essa especificação é realizada com o emprego de técnicas UML, através de diagramas de classes e diagramas de casos de uso. A Figura 3.15(a) mostra, por exemplo, um diagrama de casos de uso, construído pelo Engenheiro de Aplicação com auxílio da MVCASE, que especifica os requisitos identificados para o *WebRES*, como os relacionados à autenticação do usuário e à recuperação dos registros pressóricos dos pacientes. A Figura 3.15(b) mostra um diagrama de classes que especifica as entidades *Paciente*, *Médico*, *Pessoa*, *Prontuário* e *RegistroPressório* associadas ao *WebRES*.

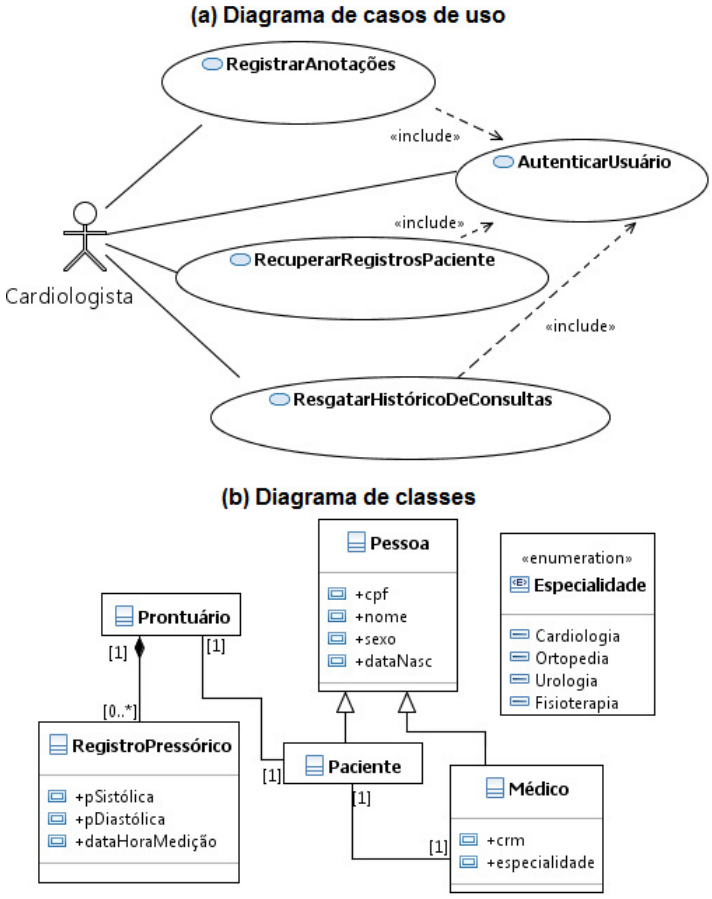


Figura 3.15 – Especificação dos requisitos do *WebRES*.

3.2.2.2 Projetar

Nesta atividade, a especificação da aplicação produzida na atividade precedente é refinada com as tecnologias e plataformas de hardware e software que permitem a implementação da aplicação, como, por exemplo, a plataforma *Java EE*[33]

[33] http://www.oracle.com/us/javaee/.

e o *framework* JSF. Baseado nos diagramas de casos de uso, o Engenheiro de Aplicação também realiza nesta atividade a modelagem das interfaces ricas da aplicação como instância do metamodelo construído na ED. Para tanto, o Engenheiro de Aplicação inclui no modelo das interfaces os componentes apropriados que atendam, tanto quanto possível, a cada um dos casos de uso identificados na especificação da aplicação.

Por exemplo, a Figura 3.16 apresenta, à esquerda, a visão em árvore do modelo construído para as interfaces do *WebRES* através do *plugin* editor de modelos integrado à MVCASE. À direita têm-se os diagramas de objetos ilustrando a instanciação dos componentes do metamodelo implementado. É possível observar no modelo a definição de componentes para atender aos casos de uso "AutenticarUsuário" e "RecuperarRegistrosPaciente" apresentados no diagrama da Figura 3.15(a), tais como o formulário para entrada de dados de autenticação numa página para *login*, e o formulário para busca dos registros pressóricos definido dentro de um componente painel de abas numa página de consultas.

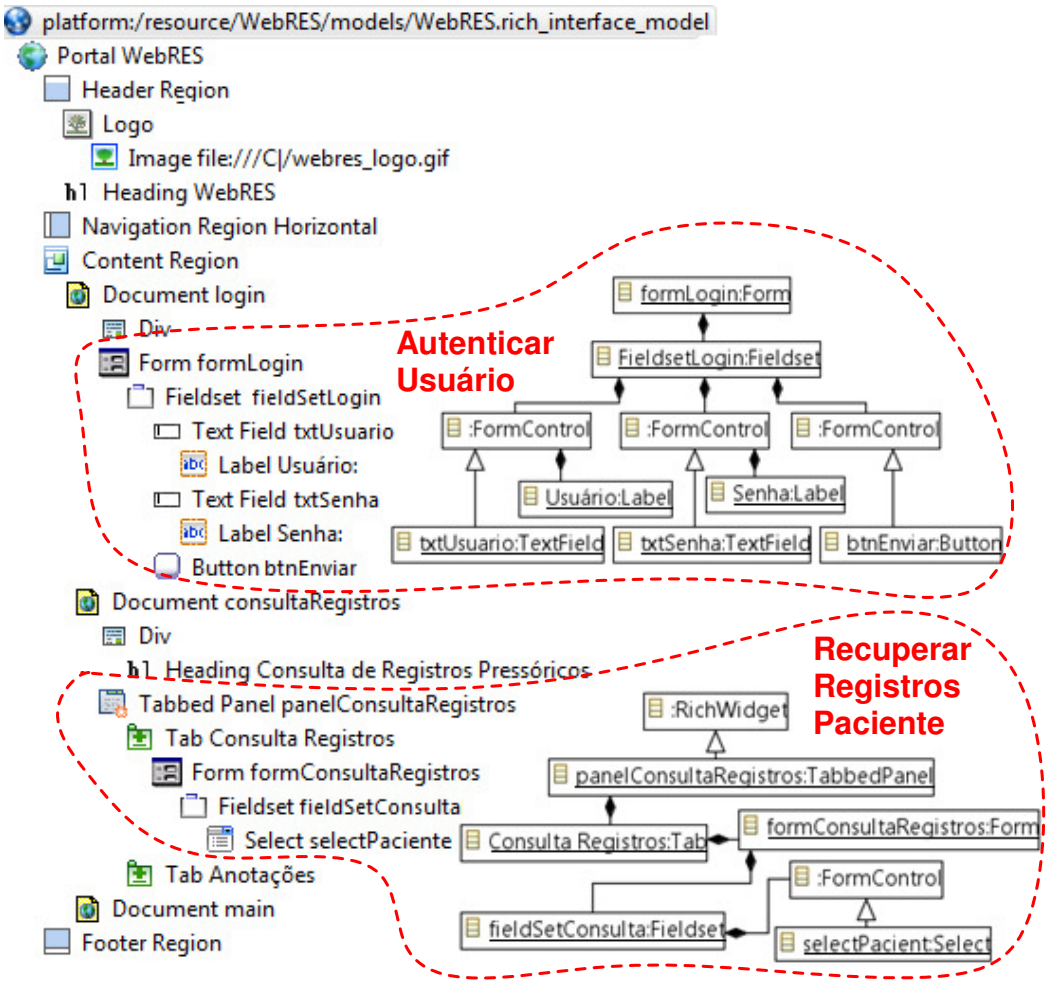


Figura 3.16 – Modelo das interfaces do *WebRES*.

3.2.2.3 Implementar e Testar

Nesta atividade, são realizados a codificação e testes da aplicação. Uma vez que este trabalho está voltado para a construção da camada de apresentação da aplicação ubíqua que varia de acordo com o contexto da interação, o enfoque desta atividade está na implementação e teste das interfaces ricas adaptativas.

No IDE Eclipse, com o auxílio da ferramenta MVCASE, são executadas as transformações M2C sobre os modelos de interface, produzidos na atividade de Projeto, para a geração parcial do código das versões da interface. Por exemplo, conforme ilustra a Figura 3.17, na construção das interfaces do *WebRES*, 71% das linhas de código (*Lines of Code – LOC*) da versão para *desktops* foram geradas através da execução das transformações M2C (≈ 468 LOC), considerando apenas o código das páginas Web e os arquivos JavaScript e folhas de estilo que compõem a interface. Já a versão construída para *smartphones* teve 87% de seu código gerado pelas transformações (≈ 473 LOC).

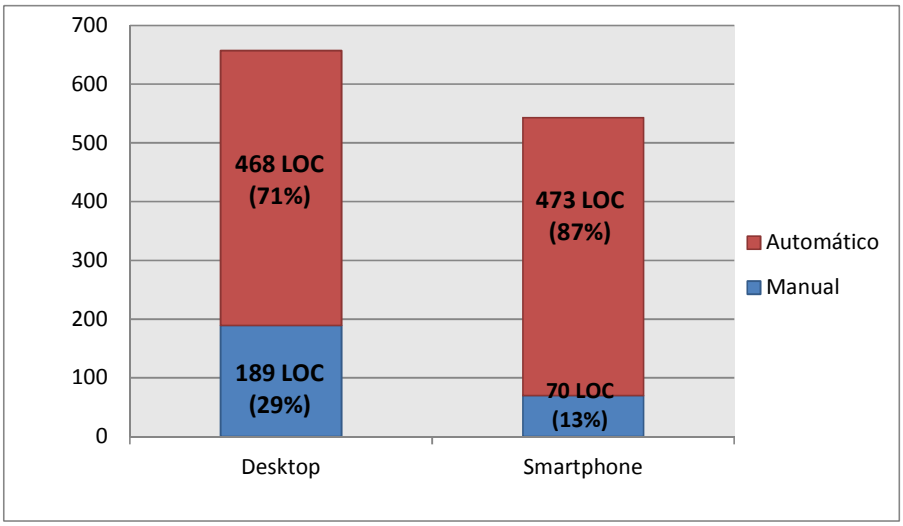


Figura 3.17 – Codificação manual e geração automática de código no *WebRES*.

Por maior que seja a automação na geração de código advinda do uso de transformações, toda aplicação terá suas particularidades e aspectos que fogem ao escopo dos modelos e que necessitarão ser implementados e/ou customizados. Dessa forma, o código parcial gerado deve ser complementado pelo Engenheiro de Aplicação até a finalização da interface, com a inclusão de características não abrangidas nos modelos das interfaces ricas, tais como código para recuperação de dados de fontes externas, folhas de estilo personalizadas, métodos de funções *JavaScript* e implementação da lógica de negócios e navegabilidade entre as interfaces.

Além disso, são incorporados à aplicação os adaptadores de conteúdo construídos na ED para atribuir às interfaces uma característica adaptativa. Como exemplo, a Figura 3.18 apresenta o código do *Servlet* do *WebRES*, chamado `ContextServlet`, onde é realizado o reúso dos adaptadores de conteúdo. O método `doGet`, que intercepta as requisições HTTP, realiza a invocação ao método `adapt` da classe `ContentAdapter`. A partir daí, a classe `ContentAdapter` recupera o perfil do dispositivo, seleciona a versão da interface mais apropriada ao perfil recuperado do contexto, e aplica as adaptações apropriadas na interface da aplicação conforme as peculiaridades do dispositivo de acesso atual.

```
package br.ufscar.dc.webres.servlets;
import br.ufscar.dc.ctx.content.adapter.ContentAdapter;
public class ContextServlet extends HttpServlet {
   private ContentAdapter ca;

   public ContextServlet() {
       super();
   }
   protected void doGet(HttpServletRequest request,
            HttpServletResponse response)
                throws ServletException, IOException {
       if (ca == null) {
           ca = new ContentAdapter(request, response);
       } else {
           ca.reload(request, response);
       }
       ca.adapt();
   }
}
```

Figura 3.18 – Reúso dos adaptadores de conteúdo no *WebRES*.

A Figura 3.19 apresenta os resultados dos testes da execução da página de consultas de registros pressóricos do *WebRES* em um *smartphone* e em um *desktop*. A Figura 3.19(a) mostra a execução da interface no emulador do *iPhone*. Neste caso, a versão da interface entregue foi a versão construída para *smartphones* adaptada às características do *iPhone*, como largura de tela, por exemplo. Para tornar clara a diferença obtida com a adaptação, é mostrada a página adaptada e sua correspondente sem adaptação. Nota-se que, com o serviço de adaptação de conteúdo fornecido pelos adaptadores de conteúdo, o logotipo da aplicação e o painel de abas que contém a tabela que lista os dados pressóricos do paciente foram redimensionados para não extrapolarem a largura de tela do dispositivo. A Figura 3.19(b), por sua vez, mostra a página de consultas visualizada num computador pessoal. Neste caso, a versão da interface selecionada foi aquela construída para *desktops*. Os testes com a execução da interface, usando

(a) Execução no *iPhone*

Sem adaptação Com adaptação

(b) Execução no *desktop*

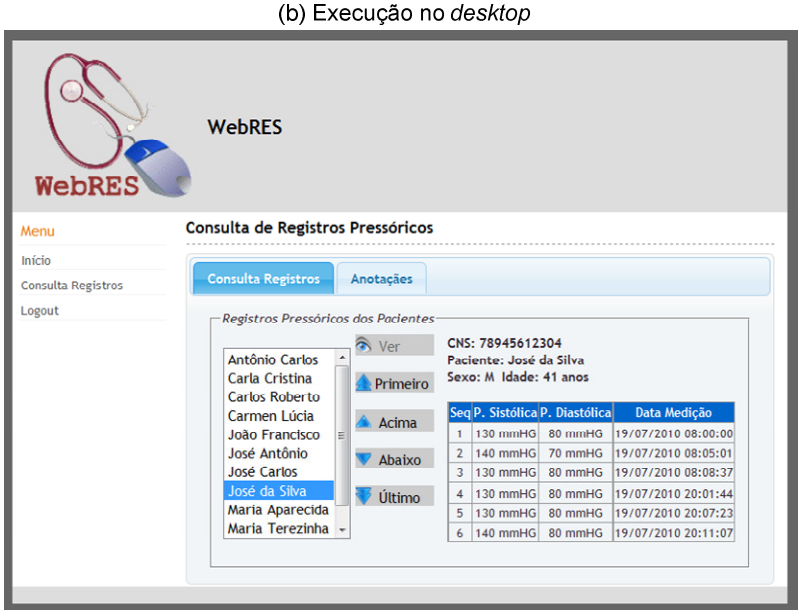


Figura 3.19 – Testes com a execução das interfaces do *WebRES* em dispositivos distintos.

emuladores dos dispositivos-alvo, fornecem o *feedback* para retorno ou não às atividades anteriores do processo.

3.3 Considerações Finais

Este capítulo apresentou o processo de software desenvolvido neste trabalho para apoio à construção de interfaces ricas de aplicações ubíquas que se adaptam

quando visualizadas em diferentes dispositivos. O processo, denominado *Model Driven RichUbi*, baseia-se nas concepções de MDD e DSM, focando-se no uso da modelagem, a partir de um metamodelo do Domínio de Interfaces Ricas, para geração parcial de código das interfaces para diferentes tecnologias, e na adaptação dinâmica dos conteúdos das interfaces conforme o perfil do dispositivo de acesso recuperado do contexto da interação.

O processo proposto é realizado em duas etapas: *Engenharia de Domínio (ED)* e *Engenharia de Aplicação (EA)*. A ED engloba as atividades para a construção dos artefatos reutilizáveis que apoiam o desenvolvimento das aplicações ubíquas com interfaces ricas adaptativas. Esses artefatos incluem um metamodelo que expressa a sintaxe abstrata de uma DSL para a modelagem das interfaces ricas, transformações para geração de código e adaptadores dinâmicos das interfaces. A EA abrange as atividades para o desenvolvimento das aplicações ubíquas com interfaces ricas adaptativas com o reúso dos artefatos produzidos na ED.

Para adaptar as interfaces, no *Model Driven RichUbi* é empregada uma abordagem híbrida de adaptação. No processo são combinadas a geração de código a partir da modelagem em tempo de desenvolvimento (através do reúso do metamodelo e das transformações), com a geração de código em tempo de execução (através do reúso dos adaptadores de conteúdo). O processo contribui para reduzir a complexidade na construção de interfaces ricas adaptativas para os diversos dispositivos da Computação Ubíqua. A abordagem híbrida de adaptação colabora para a redução dos esforços de desenvolvimento, já que boa parte das tarefas de codificação das interfaces para diferentes tecnologias são automatizas e um número menor de versões da interface podem ser desenvolvidas.

As experiências iniciais com o processo *Model Driven RichUbi* serviram para o seu refinamento. O principal refinamento consistiu em organizar e estruturar os adaptadores de conteúdo produzidos na ED em um *framework* para facilitar sua reutilização na fase da Engenharia de Aplicação, bem como para possibilitar a incorporação futura de perfis de outras entidades contextuais, além do dispositivo de acesso, na adaptação. Esse *framework*, chamado *Ubiquitous Context Framework (UbiCon)*, é mais uma contribuição desta pesquisa e seu desenvolvimento é apresentado em detalhes no próximo capítulo.

Capítulo 4

FRAMEWORK UBICON

4.1 Considerações Iniciais

Ambientes de Computação Ubíqua geralmente envolvem diversos fatores que podem influenciar na adaptação da aplicação. Esses fatores podem relacionar-se não apenas com o perfil do dispositivo de acesso, mas envolvem também as preferências e condições do usuário, as características da rede de acesso e do ambiente circundante, dentre outros fatores relevantes. Dessa forma, para operar apropriadamente, uma aplicação ubíqua certamente deverá considerar em sua adaptação as variações que ocorrem tanto no perfil do dispositivo, como no perfil do usuário, da rede, do ambiente e assim por diante.

Neste capítulo apresenta-se o *Ubiquitous Context Framework (UbiCon)*, um *framework* desenvolvido para apoio à construção de interfaces ricas adaptativas que se adaptam de forma híbrida, considerando os perfis de diferentes entidades contextuais envolvidas na execução da aplicação. *Framework* é um termo genérico para uma técnica de reúso orientada a objetos que tipicamente enfatiza a reutilização de arquiteturas e padrões de projeto (KOBRYN, 2000). Um *framework* pode incluir modelos, padrões, classes abstratas e componentes que são especialmente projetados para serem reutilizados e estendidos no desenvolvimento de aplicações específicas. Sua utilização aumenta a produtividade do desenvolvimento de software e a qualidade das aplicações produzidas, pois essas aplicações são construídas com o reúso de uma arquitetura previamente testada (ABI-ANTOUN, 2007).

O *framework UbiCon* fornece aos desenvolvedores um “esqueleto” que pode ser instanciado para o desenvolvimento de aplicações ubíquas que adaptam sua interface de maneira híbrida através da combinação de diferentes perfis. Para o desenvolvimento das funcionalidades de adaptação fornecidas pelo *UbiCon* foi feita a reutilização dos adaptadores de conteúdo desenvolvidos durante a etapa de Engenharia de Domínio do processo *Model Driven RichUbi*, organizando-os de forma apropriada para facilitar sua reutilização e extensão. O *UbiCon* encapsula as tarefas de manipulação do contexto e adaptação de conteúdo das interfaces ricas, de forma que os desenvolvedores de software possam concentrar-se em aspectos mais importantes dos requisitos de negócio de suas aplicações. O *UbiCon* pode, então, ser empregado na etapa de EA do processo *Model Driven RichUbi* de modo a estender a adaptação das interfaces ricas para atender aos variados perfis de entidades contextuais usualmente presentes em um ambiente ubíquo.

Este capítulo está organizado da seguinte forma: a Seção 4.2 apresenta uma visão geral do *framework UbiCon* destacando sua estrutura e organização de seus módulos, os quais são detalhados nas subseções seguintes (Seções 4.2.3 a 4.2.1); a Seção 4.3 descreve o funcionamento e reúso do *UbiCon* no processo *Model Driven RichUbi*; e a Seção 4.4 tece algumas considerações finais, encerrando o capítulo.

4.2 Ubiquitous Context Framework (UbiCon)

O *UbiCon* abstrai as funcionalidades relacionadas à manipulação do contexto e fornece serviços para adaptar as interfaces ricas de uma aplicação ubíqua aplicando a abordagem de adaptação híbrida. O *UbiCon* foi projetado de forma a contemplar a obtenção do contexto de diferentes entidades envolvidas na execução de uma aplicação ubíqua. A Figura 4.1 mostra o diagrama de componentes do *UbiCon*, cuja implementação foi realizada usando a linguagem de programação Java. Sua estrutura foi organizada em quatro módulos: *Adaptação de Conteúdo, Disseminação, Processamento* e *Aquisição*. Essa estrutura é baseada na *Arquitetura de Contexto* proposta por Santos (2008), apresentada no Capítulo 2 (Seção 2.4.2).

Na Figura 4.1 são destacados, em sombreado, os componentes implementados, responsáveis pela obtenção do contexto e adaptação baseada no

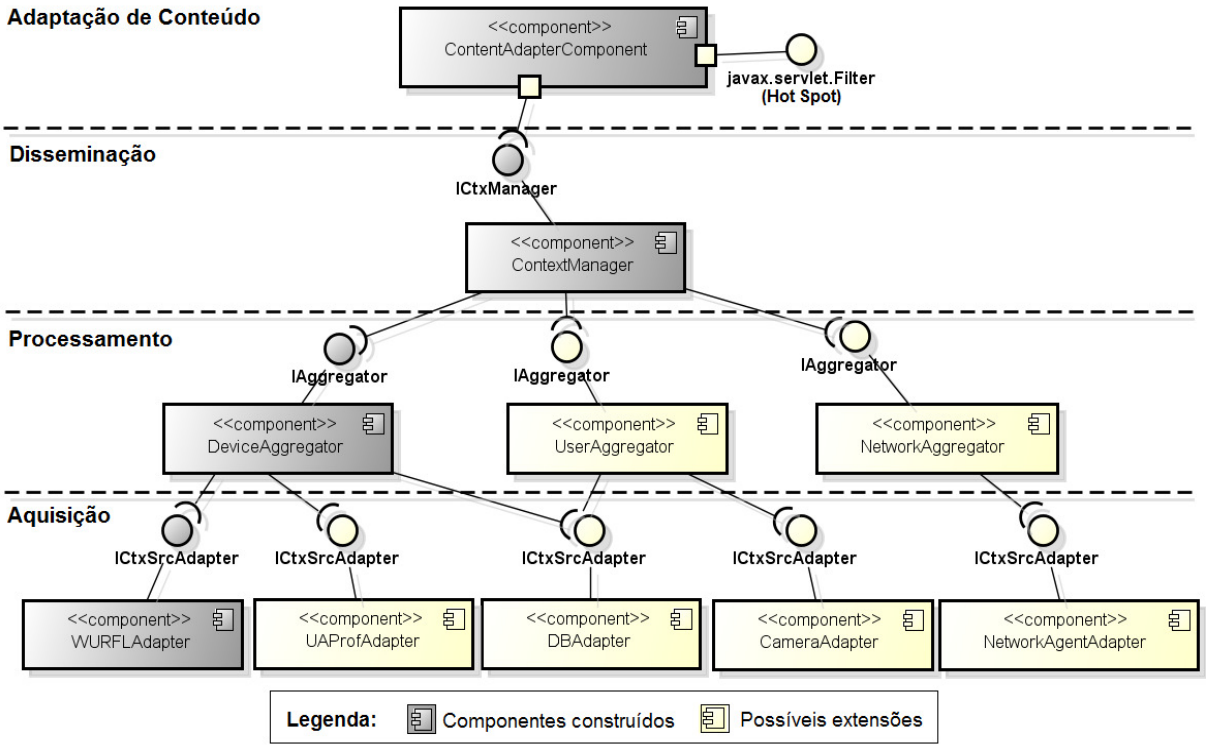


Figura 4.1 – Diagrama de componentes do *UbiCon*.

perfil do dispositivo. Os demais componentes, ainda não implementados, foram projetados para atender a outros perfis, tais como o perfil do usuário, da rede de acesso, do ambiente circundante, e outros. Portanto, o *UbiCon* foi construído de forma extensível o suficiente para permitir que fontes de contexto que fornecem *Elementos Contextuais (ECs)* relacionados a tais perfis possam ser mais facilmente adicionadas ao *framework*. Dessa maneira, é possível estender o *UbiCon* para prover serviços de adaptação baseados na combinação de diferentes perfis.

As seções seguintes detalham cada um dos módulos que compõe o *framework UbiCon*.

4.2.1 Módulo de Adaptação de Conteúdo

Este módulo é responsável pela adaptação das interfaces ricas das aplicações ubíquas considerando os vários aspectos importantes que podem ser extraídos do contexto da interação, tais como as características do dispositivo, as preferências dos usuários, o estado de congestionamento da rede, o ambiente circundante, e assim por diante.

No desenvolvimento deste módulo foram reutilizados os conhecimentos adquiridos com a construção dos adaptadores de conteúdo desenvolvidos na etapa de

ED do *Model Driven RichUbi*. Neste módulo foram encapsulados os adaptadores de conteúdo e uma interface, disponível através do *filtro*[34] `javax.servlet.Filter`[35], foi provida para acesso e reutilização no desenvolvimento das aplicações.

Como consumidor de contexto, o *Módulo de Adaptação de Conteúdo* consome os ECs obtidos através dos módulos inferiores na hierarquia do *UbiCon* para prover as funcionalidades contextualizadas de adaptação. Conforme mostra a Figura 4.2, este módulo possui duas classes principais: `ContentAdapter` e `ContentAdapterFilter`. Para realizar a adaptação das interfaces, a classe `ContentAdapter` consome os ECs relevantes associados com as entidades envolvidas no contexto de execução da aplicação ubíqua, requisitando-os ao componente `ContextManager` do *Módulo de Disseminação*, através da interface `ICtxManager`. A classe `ContentAdapter` implementa os métodos definidos na interface `IContentAdapter`, os quais correspondem aos adaptadores de conteúdo. Na implementação desses métodos houve o reaproveitamento das funcionalidades desenvolvidas nos adaptadores de conteúdo construídos na etapa

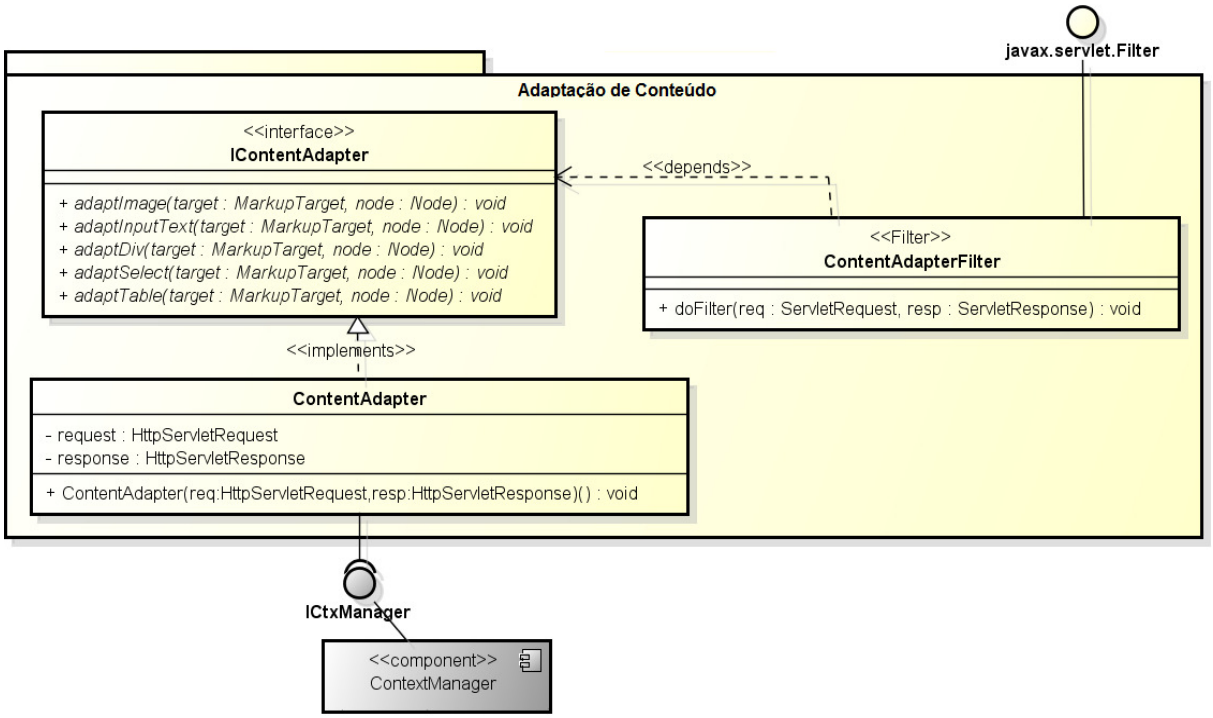


Figura 4.2 – Módulo de Adaptação de Conteúdo do *UbiCon*.

[34] Um *filtro* intercepta dinamicamente as requisições e respostas de uma aplicação Web para transformar ou utilizar a informação que é transportada. Dessa forma, *filtros* podem ser empregados para a formatação os conteúdos que são retornados pela aplicação ao cliente, o que os torna apropriados para transformar as páginas Web requisitadas com o intuito de realizar a adaptação das interfaces.

[35] http://java.sun.com/blueprints/corej2eepatterns/Patterns/InterceptingFilter.html.

de ED do processo *Model Driven RichUbi*. Os serviços contextualizados de adaptação de conteúdo fornecidos pelo `ContentAdapter` são disponibilizados à classe `ContentAdapterFilter` por meio da interface `IContentAdapter`. Essa classe é responsável por interceptar as requisições feitas à aplicação com o intuito de processar a adaptação de conteúdo. Para isso, o `ContentAdapterFilter` implementa o método abstrato `doFilter` da interface `javax.servlet.Filter`, que captura qualquer requisição feita à aplicação.

Após interceptar a requisição, o método `doFilter` implementado na classe `ContentAdapterFilter` inicia a adaptação híbrida da interface: primeiro, seleciona a versão estática da interface mais adequada conforme o contexto (perfil do dispositivo, usuário, rede, etc) e, segundo, invoca os serviços apropriados de adaptação de conteúdo do `ContentAdapter` para ajustar os diversos componentes da interface. Para permitir a manipulação e transformação das interfaces em tempo de execução, no `ContentAdapter` também são empregadas as APIs Java DOM e *TagSoup* que foram usadas no desenvolvimento dos adaptadores de conteúdo.

4.2.2 Módulo de Disseminação

Este módulo, mostrado na Figura 4.3, é composto por um componente principal, chamado `ContextManager`, o qual provê os ECs manipulados no *Módulo de Processamento* aos consumidores de contexto interessados (neste caso, o *Módulo de Adaptação de Conteúdo*) através da interface `ICtxManager`. Os consumidores de

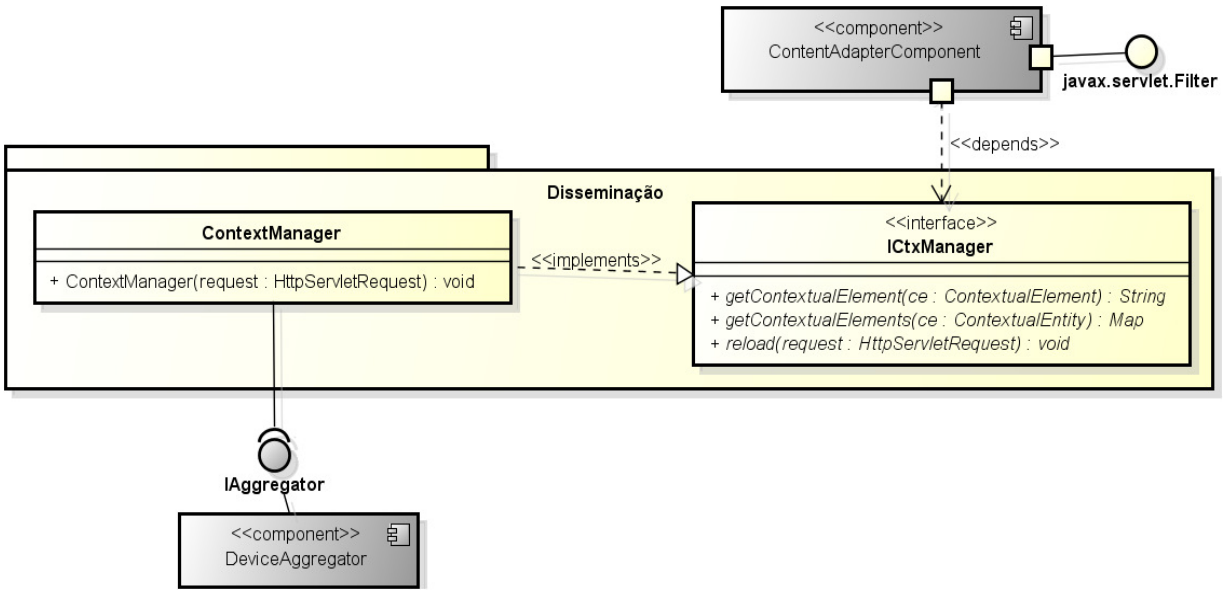


Figura 4.3 – Módulo de Disseminação do *UbiCon*.

contexto podem requisitar o *Módulo de Disseminação* para obterem o valor atual de um EC específico ou todos os ECs relacionados a uma dada entidade contextual. A partir da perspectiva dos consumidores de contexto, o componente `ContextManager` atua como uma *fachada*, uma vez que oculta a manipulação dos diversos componentes do *Módulo de Processamento* (chamados *agregadores*) e fornece uma única interface para obtenção dos ECs. A tarefa de determinar a partir de qual agregador requisitar um EC específico é encapsulada pelo `ContextManager`.

A Figura 4.4 mostra um trecho de código do `ContextManager`. O método `getContextualElement` seleciona o agregador mais apropriado para obter o EC requisitado, sendo o mesmo identificado através do item da enumeração `ContextualElement` passado como parâmetro para esse método. Considerando que cada EC pode estar relacionado com quaisquer entidades contextuais, as quais devem possuir um agregador correspondente no *Módulo de Processamento*, o rótulo do EC é utilizado para auxiliar na identificação do agregador apropriado que representa a entidade com a qual o EC está associado. Por exemplo, se o rótulo de um dado EC começa com a sequência *"DEVICE_"*, então o componente `DeviceAggregator` é selecionado para recuperar esse elemento contextual. Conforme também apresentado na Figura 4.4, o método `getContextualElements` retorna o conjunto dos ECs relacionados com a entidade contextual fornecida como parâmetro. Essa entidade é

```
package br.ufscar.dc.ubicon.manager;
import br.ufscar.dc.ubicon.aggregators.DeviceAggregator;
public final class ContextManager implements IContextManager {
    private DeviceAggregator deviceAggregator;

    public ContextManager(HttpServletRequest request) {
        deviceAggregator = new DeviceAggregator(request);
    } // end of constructor

    @Override
    public String getContextualElement(ContextualElement ce) {
        if (ce.toString().startsWith("DEVICE_")) {
            return deviceAggregator.getContextualElement(ce);
        }
        return null;
    } // end of getContextualElement method

    @Override
    public Map<String, String> getContextualElements(ContextualEntity ce) {
        if (ce == ContextualEntity.DEVICE) {
            return deviceAggregator.getContextualElements();
        }
        return null;
    } // end of getContextualElements method
    ...
}// end of ContextManager class
```

Figura 4.4 – Trecho do código do *ContextManager*.

identificada através de um item da enumeração `ContextualEntity`, a qual lista todas as entidades contextuais abrangidas pelo *framework UbiCon*.

4.2.3 Módulo de Processamento

Este módulo, representado na Figura 4.5, inclui os componentes *agregadores* que processam e agrupam os ECs de acordo com a entidade contextual que caracterizam. O processamento envolve a transformação, se necessária, dos ECs brutos em um formato ou nível de abstração mais adequado para ser utilizado pelos consumidores de contexto, bem como a resolução de conflitos entre valores distintos para o mesmo EC obtido a partir de fontes de contexto diferentes (BALDAUF et al., 2007). Cada agregador é, então, uma abstração de uma entidade contextual, fornecendo ECs refinados relacionados com a entidade a que está associado (DEY et al., 2001).

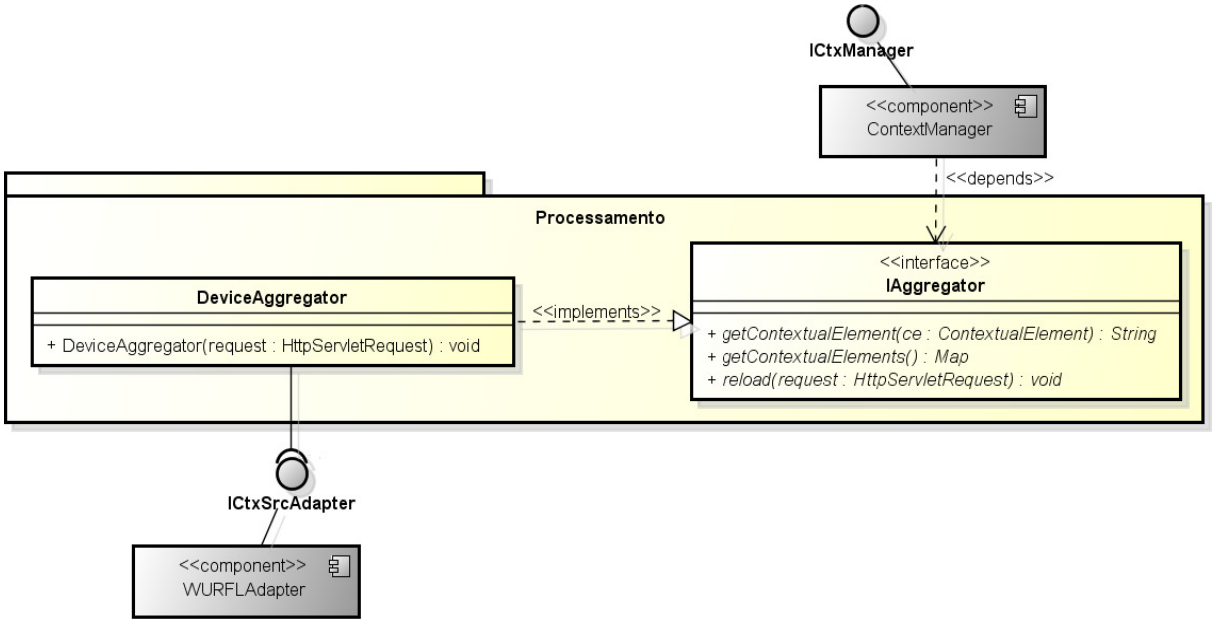


Figura 4.5 – Módulo de Processamento do *UbiCon*.

No *framework UbiCon*, os agregadores são construídos utilizando o padrão de projeto denominado *Facade* (GAMMA et al., 1994). Assim, cada agregador atua como uma fachada que oculta a complexidade inerente à manipulação dos vários componentes do *Módulo de Aquisição* (chamados *adaptadores*), fornecendo uma interface única para a obtenção dos ECs de uma determinada entidade contextual. Por exemplo, o componente `DeviceAggregator` da Figura 4.1 representa o dispositivo de acesso e interage diretamente com todos os componentes adaptadores do *Módulo de Aquisição* que acessam as fontes de contexto que fornecem ECs relacionados ao perfil do dispositivo.

Todos os componentes agregadores devem implementar a interface `IAggregator`, a qual define métodos abstratos (`getContextualElement` e `getContextualElements`) para permitir a recuperação dos ECs a partir de diferentes fontes de contexto através de um único ponto. As tarefas de decidir a partir de qual fonte de contexto recuperar um dado EC, transformar os ECs em um nível de abstração apropriado quando necessário e resolver os conflitos entre valores discrepantes de um mesmo EC são, portanto, encapsuladas pelos agregadores. Por exemplo, quando dois valores diferentes para o EC rotulado *"largura de resolução de tela"* são adquiridos a partir de perfis do dispositivo armazenados, respectivamente, em um arquivo de configuração e em uma base de dados externa, o agregador deve decidir qual desses valores deve ser retornado ao consumidor de contexto requisitante. Neste caso, informações adicionais, como a confiabilidade das fontes de contexto ou seu nível de atualização, podem auxiliar na tomada de decisão do agregador (SANTOS, 2008). Ainda, esses valores recuperados podem estar representados em diferentes unidades de medida, como pixels e polegadas, por exemplo. Neste caso, o agregador deve estabelecer uma unidade de medida padrão para o EC em questão e prover mecanismos para desempenhar as conversões necessárias. Em ambas as situações, o agregador deve agir como uma "caixa-preta" que fornece os ECs refinados aos módulos acima na hierarquia do *framework*.

Conforme ilustrado no diagrama de componentes da Figura 4.1, dada a característica extensível do *UbiCon*, à medida em que novas fontes de contexto que fornecem ECs associados a outras entidades contextuais (e.g. usuário, rede de acesso) são incluídas no *framework*, novos componentes agregadores correspondentes a essas entidades devem também ser construídos para estender o *UbiCon*.

4.2.4 Módulo de Aquisição

Este módulo compreende os componentes que acessam diretamente as fontes de contexto (e.g. bases de dados, *drivers* de sensores, agentes de software) para adquirir os ECs brutos, i.e., no formato entregue por essas fontes. Esses componentes são chamados de *adaptadores* (VIEIRA et al., 2009), pois encapsulam os detalhes de acesso a fontes de contexto distintas, conectando o *framework* a fontes de contexto heterogêneas e fornecendo métodos genéricos para adquirir os

ECs disponíveis. Esses métodos são disponibilizados através da interface `ICtxSrcAdapter`.

As fontes de contexto são, por natureza, heterogêneas, autônomas e dinâmicas devido ao fato de que existem independentemente da aplicação sendo desenvolvida (SANTOS, 2008). Dessa forma, no *UbiCon* deve existir um componente adaptador apropriado para cada fonte de contexto. Os adaptadores utilizam *drivers* e APIs das fontes de contexto correspondentes para acessar suas funcionalidades internas e recuperar os ECs disponíveis. Essa granularidade torna o *framework UbiCon* mais extensível, uma vez que isso facilita a inclusão de novas fontes de contexto ou a remoção daquelas que não estão mais disponíveis ou tornaram-se obsoletas.

A Figura 4.6 mostra o diagrama de classes do *Módulo de Aquisição* implementado no *UbiCon*. No *UbiCon* a base de dados WURFL foi adotada como principal fonte de contexto para obtenção dos perfis dos dispositivos. Conforme mostrado na Figura 4.6, o componente chamado `WURFLAdapter` acessa o WURFL para obter os ECs associados ao perfil do dispositivo. Na construção desse componente foi também reaproveitado o código do componente homônimo implementado durante o desenvolvimento dos adaptadores de conteúdo na etapa de

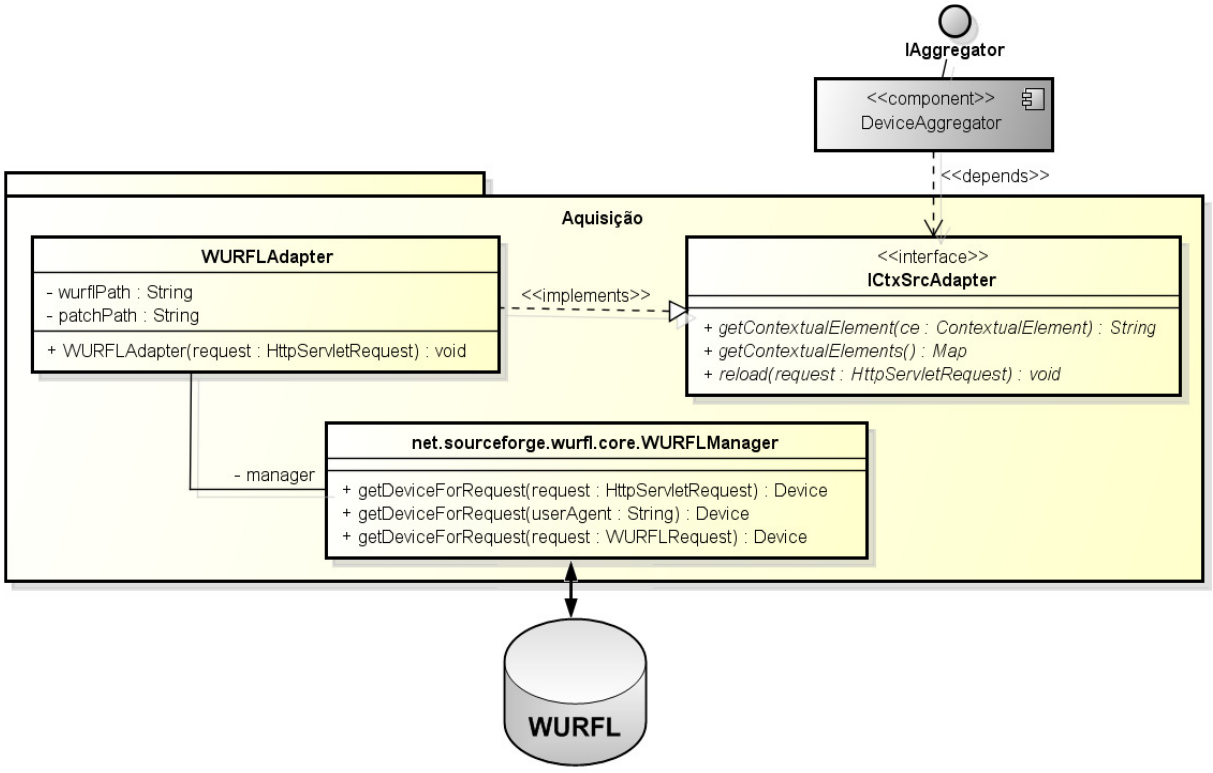


Figura 4.6 – Módulo de Aquisição do *UbiCon*.

ED do processo *Model Driven RichUbi*. A Figura 4.7 mostra um trecho de código do `WURFLAdapter` implementado no *UbiCon*. Tal como anteriormente, a API Java do WURFL é empregada para acesso às informações nele armazenadas. A captura do contexto é baseada no campo *User-Agent* da requisição HTTP originada pelo dispositivo de acesso do usuário. Através dessa informação contextual é possível recuperar os demais elementos contextuais relacionados ao perfil do dispositivo contidos no WURFL. Para satisfazer à interface `ICtxSrcAdapter`, o componente `WURFLAdapter` implementa os métodos `getContextualElement` e `getContextualElements`. O primeiro realiza a recuperação de um EC específico identificado pelo item da enumeração `ContextualElement` recebido como parâmetro. O último retorna o conjunto de todos os ECs disponíveis na fonte de contexto manipulada pelo adaptador – neste caso a base de dados WURFL.

```
package br.ufscar.dc.ubicon.ctxsources.adapters;
import net.sourceforge.wurfl.core.DefaultDeviceProvider;
public clas  WURFLAdapter implements ICtxSrcAdapter {
 private String wurflPath = "wurfl.zip";
 private String patchPath = "web_browsers_patch.xml";
 private WURFLManager manager;
 private Device deviceProfile;
 ...
 public WURFLAdapter (HttpServletRequest request) {
   ...
   deviceProfile = manager.getDeviceForRequest(
                       request.getHeader("User-Agent"));
 } //end of constructor

 @Override
 public String getContextualElement(ContextualElement ce) {
  switch(ce) {
   case DEVICE_DISPLAY_COLUMNS_NUMBER:
      return deviceProfile.getCapability("columns");
   case DEVICE_DISPLAY_RESOLUTION_WIDTH:
      return deviceProfile.getCapability("resolution_width");
   case DEVICE_PRODUCT_INFO_IS_MOBILE_DEVICE:
      return deviceProfile.getCapability("is_wireless_device");
   ...
  }
 } // end of getContextualElement method
 @Override
 public Map<String,String> getContextualElements() {
     return deviceProfile.getCapabilities();
 } //end of getContextualElements method
 ...
} //end of WURFLAdapter class
```

Figura 4.7 – Trecho do código do *WURFLAdapter*.

Devido à natureza extensível pela qual o *UbiCon* foi projetado, outras fontes de contexto, tais como *User Agent Profile (UAProf)* (OMA, 2001) e bases de dados de perfis, podem ser adicionadas futuramente ao *framework* para complementar a

aquisição dos ECs associados ao perfil do dispositivo. Isso implica na implementação de adaptadores específicos para acesso a essas fontes de contexto, conforme ilustrado no diagrama de componentes da Figura 4.1. A característica extensível do *UbiCon* também simplifica a incorporação de fontes de contexto que fornecem ECs relacionados a outras entidades, tais como o usuário (e.g. banco de dados de perfis, *drivers* de câmeras de vídeo que capturam a localização do usuário dentro de um cômodo), a rede de acesso (e.g. agentes de software que monitoram o tráfego da rede), dentre outros. Essas extensões permitem tornar o *framework* mais abrangente e possibilita fornecer serviços de adaptação de interfaces contextualizados baseados em diferentes perfis.

4.3 Funcionamento e Reutilização do UbiCon

A Figura 4.8 ilustra o funcionamento da adaptação híbrida com o uso do *UbiCon*, considerando apenas o perfil do dispositivo do usuário. Quando um cliente acessa a aplicação, a requisição HTTP é interceptada pelo componente `ContentAdapterFilter` do *UbiCon* (①), o qual inicia o processamento da adaptação. Em seguida, o componente `ContextManager` é invocado para obter o perfil do dispositivo de acesso (②), o qual é recuperado do WURFL (③). A seguir, a parte estática da adaptação é realizada através da seleção da página Web requisitada a partir da versão mais apropriada ao perfil do dispositivo obtido (④). A adaptação dinâmica é então iniciada, com a invocação das funcionalidades de adaptação fornecidas pelo componente `ContentAdapter` (⑤). Uma cópia da página escolhida é criada na memória, e os trechos da interface que necessitam ser refinados para atender ao perfil do dispositivo recuperado são identificados aplicando-se as regras implementadas nos serviços de adaptação (⑥). Os ajustes necessários (e.g. redimensionamento, transcodificação) são aplicados (⑦), a cópia da página com o conteúdo adaptado é escrita no fluxo de saída da resposta HTTP (⑧), e finalmente enviada ao dispositivo do usuário (⑨).

Para determinar a versão mais apropriada da interface durante a parte estática da adaptação, o *UbiCon* realiza a leitura de um arquivo de configuração XML, chamado *context-rules.xml*, o qual contém definições de regras contextuais estáticas

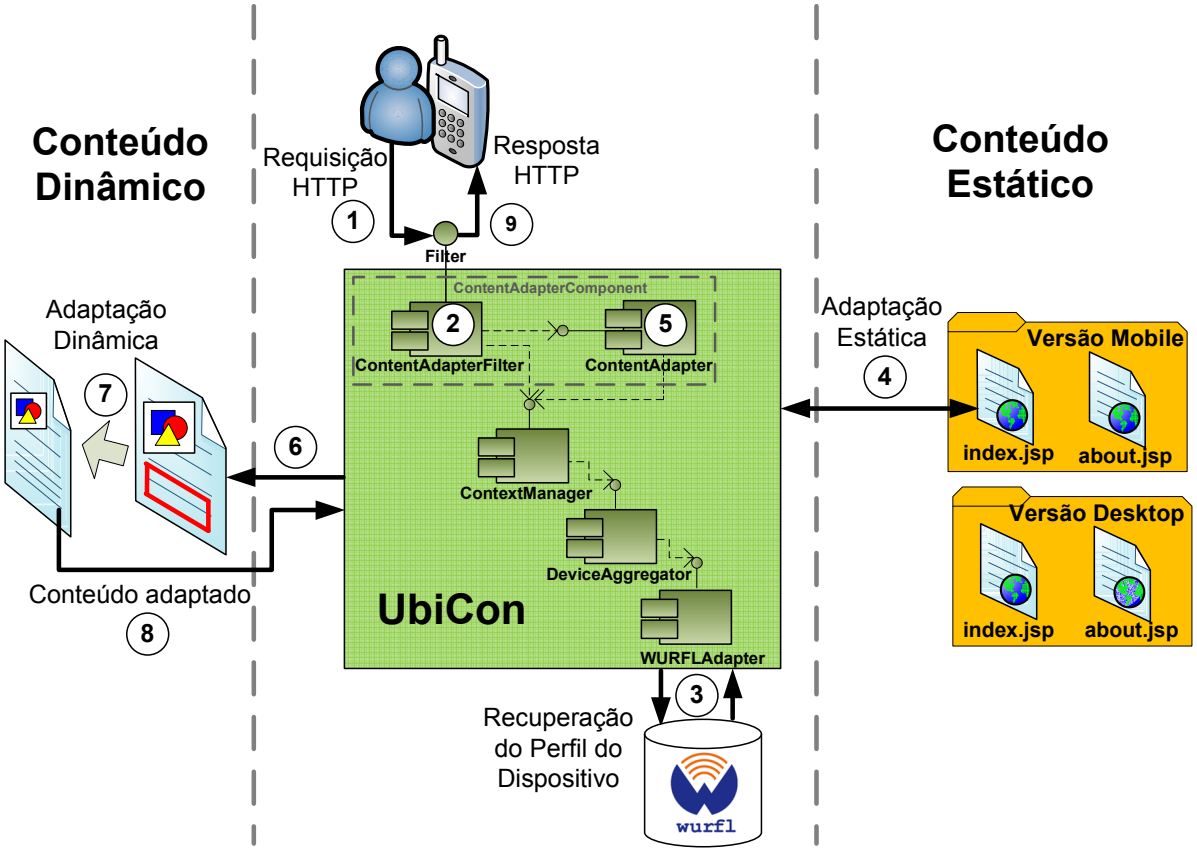


Figura 4.8 – Funcionamento da adaptação híbrida com o *framework UbiCon*.

que determinam a escolha da versão adequada que atenda aos requisitos do contexto. Este arquivo é preparado pelo Engenheiro de Aplicação, que decide, em tempo de desenvolvimento, a melhor combinação de ECs para definir a escolha de cada versão desenvolvida para a interface. Conforme mostrado no arquivo *context-rules.xml* ilustrado na Figura 4.9, as regras para a seleção de uma determinada versão da interface são expressas nos nós `contextRule`, os quais especificam os ECs e seus respectivos valores que determinam a escolha da versão da interface relacionada com o nó de rótulo `context` no qual estão contidos. Por exemplo, conforme se observa no trecho destacado do arquivo da Figura 4.9, a versão da interface desenvolvida para *desktops* está associada com o nó `context` cujo atributo `interfaceID` contém o valor "Desktop". Esse valor corresponde exatamente ao nome da pasta na qual essa versão está armazenada no diretório da aplicação. Os nós filhos de rótulo `contextRule` indicam que tal versão será selecionada somente no caso dos ECs `DEVICE_PRODUCT_INFO_IS_MOBILE_DEVICE` e `DEVICE_MARKUP_XHTML_SUPPORT` recuperados do contexto estejam com valores iguais (`eq`) a "falso" (`false`) e "verdadeiro" (`true`), respectivamente. Conforme pode-se observar na Figura 4.9, também é possível estabelecer o grau de relevância (`relevanceLevel`) de cada EC

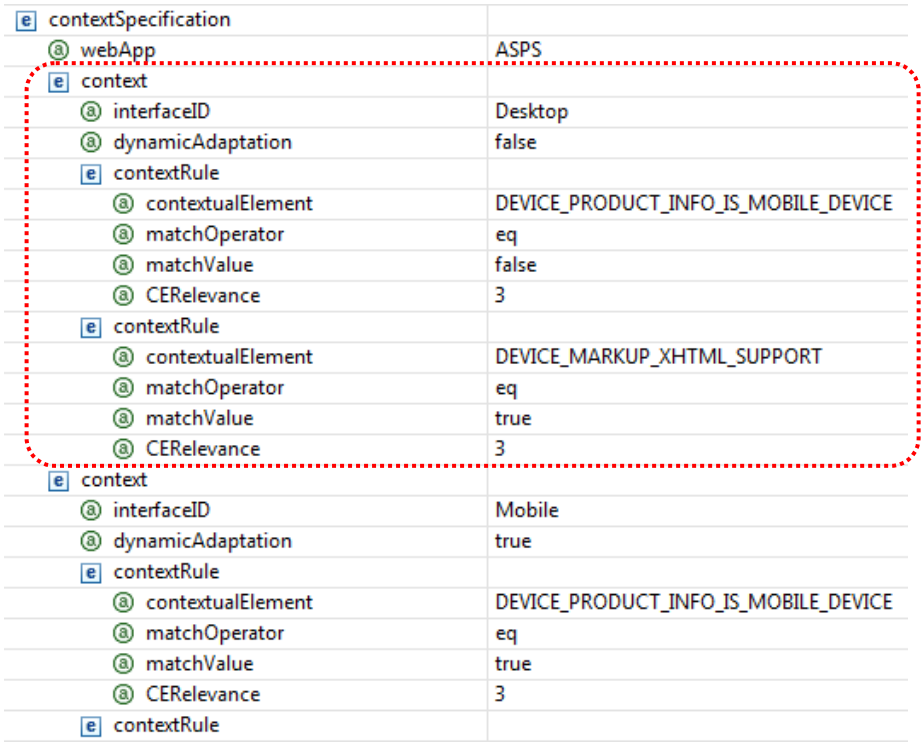


Figura 4.9 – Arquivo *context-rules.xml*.

em relação à decisão de escolha da versão da interface, o qual varia entre alto (`3`), médio (`2`) ou baixo (`1`). No arquivo de exemplo, o nível de relevância do EC rotulado `DEVICE_PRODUCT_INFO_IS_MOBILE_DEVICE` na decisão de escolha da versão para *desktops* foi configurado como "alto" (`3`), uma vez que tal informação contextual possui grande importância na determinação dessa escolha.

Durante a leitura do arquivo *context-rules.xml*, o *UbiCon* calcula, para cada versão da interface, as quantidades de regras contextuais que coincidem com os valores dos ECs recuperados do contexto. Essas quantidades são agrupadas por grau de relevância conforme especificado em cada EC nos nós `contextRule`. O *UbiCon* então executa o algoritmo apresentado na Figura 4.10, escrito em pseudocódigo. Nesse algoritmo, a variável `R` representa o grau de relevância, variando entre `1` e `3`; a variável `I` é um índice que representa a versão da interface cujo valor varia de `1` a `n`, onde tal valor associa-se com uma determinada versão de acordo com a sequência em que seu nó `context` correspondente aparece no arquivo *context-rules.xml*; e a variável `score[I][R]` é um vetor bidimensional que armazena as quantidades calculadas de regras contextuais para cada versão da interface por grau de relevância dos ECs. O objetivo do algoritmo é retornar o índice da versão que possui o maior número de regras contextuais satisfeitas com maiores níveis de relevância. Se todas

```
VAR
  largest, indexChosen: INTEGER;
  found: BOOLEAN;
BEGIN
  found ← false;
  indexChosen ← 1;
  IF n > 1 THEN
    BEGIN
      FOR R ← 3 UNTIL 1 DO
        BEGIN
          largest ← score[1][R];
          FOR I ← 2 UNTIL n DO
            BEGIN
              IF score[I][R] > largest THEN
                BEGIN
                  largest ← score[I][R];
                  indexChosen ← I; found ← true;
                END IF
              ELSE IF score[I][R] < largest THEN
                found ← true;
            END FOR
          IF found = true THEN
            BEGIN
              break the most external FOR loop;
            END IF
        END FOR
    END IF
    RETURN indexChosen;
END
```

Figura 4.10 – Algoritmo para escolha da versão estática da interface.

as versões possuírem a mesma quantidade de regras satisfeitas em cada grau de relevância, então o índice da primeira versão é retornado.

Também é possível observar no arquivo *context-rules.xml* da Figura 4.9 a definição do atributo `dynamicAdaptation` em cada nó `context`. Esse atributo indica quando certa versão da interface deverá passar pela etapa de adaptação dinâmica após sua escolha. A presença desse atributo é justificada pela possibilidade de se construir versões da interface altamente especializadas para dispositivos específicos, as quais, portanto, não requerem refinamentos adicionais. Pode-se, por exemplo, considerando o tipo de dispositivo do público-alvo que utilizará a aplicação com mais frequência, desenvolver uma versão específica para o dispositivo mais usual e definir que tal versão não deverá ser adaptada em tempo de execução.

Após sua implementação, o *framework Ubicon* é reutilizado no desenvolvimento das aplicações ubíquas com interfaces ricas adaptativas. No processo Model Driven RichUbi, o *UbiCon* é um importante mecanismo de suporte à adaptação das interfaces na etapa de EA, conforme destacado na Figura 4.11. O *UbiCon* contribui para facilitar o desenvolvimento, no sentido de que o desenvolvedor precisa apenas conhecer sua interface (*hot spot* do *framework*) para a reutilização dos

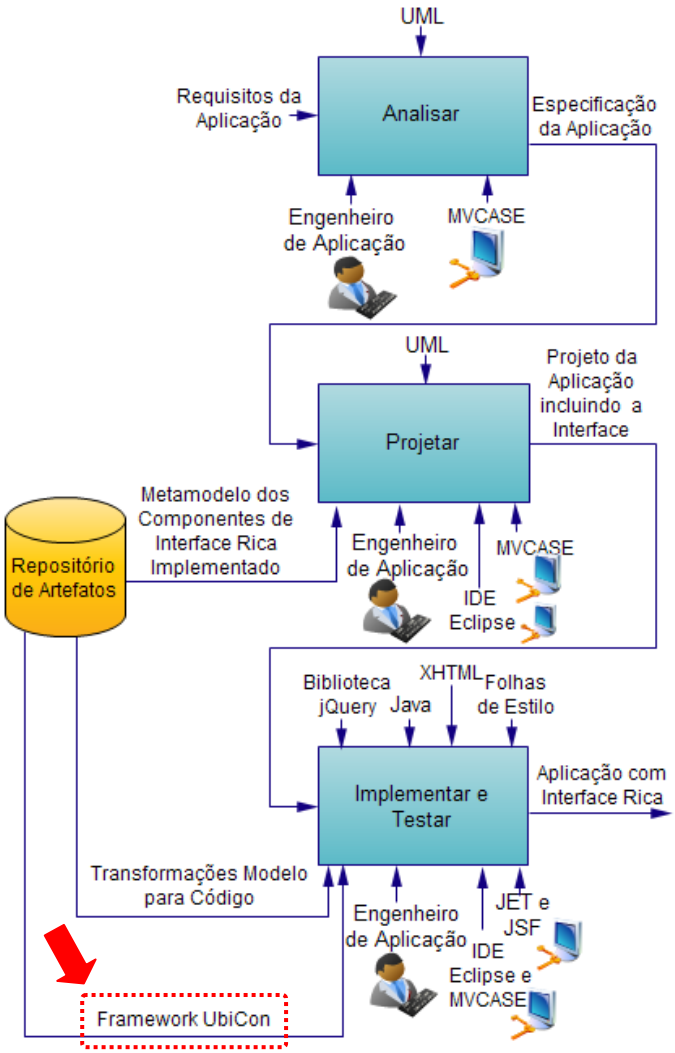


Figura 4.11 – Engenharia de Aplicação no *Model Driven RichUbi* com reúso do *UbiCon*.

adaptadores de conteúdo. Além disso, o *framework* pode ser futuramente estendido com a incorporação de novas entidades contextuais, de modo a fornecer serviços de adaptação que atendam os diferentes dispositivos, preferências do usuário, características da rede e outros contextos.

4.4 Considerações Finais

Para que a adaptação de uma aplicação que executa em um ambiente ubíquo ocorra de forma satisfatória, diferentes fatores devem ser considerados pela aplicação. Esses fatores incluem não apenas as características do dispositivo de acesso, mas também características relacionadas com outros perfis, como o do usuário, da rede de acesso e outros.

Visando a prover auxílio aos desenvolvedores de software no desenvolvimento de interfaces ricas adaptativas de aplicações ubíquas, foi construído o *framework UbiCon*, apresentado neste capítulo, reaproveitando-se os adaptadores de conteúdo elaborados na etapa de ED do processo *Model Driven RichUbi*. O *UbiCon* encapsula as tarefas de manipulação do contexto na execução da aplicação e fornece funcionalidades para adaptação do conteúdo de interfaces ricas.

Como um refinamento do processo, o *framework UbiCon* fornece um apoio computacional mais adequado para o emprego dos adaptadores de conteúdo, através de uma interface que omite detalhes de implementação e torna disponível, para o Engenheiro da Aplicação, apenas o que interessa para a sua reutilização. Embora a versão atual trate apenas do contexto associado ao perfil do dispositivo de acesso, o *UbiCon* foi projetado de forma que novos componentes que manipulam o contexto de outras entidades contextuais possam ser futuramente adicionados. Assim, é possível estender o *UbiCon* para atender a diferentes requisitos de adaptação demandados por diferentes entidades contextuais. Dessa forma, o *UbiCon* pode ser reutilizado na etapa de EA do processo *Model Driven RichUbi* para apoiar a construção de interfaces ricas que se adaptam conforme as variações contextuais de diferentes entidades.

No próximo capítulo é abordada a avaliação do processo *Model Driven RichUbi*. Nessa avaliação o reúso do *framework UbiCon* foi fundamental para facilitar o processo no atendimento aos requisitos de adaptação dos conteúdos das interfaces.

Capítulo 5

AVALIAÇÃO

5.1 Considerações Iniciais

Avaliação é um processo altamente crítico para a melhoria da qualidade do processo de produção de software (ARES et al., 1998). Através da avaliação é que se pode verificar a efetividade de novas ferramentas, métodos, processos e estratégias de desenvolvimento de software antes de colocá-los em prática. Se nenhuma avaliação é realizada, a introdução de uma nova tecnologia no desenvolvimento de software pode ser ineficiente e até mesmo ter impactos negativos na qualidade do software desenvolvido. A avaliação na Engenharia de Software, portanto, fornece as bases para validar as teorias a respeito de uma tecnologia e confrontá-las com a realidade, com o intuito de descobrir se irão, de fato, surtir o efeito esperado tanto no produto quanto no processo (TRAVASSOS et al., 2002; WOHLIN et al., 2000).

O desenvolvimento de software é, na maioria das vezes, uma tarefa complexa. Projetos de software podem se estender por longos períodos de tempo, envolver diferentes pessoas com habilidades distintas, e consistir de várias atividades que produzem diversos documentos intermediários para especificação do software a ser entregue. Essa complexidade acarreta em dificuldades para a otimização dos processos de desenvolvimento (WOHLIN et al., 2000). Neste sentido, a avaliação e o aperfeiçoamento dos processos de software têm sido estratégias aplicadas ao longo dos anos em busca da melhoraria da qualidade do software desenvolvido (e.g. *Quality Improvement Paradigm – QIP* (BASILI & GREEN, 1994), *Capability Maturity Model –*

CMM (PAULK et al., 1994)). Em geral, os métodos de avaliação baseiam-se na premissa de que a qualidade do produto de software é determinada pela qualidade de seu processo de desenvolvimento, considerando a máxima de que um bom produto é resultado direto de um bom processo (ARES et al., 1998).

De fato, a literatura sugere que novas abordagens de desenvolvimento de software devem ser avaliadas no sentido de verificar sua efetividade antes de colocá-las em prática (TRAVASSOS et al., 2002; WOHLIN et al., 2000). Assim, neste capítulo é apresentada a validação do processo *Model Driven RichUbi*, elaborado neste trabalho. Para avaliação do processo foram conduzidos dois tipos de experimentos. Primeiro, um estudo de caso preliminar foi desenvolvido para investigar a viabilidade do processo, bem como avaliar o uso do *framework UbiCon* para suporte à abordagem de adaptação híbrida empregada no *Model Driven RichUbi*. O principal objetivo do estudo de caso foi verificar a aplicabilidade das atividades da Engenharia de Aplicação para a construção das interfaces ricas adaptativas, e determinar quão bom é o suporte fornecido pelo *UbiCon* com respeito ao comportamento dos serviços de adaptação fornecidos. Segundo, uma avaliação controlada, com a aplicação da metodologia experimental (WOHLIN et al., 2000), foi conduzida com o intuito de avaliar o impacto do processo proposto na eficiência de equipes que desenvolvem interfaces ricas adaptativas.

Este capítulo está organizado da seguinte forma: a Seção 5.2 apresenta o estudo de caso preliminar, incluindo a discussão acerca dos resultados observados (Seções 5.2.4 e 5.2.5); a Seção 5.3 descreve o desenvolvimento da experimentação do processo, incluindo a análise e interpretação dos resultados obtidos; e na Seção 5.4 estão as considerações finais que encerram este capítulo.

5.2 Estudo de Caso Preliminar

Como forma de avaliar a aplicabilidade do processo proposto, bem como averiguar o comportamento dos serviços de adaptação fornecidos pelo *framework UbiCon*, um estudo de caso inicial foi conduzido desenvolvendo uma aplicação no domínio de Atendimento de Urgência e Emergência. As atividades da etapa de EA do *Model Driven RichUbi* foram executadas para o desenvolvimento das interfaces

ricas adaptativas do módulo Web do *Sistema de Posicionamento Espacial de Ambulâncias* (*Ambulance Space Positioning System – ASPS*) (BELLINI et al., 2010), de forma que o mesmo pudesse ser acessado a partir de *desktops* e *smartphones*. O ASPS surgiu de um estudo piloto com o objetivo de verificar o uso dos sinais de antenas *Global System for Mobile Communication (GSM)* para a localização de pessoas ou objetos, com foco na localização de veículos de atendimento emergência. Com essa funcionalidade, o *ASPS* permite que uma equipe de gerenciamento de frotas monitore a mobilidade das ambulâncias e direcione uma determinada ambulância para o atendimento de emergência em um local próximo à região em que a mesma encontra-se atualmente.

Sendo de natureza ubíqua, o *ASPS* é constituído por três partes, conforme ilustrado na Figura 5.1. A primeira, executada no terminal GSM presente na ambulância, realiza o cálculo do *posicionamento* por meio da triangulação das antenas de telefonia celular próximas às ambulâncias, e transmite a posição calculada a um servidor via HTTP. A segunda, executada no servidor, realiza o *processamento* dos dados transmitidos pelo terminal GSM e os armazena em uma base de dados espacial. A terceira parte trata-se do módulo Web, o qual inclui a página administrativa do ASPS. Esse módulo possui uma interface gráfica para a *visualização* da posição das ambulâncias no mapa da cidade. Em pequenos intervalos de tempo são efetuadas requisições assíncronas ao servidor, através de AJAX, para recuperar e

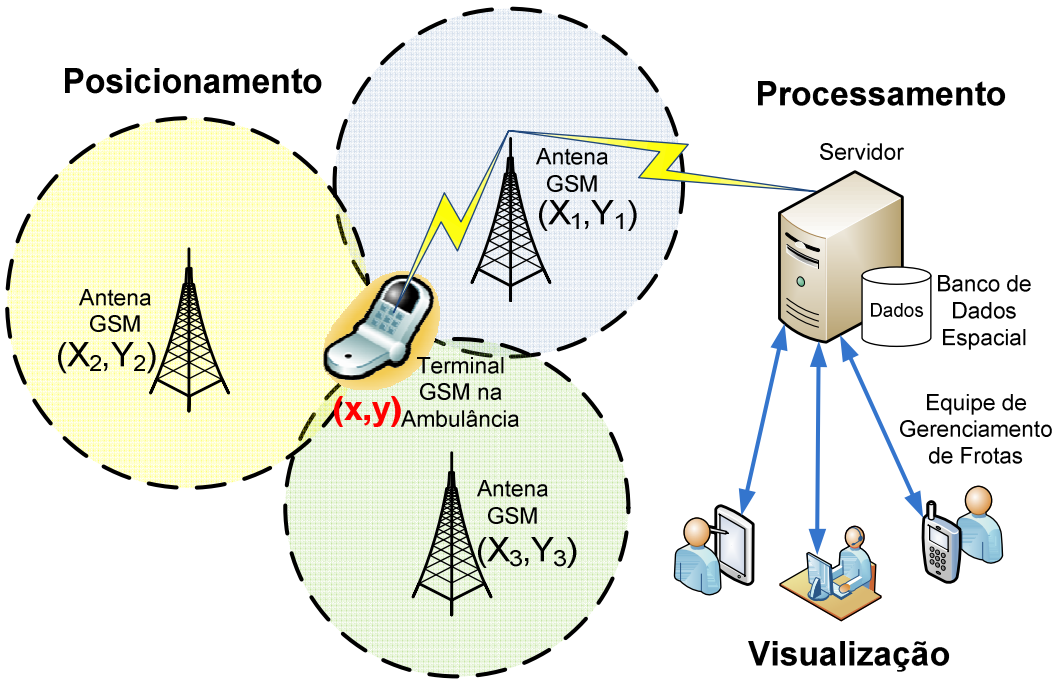


Figura 5.1 – Arquitetura do *ASPS* (adaptado de BELLINI et al., 2010).

atualizar no mapa a posição mais recente das ambulâncias. A interface também dispõe de recursos que possibilitam ao gerenciador da frota atualizar o *status* de uma determinada ambulância (e.g. disponível, em atendimento de emergência, em manutenção) para indicar se a mesma está disponível para atendimento.

O módulo Web do ASPS foi desenvolvido usando o *framework JSF*. Duas versões genéricas da interface foram construídas em tempo de desenvolvimento, sendo uma adequada para visualização em *smartphones* e outra para *desktops*. Durante a execução da etapa de EA, o metamodelo dos componentes de interface rica e as transformações M2C, desenvolvidos na etapa de ED do processo *Model Driven RichUbi*, foram empregados para modelagem das interfaces e geração parcial do código. Além disso, o *framework UbiCon*, derivado dos adaptadores de conteúdo originalmente produzidos na ED, foi reutilizado para prover adaptação às interfaces de forma híbrida.

A seguir, apresenta-se as tarefas realizadas em cada uma das atividades da Engenharia de Aplicação, instanciadas para a construção das interfaces do módulo Web do *ASPS*.

5.2.1 Analisar

A EA iniciou-se com a análise, onde o *ASPS* foi especificado a partir de seus requisitos. Inicialmente, a especificação dos requisitos foi realizada através de diagramas de casos de uso e de classes. A Figura 5.2(a) mostra, por exemplo, o diagrama de casos de uso construído com auxílio da ferramenta MVCASE para especificar alguns dos requisitos identificados para o *ASPS*, como os relacionados à autenticação do gerenciador da frota (*fleet manager*), à visualização das posições das ambulâncias e à edição do *status* das mesmas. A Figura 5.2(b) mostra o diagrama de classes que especifica as entidades “Pessoa” (*Person)*, “Gerenciador da Frota” (*FleetManager)*, “Ambulância” (*Ambulance)*, e “Centro de Atendimento de Emergência” (*Public Safety Answering Point – PSAP)* que fazem parte do domínio de aplicação do *ASPS*.

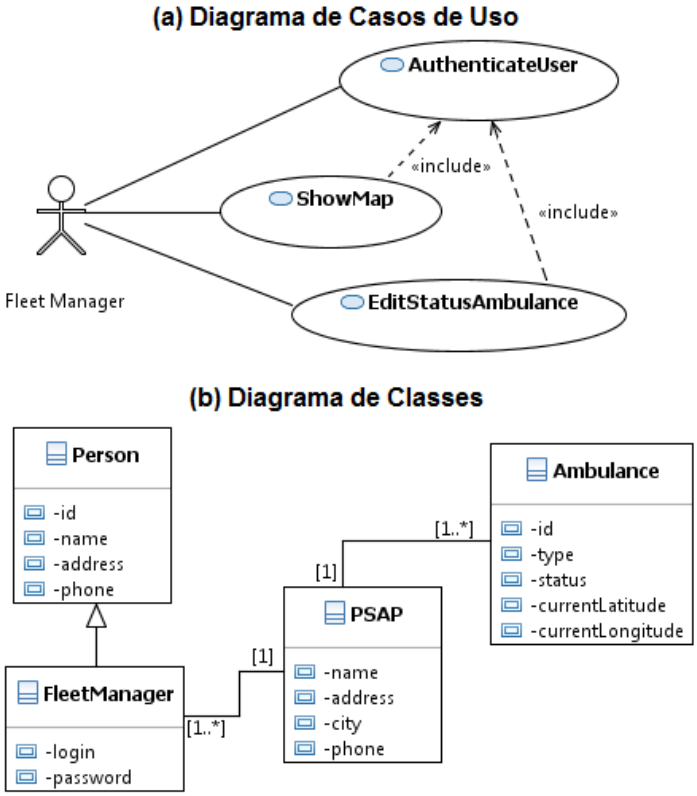


Figura 5.2 – Especificação dos requisitos do *ASPS*.

5.2.2 Projetar

Após a análise, realizou-se o projeto, onde foram tomadas as decisões de adoção das tecnologias e plataformas que viabilizam a implementação da aplicação. No caso do módulo Web do ASPS, decidiu-se adotar a plataforma Java EE e o *framework JSF* para o desenvolvimento das interfaces. Além disso, o metamodelo foi instanciado para a modelagem das interfaces ricas.

A modelagem foi baseada nos casos de uso identificados durante a análise da aplicação, de forma que cada caso de uso fosse traduzido para componentes de interface que os satisfizessem. A Figura 5.3 mostra, à esquerda, parte do modelo elaborado para as interfaces do módulo Web do *ASPS* no *plugin* editor de modelos integrado à ferramenta MVCASE. À direita têm-se os diagramas de objetos ilustrando a instanciação do metamodelo dos componentes de interface rica. Os casos de uso “AuthenticateUser” e “ShowMap” apresentados na Figura 5.2(a) foram mapeados, respectivamente, para um formulário de entrada de dados numa página de autenticação, e para uma região de conteúdo (*div*) que contém o mapa em um painel de abas na página administrativa do *ASPS*.

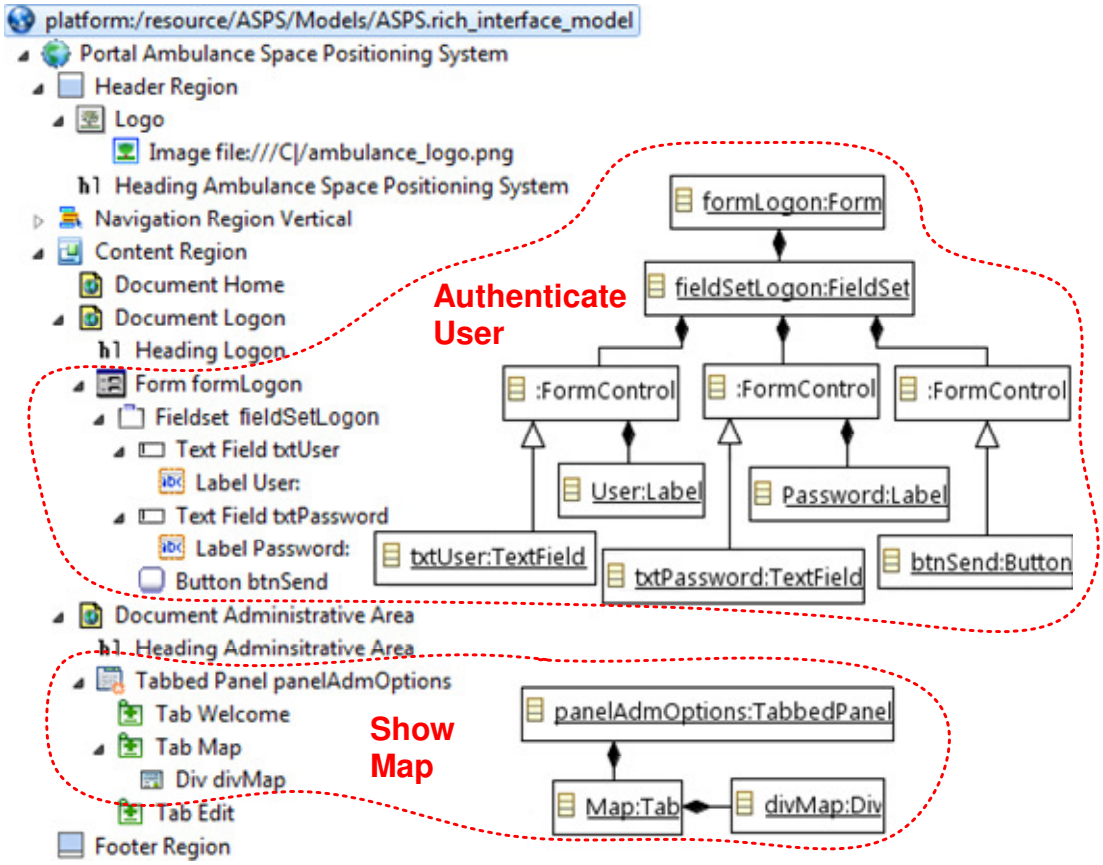


Figura 5.3 – Modelo das interfaces do *ASPS*.

5.2.3 Implementar e Testar

Tendo realizado o projeto do *ASPS*, seu módulo Web foi implementado e testado. Nesta fase, utilizando a MVCASE, foram executadas as transformações M2C sobre o modelo das interfaces ricas para a geração parcial do código das versões para *desktop* e *smartphones*. A Figura 5.4 mostra a página administrativa do ASPS construída automaticamente a partir do modelo apresentado na Figura 5.3 aplicando-se a transformação M2C de geração de código XHTML para *desktops*. É possível notar que a transformação aplicada já inclui no código gerado alguns estilos que conferem à interface uma pré-formatação de seu leiaute. Além disso, componentes de interface rica, como o painel de abas, são renderizados graças ao uso das funções da biblioteca *jQuery* no código produzido. Após isso, o código parcial foi complementado com a incorporação do *mashup* do *Google Maps*[36] na página administrativa, a inclusão das funções *JavaScript* que recuperam os dados assincronamente para atualização do mapa, e a implementação de folhas de estilo personalizadas.

[36] http://code.google.com/intl/en/apis/maps/index.html.

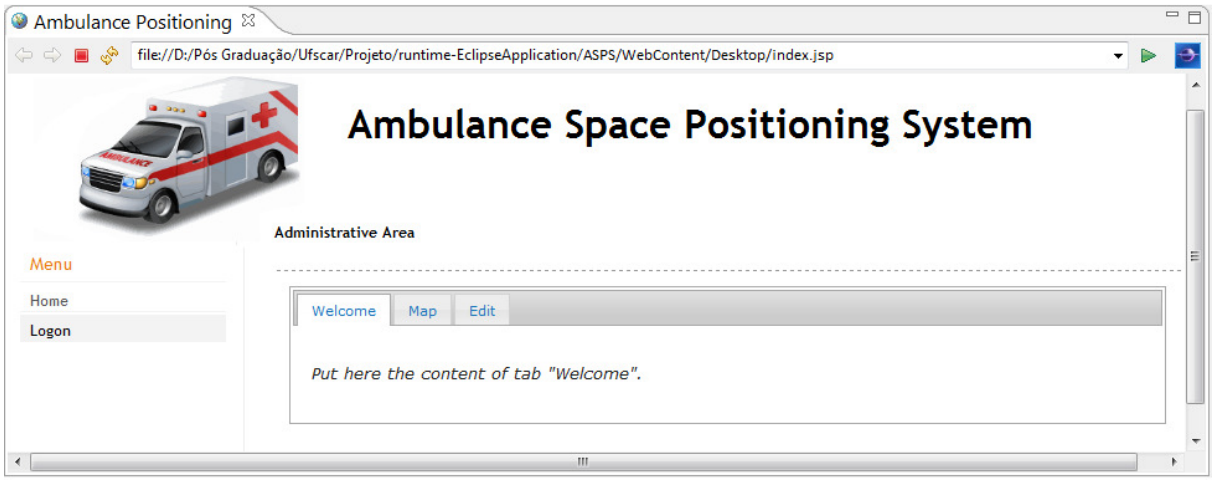


Figura 5.4 – Interface parcial da página administrativa do *ASPS* para *desktops* gerada pelas transformações.

O *framework UbiCon* também foi reutilizado para conferir às interfaces uma característica adaptativa. A Figura 5.5 mostra um trecho do arquivo de configuração do *ASPS*, chamado *web.xml*, onde o reúso do *UbiCon* é definido. Nesse arquivo, os nós *filter* e *filter-mapping* são utilizados, respectivamente, para indicar o reúso do componente `ContentAdapterFilter` na aplicação, e para mapear o disparo desse *Filtro* de acordo com determinado tipo de requisição e padrão de URL. Conforme observa-se no trecho destacado da Figura 5.5, no *ASPS* o `ContentAdapterFilter` foi mapeado para chamadas a recursos Web com qualquer padrão de nomenclatura na URL (`/*`) originadas a partir de requisições vindas diretamente do cliente (`REQUEST`), ou feitas como resultado de um encaminhamento (`FORWARD`) ou de uma ação de inclusão (`INCLUDE`), ou ainda redirecionadas pelo mecanismo de tratamento de erros do servidor (`ERROR`). Dessa forma, quaisquer requisições feitas ao módulo Web do *ASPS* serão interceptadas pelo `ContentAdapterFilter` para o processamento da adaptação.

Preparou-se também o arquivo *context-rules.xml* para definir as regras de escolha de cada versão estática da interface pelo *framework UbiCon* durante o processamento da adaptação híbrida. Nesse arquivo, determinou-se que a versão para *desktops* do módulo Web do ASPS será selecionada apenas se o EC `DEVICE_PRODUCT_INFO_IS_MOBILE_DEVICE`, recuperado do contexto, possuir o valor "falso" (`false`), bem como se o dispositivo fornece suporte a XHTML, i.e., o EC `DEVICE_MARKUP_XHTML_SUPPORT` deve possuir valor verdadeiro (`true`). Caso contrário, a versão para *smartphones* deverá ser entregue ao dispositivo do usuário. O arquivo *context-rules.xml* do *ASPS* foi ilustrado no capítulo anterior, na Figura 4.9.

web-app	(((description*, display-name*, icon*)) | distributable | context-param | filter
xmlns:xsi	http://www.w3.org/2001/XMLSchema-instance
xmlns	http://java.sun.com/xml/ns/javaee
xmlns:web	http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd
xsi:schemaLocation	http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web
id	WebApp_ID
version	2.5
display-name	ASPS
⋮	
filter	(((description*, display-name*, icon*)), filter-name, filter-class, init-param*)
display-name	ContentAdapterFilter
filter-name	ContentAdapterFilter
filter-class	br.ufscar.dc.ubicon.contentadaptation.ContentAdapterFilter
filter-mapping	(filter-name, (url-pattern | servlet-name)+, dispatcher*)
filter-name	ContentAdapterFilter
url-pattern	/*
dispatcher	REQUEST
dispatcher	INCLUDE
dispatcher	FORWARD
dispatcher	ERROR

Figura 5.5 – Reúso do *UbiCon* no *ASPS*.

Para testar a execução das interfaces foram utilizados os navegadores Web dos emuladores dos *smartphones HTC G1* e *iPhone*, bem como o navegador Web de um computador pessoal. Para fins de testes, os dados de posição das ambulâncias para visualização no mapa foram obtidos através da simulação da triangulação das antenas de telefonia celular usando o emulador de terminal GSM disponível com o *Software Development Kit (SDK)* do *Android*[37]. As intensidades de sinais das antenas para o cálculo dos dados de posição também foram obtidas através de simulação (BELLINI et al., 2010). A Figura 5.6(a) e (b) mostram a execução do módulo Web do *ASPS*, com e sem a parte da adaptação dinâmica, nos *smartphones HTC G1* e *iPhone*, respectivamente. Na Figura 5.6(a) é mostrado o topo da interface que contém o logotipo do *ASPS*. É possível observar que, com a adaptação da interface, o logotipo foi redimensionado para encaixar-se à tela do *HTC G1*. A Figura 5.6(b) mostra a parte inferior da mesma interface, onde se tem o painel de abas com o *mashup* do *Google Maps*. É possível notar que, através da adaptação da interface, o painel de abas foi ajustado para encaixar-se à largura de tela do *iPhone*. A Figura 5.6(c) mostra a interface visualizada num computador pessoal, para o qual foi entregue a versão construída para *desktops*.

[37] http://developer.android.com/sdk/index.html.

Figura 5.6 – Teste da execução das interfaces do *ASPS* em dispositivos distintos.

5.2.4 Discussão – Aplicabilidade do Processo

Com relação à aplicabilidade do processo, observou-se que as atividades e os artefatos de suporte prescritos na etapa de EA facilitaram o desenvolvimento das interfaces ricas adaptativas. Uma vez que na modelagem foram usados os termos e conceitos do Domínio de Interfaces Ricas, o que foi viabilizado graças ao uso de uma DSL expressa através de um metamodelo desse domínio, houve uma simplificação na tradução dos requisitos da aplicação em componentes de interface

que satisfizessem os requisitos identificados. Por não estarem atrelados a detalhes técnicos de implementação, os modelos tornaram-se mais legíveis e de fácil compreensão. Mesmo não tendo ocorrido a participação dos possíveis usuários do *ASPS* durante a implementação das interfaces de seu módulo Web, é importante ressaltar que com modelos mais intuitivos é possível aos usuários compreender melhor como os requisitos são mapeados em interfaces e propor mudanças diretamente nos modelos antes de partir para a implementação.

Outro ponto importante observado foi a independência de plataforma obtida através dos modelos das interfaces. Foi possível gerar código para duas versões distintas da interface a partir dos mesmos modelos apenas trocando-se as transformações M2C. Apesar de ainda existirem desafios a serem superados com relação à geração automatizada de código, como o mapeamento entre o código gerado e seu respectivo modelo para evitar que as partes customizadas sejam sobrescritas nas manutenções, essa abordagem torna-se bastante interessante na Computação Ubíqua onde o universo de dispositivos é grande. Neste sentido, não há necessidade de se construir as diferentes versões das interfaces a partir do zero, sendo as tarefas repetitivas de codificação implementadas nas transformações.

5.2.5 Discussão – Comportamento da Adaptação Híbrida

Com relação ao comportamento da adaptação híbrida fornecida pelo *framework UbiCon*, observou-se que a adaptação das interfaces do módulo Web do *ASPS* em três dispositivos diferentes mostrou-se satisfatória, atendendo às peculiaridades de cada dispositivo. Conforme observa-se na Figura 5.6, as interfaces puderam ser visualizadas em dispositivos distintos com os conteúdos adaptados em conformidade com as capacidades gráficas de cada um.

Com o intuito de avaliar o comportamento em relação à capacidade de carga do *framework UbiCon*, foi preparado um ambiente de testes envolvendo dois computadores. Conforme ilustra a Figura 5.7, o computador (1) foi configurado com Windows 7 (2 GHz/2 GB) e o computador (2) com o sistema operacional Linux CentOS (2.8 GHz/ 2 GB). O *Apache JMeter*[38] foi instalado no computador (1) para simular acessos simultâneos de diferentes usuários. O módulo Web do *ASPS*, que

[38] http://jakarta.apache.org/jmeter/.

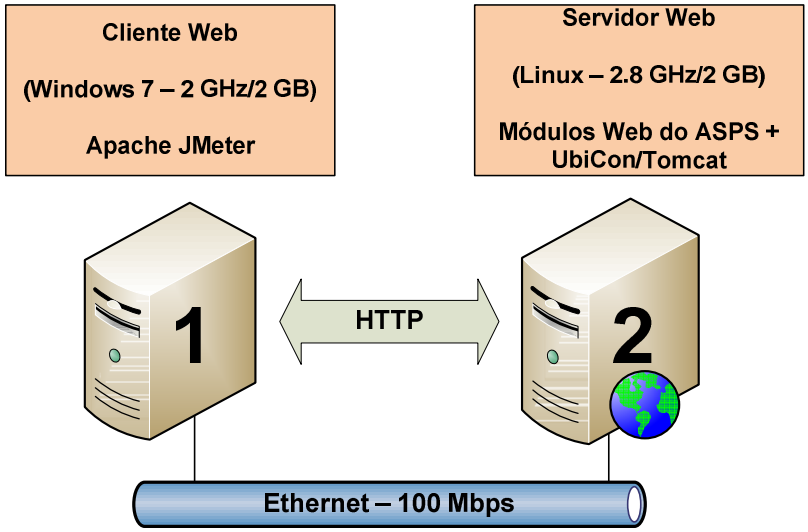


Figura 5.7 – Ambiente de testes para avaliar a capacidade de carga do *framework UbiCon*.

reutiliza o *framework UbiCon*, foi implantado no servidor *Apache Tomcat 6* instalado no computador (2). Com o propósito de comparar a capacidade de carga em relação à estratégia de adaptação puramente estática, outro módulo Web do ASPS, que aplica a adaptação estática, também foi instalado no computador (2). O teste consistiu em acessar repetidamente o *link* da página administrativa do ASPS por um período de cinco minutos. Para acessar esse *link*, *desktops* e dispositivos móveis foram distribuídos randomicamente entre os usuários virtuais configurados no *JMeter*. Dois cenários de testes foram definidos. No primeiro cenário, os acessos foram feitos ao módulo que implementa a adaptação híbrida com o *UbiCon*. No segundo, os acessos foram feitos ao módulo que utiliza adaptação estática. Em ambos os casos, uma carga de acessos progressivamente crescente foi aplicada até o limite de 50 usuários virtuais simultâneos, perfazendo um total de 1275 requisições em cada cenário. Para simulação de dispositivos diferentes, amostras de requisições HTTP contendo diferentes valores para o campo *User-Agent* foram configuradas para cada usuário virtual no *JMeter*.

Os resultados dos testes da capacidade de carga são apresentados na Figura 5.8. Uma média geral de tempo de resposta de 1109 ms foi obtida usando a adaptação híbrida realizada pelo *framework UbiCon*, com as versões estáticas das interfaces sendo refinadas dinamicamente conforme o perfil do dispositivo. Já com a adaptação puramente estática, uma média geral de tempo de resposta de resposta de 55 ms foi alcançada, neste caso apenas com o redirecionamento para as páginas estáticas mais apropriadas ao dispositivo atual, porém sem qualquer refinamento dinâmico.

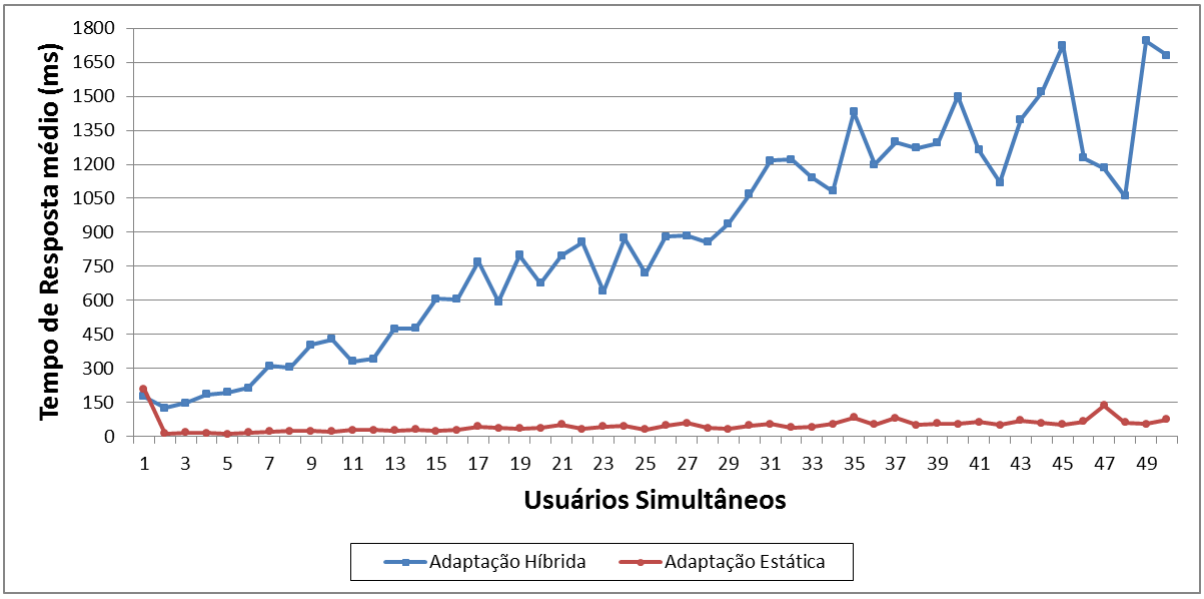


Figura 5.8 – Resultados do teste de capacidade de carga do UbiCon.

Sob uma perspectiva quantitativa, o aumento no tempo médio de resposta observado na Figura 5.8 com o uso do *framework UbiCon* pode ser justificado pela maior demanda de processamento da parte dinâmica da adaptação em face do crescimento do número usuários concorrentes. Em alguns casos, mesmo com número maior de usuários, o tempo médio de resposta observado foi menor do que aquele obtido com menos usuários. Essas oscilações se devem provavelmente às condições de tráfego da rede, e, sobretudo, ao número variado de acessos a partir de dispositivos móveis, onde a frequência de adaptação dinâmica foi mais intensa em face da heterogeneidade inerente a esse grupo de dispositivos. Conforme mostra a Figura 5.9, para a adaptação híbrida os tempos médios de resposta das requisições feitas a partir de dispositivos móveis foi, em geral, maior do que para requisições a partir de *desktops*. O maior impacto no tempo de resposta para os dispositivos móveis evidencia uma maior carga de processamento da adaptação dinâmica das interfaces para esses dispositivos.

Por outro lado, de um ponto de vista qualitativo, deve-se levar em consideração a redução de complexidade no desenvolvimento e a melhoria da precisão no ajuste das interfaces resultante da adaptação híbrida realizada pelo *UbiCon*, o qual facilita o trabalho de autoria de interfaces ricas adaptativas, em comparação com a adaptação puramente estática. Além disso, a adaptação híbrida também permite atender a uma gama maior de dispositivos. Conforme ilustrado na Figura 5.6, com apenas uma versão genérica da interface para dispositivos móveis a aplicação pôde ser acessada a partir de dois *smartphones* diferentes sem qualquer

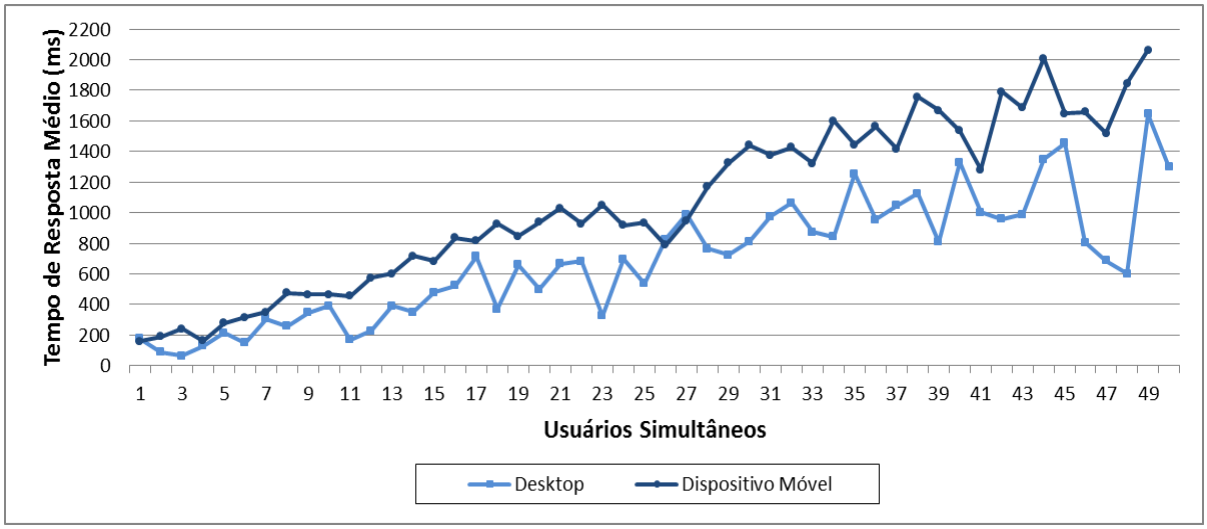


Figura 5.9 – Tempo médio de resposta da adaptação híbrida por grupo de dispositivos.

distorção. Outros *smartphones* poderiam acessar a aplicação e obteriam o mesmo efeito. Por outro lado, para atender a um conjunto maior de dispositivos na adaptação estática, a construção de versões altamente especializadas é necessária. Essa tarefa é de difícil gerenciamento em vista da infinidade de dispositivos existentes e das restrições de tempo e orçamento a que os projetos de software estão limitados.

5.3 Experimentação do Processo Model Driven RichUbi

Na Engenharia de Software a experimentação é importante para testar hipóteses sobre uma nova tecnologia, observando se os efeitos da sua adoção estão alinhados com o que foi declarado nas hipóteses (WOHLIN et al., 2000). A experimentação, portanto, verifica a previsão teórica de encontro à realidade, de forma a prover evidências de que o objeto avaliado (e.g. processo, método, ferramenta, abordagem de desenvolvimento, recurso, modelo, teoria) está ou não indo na direção esperada (TRAVASSOS et al., 2002).

Na próxima seção faz-se uma breve introdução sobre os princípios experimentais aplicados à Engenharia de Software, nos quais se baseou esta parte da avaliação. Em seguida, apresenta-se o experimento conduzido que avaliou o impacto na eficiência de equipes que desenvolveram interfaces ricas adaptativas utilizando o processo *Model Driven RichUbi* em comparação a um processo, não dirigido a modelos, baseado no *ciclo de vida clássico do software* (PRESSMAN, 2004).

5.3.1 Experimentação em Engenharia de Software

A Engenharia de Software é descrita formalmente pelo IEEE como “a aplicação de uma abordagem sistemática, disciplinada e quantificável para o desenvolvimento, operação e manutenção de software” (IEEE, 1990). Tratando-se de uma abordagem de engenharia, dentre outros aspectos, a Engenharia de Software implica num processo que seja previsível e disciplinado para o desenvolvimento de produtos de software confiáveis de forma controlada e eficiente. Esse mesmo rigor com que o software é produzido certamente deverá ser aplicado quando da avaliação e aperfeiçoamento da construção de software (WOHLIN et al., 2000).

Neste sentido, a experimentação oferece um modo sistemático, disciplinado, quantificável e controlado para avaliação da atividade humana envolvida num processo de criação de software. Em Engenharia de Software os experimentos normalmente são realizados em laboratórios, sob condições controladas. O objetivo é manipular uma ou mais variáveis relacionadas com o objeto sendo estudado enquanto mantêm-se outras variáveis em um nível fixo. O efeito dessa manipulação observado nos resultados do experimento é medido, e, com base nisso, análises são desempenhadas para validar ou refutar as hipóteses formuladas (TRAVASSOS et al., 2002; WOHLIN et al., 2000).

5.3.1.1 Princípios da Experimentação

Os experimentos são apropriados para validar teorias, confirmar o conhecimento convencional, explorar relacionamentos, avaliar a predição de modelos ou validar medidas associados ao desenvolvimento de software. (TRAVASSOS et al., 2002). Para que um experimento seja realizado de forma adequada, o mesmo deve ser preparado, conduzido e analisado apropriadamente. Uma das principais vantagens da experimentação é o nível de controle obtido com relação aos participantes do experimento, objetos, instrumentação, dentre outros. Outras vantagens incluem a possibilidade de desempenhar análises estatísticas para teste das hipóteses. Os principais elementos que compõem um experimento são ilustrados na Figura 5.10.

Conforme mostra a Figura 5.10, na abordagem experimental assume-se a existência de uma relação de **causa** e **efeito**, i.e., acredita-se que determinado fenômeno relacionado com o objeto de estudo possua uma causa provável. Tem-se,

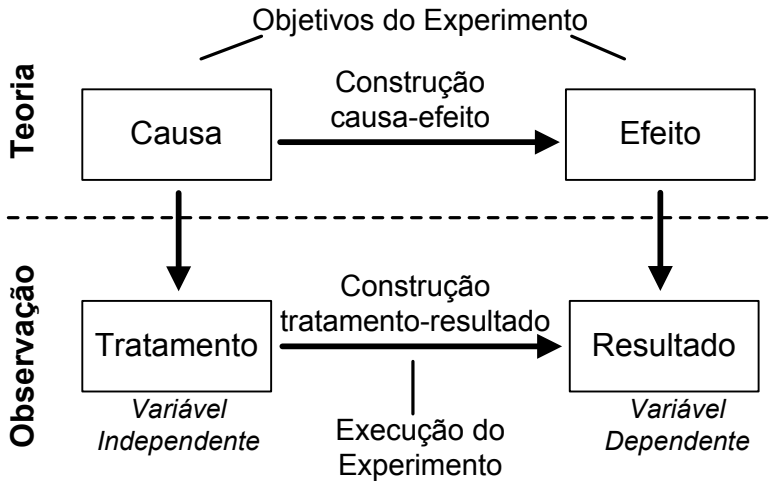


Figura 5.10 – Elementos de um experimento (adaptado de WOHLIN et al., 2000).

então, uma **teoria**. Geralmente um experimento é formulado através de **hipóteses**. Nas hipóteses são formalizadas as ideias dos prováveis relacionamentos de causa e efeito que se quer investigar.

O experimento, então, é criado para testar a teoria ou as hipóteses formuladas através da **observação** dos resultados obtidos. Se o experimento for preparado apropriadamente, então ao final da experimentação deve-se ser capaz de tirar conclusões a respeito do relacionamento de causa e efeito para o qual as hipóteses foram formuladas. A análise e interpretação dos resultados obtidos permite cumprir com os **objetivos do experimento**, os quais podem estar associados tanto com *caracterização* ("o que está acontecendo?"), *avaliação* ("quão bom é isto?"), *previsão* ("é possível levantar estimativas?"), *controle* ("é possível manipular o fenômeno?") ou *melhoria* ("é possível melhorar o fenômeno?") do objeto estudado (TRAVASSOS et al., 2002).

A hipótese principal de um experimento se chama **hipótese nula** (denotada por H_0) e declara que não há um relacionamento estatisticamente significante entre a causa e o efeito; a única justificativa para o fenômeno observado seria apenas casualidade ou coincidência. O objetivo principal do experimento é rejeitar a hipótese nula em favor de uma ou mais **hipóteses alternativas** (denotadas por H_a, H_1, H_2, etc). A decisão de rejeição da hipótese nula deve ser balizada pela verificação dos resultados obtidos no experimento.

De maneira geral, quando se conduz um experimento formal, deseja-se estudar o resultado quando se altera algumas variáveis de entrada para o processo experimental. Conforme ilustra a Figura 5.10, existem dois tipos de variáveis em um experimento: **dependente** e **independente**. As variáveis independentes (ou

variáveis de entrada) são todas aquelas que são manipuladas e controladas durante o experimento. As variáveis dependentes (ou variáveis de resposta) são aquelas que estão sob análise. Deve-se observar suas variações com base nas mudanças feitas nas variáveis independentes. Geralmente, existe apenas uma variável dependente em um experimento. Por exemplo, num estudo para avaliar o efeito de um novo método de desenvolvimento alternativo ao método orientado a objetos na produtividade dos desenvolvedores de uma companhia, a variável dependente seria a *produtividade*. Já as variáveis independentes poderiam ser, por exemplo, o próprio *método de desenvolvimento*, a *experiência da equipe*, as *ferramentas de suporte* e o *ambiente de desenvolvimento*.

As variáveis independentes que são manipuladas durante um experimento são chamadas **fatores** e apresentam a causa que afeta o resultado do processo de experimentação. O valor atribuído a um fator recebe o nome de **tratamento**. As outras variáveis independentes, que não recebem tratamentos, devem ser fixadas em um determinado valor durante o experimento, caso contrário não será possível determinar que fator causa o efeito. No caso do exemplo de estudo anterior, o fator seria o *método de desenvolvimento*, uma vez que se quer estudar o efeito na produtividade com a mudança de método. Dois tratamentos podem ser usados para esse fator: o *método orientado a objetos* e o *método orientado a aspectos*. As demais variáveis independentes (e.g. *ferramentas*, *ambiente de desenvolvimento*) são mantidas constantes.

Os tratamentos são aplicados sobre uma combinação de **objetos** e **participantes** (ou **sujeitos**). Um objeto pode ser, por exemplo, uma *aplicação* que deverá ser construída com diferentes métodos de desenvolvimento. As pessoas que foram especialmente selecionadas da população de interesse para conduzir o experimento são os participantes. As características tanto dos objetos quanto dos participantes podem ser consideradas como variáveis independentes no experimento. Por exemplo, no estudo de exemplo citado anteriormente, os *programas* a serem desenvolvidos com o uso dos dois métodos de desenvolvimento são os objetos, e os participantes são os integrantes da equipe.

A combinação de participantes, objetos e tratamentos é chamada **de teste experimental** ou ***trial***. Por exemplo, um teste experimental para o exemplo anterior poderia ser um desenvolvedor *X* (participante) usando o método de desenvolvimento *orientado a aspectos* (tratamento) para desenvolver o programa *Y* (objeto).

5.3.1.2 Fases de um Experimento

O processo da execução de um experimento presume a realização de diferentes atividades. Em Engenharia de Software, quatro fases principais são propostas na literatura para a realização de um experimento (WOHLIN et al., 2000):

1. **Definição:** nesta fase o experimento é definido em termos dos problemas e objetivos. Os objetivos, o objeto do estudo, o foco da qualidade, o ponto de vista, e o contexto de execução do experimento são descritos na definição. Assim, a fase de definição estabelece a direção geral do experimento, o seu escopo, a base para formulação das hipóteses e as direções iniciais para avaliação da validade do estudo.
2. **Planejamento:** esta fase estabelece a fundamentação do experimento. No planejamento realiza-se a definição do contexto (e.g. ambiente universitário ou industrial), a formulação das hipóteses (incluindo todas as hipóteses nulas e alternativas), a seleção das variáveis (incluindo os tratamentos que cada qual assumirá), a seleção dos participantes (e.g. alunos, professores, profissionais), o projeto do experimento, a preparação conceitual da instrumentação (e.g. formulários de apoio e de coleta de dados, diretrizes para execução das atividades experimentais), e a consideração da validade do experimento (identificação das ameaças à validade). Ao final desta fase, o experimento deverá estar totalmente elaborado e pronto para execução.
3. **Execução**: Esta fase subdivide-se em três passos: *preparação*, *execução* e *validação dos dados coletados*. No passo de preparação, os participantes do experimento são preparados e o material necessário que compõe a instrumentação é elaborado. Na preparação, os participantes devem ser informados sobre a intenção do estudo, e devem dar consentimento e comprometerem-se em participar da atividade experimental. A preparação dos participantes para a experimentação, do ponto de vista moral e metodológico, visa a evitar resultados errôneos devido a mal-entendidos e desinteresse. No passo de execução, o experimento deve ser conduzido de acordo com o planejamento e a coleta dos dados deve ser realizada de maneira não intrusiva ao resultado do estudo. Finalmente, no passo de validação, deve-se garantir que os dados

coletados estão corretos e fornecem uma visão válida para o experimento.

4. **Análise e Interpretação dos Resultados**: esta fase oferece a possibilidade de caracterização e redução do conjunto de dados para viabilizar a verificação das hipóteses. Os resultados devem ser corretamente interpretados, utilizando métodos estatísticos apropriados para testar as hipóteses. Se as hipóteses puderem ser refutadas ou validadas, então conclusões poderão ser destiladas a partir do experimento.

A literatura propõe, ainda, uma quinta fase relacionada com a **apresentação** e **empacotamento** do estudo. A ideia dessa fase é organizar adequadamente os dados e as descobertas do experimento, de forma que o mesmo possa ser replicado a fim de favorecer o aumento do aprendizado dos conceitos investigados e o refinamento do experimento. O correto empacotamento dos dados experimentais pode servir como base para a criação de bibliotecas de experimentação, o que facilita reutilizar as descobertas em estudos futuros, classificar os dados experimentais e criar relatórios detalhados com os dados confiáveis (TRAVASSOS et al., 2002).

5.3.2 Desenvolvimento da Experimentação

A experimentação do processo *Model Driven RichUbi* seguiu as fases experimentais, conforme proposto por Wohlin et al. (2000), e foi conduzida no segundo semestre de 2010. O experimento consistiu de um estudo comparativo entre o uso do *Model Driven RichUbi* para a construção de interfaces ricas adaptativas e o uso do processo baseado no ciclo de vida clássico, sem o emprego de MDD, para o mesmo fim.

Para a realização do experimento foi utilizada uma aplicação ubíqua para o rastreamento de pessoas, elaborada especialmente para a execução do experimento. Essa aplicação, nomeada *TrackMe*, é constituída por três partes: a primeira, executada no dispositivo do usuário, realiza o registro no *Access Point (AP)* mais próximo; a segunda, executada no servidor, processa os registros dos usuários e os armazena em uma base de dados; e a terceira, também executada no servidor, é um módulo Web que oferece uma interface rica para a visualização das posições dos usuários no mapa da localidade em que os APs estão distribuídos.

A tarefa dos participantes durante o experimento foi desenvolver versões das interfaces do módulo Web do *TrackMe*, de modo que o mesmo pudesse ser

visualizado apropriadamente tanto a partir de *smartphones* quanto de *desktops*. Essas versões já haviam sido anteriormente especificadas, sendo essa especificação entregue aos participantes para a execução do experimento. A Figura 5.11 mostra alguns exemplos de interfaces que deveriam ser desenvolvidas pelos participantes. Estabeleceu-se que o módulo Web deveria ser construído utilizando o *framework* JSF e o IDE Eclipse. Além disso, uma vez que o foco do estudo estava no desenvolvimento da camada de apresentação da aplicação, as funcionalidades correspondentes às regras de negócio do *TrackMe* foram previamente implementadas. Assim, os participantes, divididos em grupos, receberam o projeto da aplicação, parcialmente implementado, contendo as regras de negócio a serem importadas ao seu ambiente de desenvolvimento para a construção das interfaces.

Figura 5.11 – Exemplos de interfaces desenvolvidas no experimento.

Os participantes dos grupos selecionados para aplicar o processo *Model Driven RichUbi* seguiram as atividades da etapa de Engenharia de Aplicação para a construção das interfaces, a partir da atividade de *Projeto*. Para esses grupos foram disponibilizados todos os artefatos de suporte ao processo, produzidos previamente na etapa de ED, sendo eles: o *plugin* editor de modelos para a modelagem das interfaces ricas com base no metamodelo; as transformações M2C para geração de código; e o *framework UbiCon* para adaptação híbrida das interfaces.

Os grupos selecionados para o uso do ciclo de vida clássico realizaram as disciplinas tradicionais de *Projeto*, *Implementação* e *Testes* da Engenharia de Software para o desenvolvimento das interfaces. Para fins de comparação da eficiência entre os grupos, aqueles que seguiram o ciclo de vida clássico empregaram a estratégia de adaptação estática para a construção das interfaces, uma vez que esta pode demandar grandes esforços de desenvolvimento por conta do grande número de versões a serem desenvolvidas. Por questões de simplicidade, estabeleceu-se que os grupos do ciclo de vida clássico deveriam apenas desenvolver duas versões especializadas da interface: uma para *desktops* e outra para *iPhones*.

Durante a execução do experimento, todos os grupos registraram, em um formulário entregue para cada grupo, a hora de início e fim da realização de cada atividade executada, bem como as quantidades de linhas de código geradas automaticamente e codificadas manualmente. Na contabilização das linhas de código, foi considerado apenas o código referente às interfaces, como o código das páginas Web, arquivos de *JavaScript* e folhas de estilo personalizadas. No caso dos grupos do ciclo de vida clássico, os quais não utilizaram transformações para geração de código a partir de modelos, considerou-se como código gerado aquele criado automaticamente pelo ambiente de desenvolvimento, tais como *templates* pré-fabricados de páginas Web, folhas de estilos padrões, e outros.

Cada grupo recebeu um material de apoio apropriado com diretrizes que os orientou nas atividades de construção das interfaces. Tanto para os grupos do processo *Model Driven RichUbi*, quanto para os grupos do ciclo de vida clássico, a atividade de *Projeto* compreendeu a modelagem das interfaces (obrigatória no caso do *Model Driven RichUbi* que se baseia em MDD e DSM), bem como a tomada de decisões de tecnologias, padrões e frameworks adicionais que os grupos julgaram necessários para a construção do módulo Web da aplicação. A *Implementação* envolveu a codificação das interfaces, onde cada grupo implementou o código das

páginas da aplicação seguindo suas próprias convenções de codificação, i.e., não foi estabelecida nenhuma formatação específica para o código que deveria ser gerado pelos grupos. Por fim, a atividade de *Testes* englobou a verificação do comportamento da adaptação das interfaces usando emuladores dos dispositivos conforme casos de testes definidos no material de apoio.

As fases experimentais realizadas no estudo são detalhadas nas subseções seguintes.

5.3.2.1 Definição do Experimento

O objetivo do estudo foi:

- **Analisar** o uso do processo *Model Driven RichUbi* na construção das interfaces ricas adaptativas de uma aplicação ubíqua;
- **Com o propósito** de avaliação;
- **Com respeito à** eficiência em termos de tempo despendido e produtividade;
- **Do ponto de vista de** desenvolvedores de software;
- **No contexto de** estudantes de graduação em Ciência e Engenharia da Computação.

O contexto de um experimento pode ser caracterizado em termos do número de participantes e objetos envolvidos no estudo experimental. No experimento realizado, considerando o envolvimento de diversos participantes e de um objeto (aplicação *TrackMe*), o contexto do experimento classificou-se como **estudo de objeto com vários testes** (WOHLIN et al., 2000). Isto significa que o estudo consistiu de diversos testes experimentais, onde em cada teste houve um grupo de participantes aplicando um determinado processo para construir as interfaces da aplicação proposta pelo experimentador.

5.3.2.2 Planejamento do Experimento

O planejamento do experimento envolveu as seguintes etapas:

a) Seleção do contexto. O experimento ocorreu em ambiente universitário, sendo realizado com estudantes de graduação no Laboratório de Ensino do Departamento de Computação da Universidade Federal de São Carlos (UFSCar), no âmbito da disciplina Tópicos em Informática II.

b) Formulação das Hipóteses. Foram elaboradas três hipóteses para o experimento com relação ao efeito do processo de desenvolvimento no resultado. Para a formulação das hipóteses, foram consideradas as seguintes métricas:

τ – Tempo total gasto pela equipe para o desenvolvimento das interfaces ricas adaptativas;

$Þ$ – Produtividade da equipe em termos de linhas de código produzidas (LOC) por unidade de tempo ($Þ = LOC/\ \tau$);

μ_τ – Tempo médio despendido pelas equipes para o desenvolvimento das interfaces ricas adaptativas; e

$\mu_Þ$ – Produtividade média das equipes no desenvolvimento das interfaces ricas adaptativas.

A hipótese nula e suas correspondentes alternativas são:

Hipótese nula (H_0): Em geral, não há diferença entre equipes utilizando o processo *Model Driven RichUbi* e equipes utilizando o processo baseado no ciclo de vida clássico para a construção de interfaces ricas adaptativas, com respeito à eficiência ($\mathcal{E}$) da equipe.

H_0: $\mathcal{E}_{RichUbi} = \mathcal{E}_{Clássico} \Rightarrow \mu_{\tau RichUbi} = \mu_{\tau Clássico}\ e\ \mu_{Þ RichUbi} = \mu_{Þ Clássico}$

Hipótese alternativa (H_1): Equipes utilizando o processo *Model Driven RichUbi* para a construção de interfaces ricas adaptativas são, em geral, mais eficientes do que equipes utilizando o ciclo de vida clássico.

H_1: $\mathcal{E}_{RichUbi} > \mathcal{E}_{Clássico} \Rightarrow \mu_{\tau RichUbi} < \mu_{\tau Clássico}\ e\ \mu_{Þ RichUbi} > \mu_{Þ Clássico}$

Hipótese alternativa (H_2): Equipes utilizando o ciclo de vida clássico para a construção de interfaces ricas adaptativas são, em geral, mais eficientes do que equipes utilizando o processo *Model Driven RichUbi*.

H_2: $\mathcal{E}_{RichUbi} < \mathcal{E}_{Clássico} \Rightarrow \mu_{\tau RichUbi} > \mu_{\tau Clássico}\ e\ \mu_{Þ RichUbi} < \mu_{Þ Clássico}$

c) Seleção das variáveis. A variável dependente selecionada para o experimento foi a *eficiência da equipe*. As variáveis independentes foram o *processo de desenvolvimento*, a *aplicação desenvolvida*, o *ambiente de desenvolvimento* e as *tecnologias de desenvolvimento*. Uma vez que o objetivo era investigar as consequências do uso do processo na eficiência da equipe, o *processo de desenvolvimento* foi estabelecido como o fator que receberia tratamentos distintos e

cujo efeito deveria ser observado na variável dependente. As demais variáveis independentes foram mantidas constantes durante o estudo, conforme segue:

- Aplicação = *TrackMe*;
- Ambiente de desenvolvimento = IDE Eclipse;
- Tecnologias de desenvolvimento = Java, JSF, XHTML, CSS, *JavaScript* e *jQuery*.

d) Seleção dos participantes. A seleção dos participantes foi feita através de **amostragem não probabilística por conveniência** (WOHLIN et al., 2000), selecionando os indivíduos disponíveis mais próximos para participarem do experimento. Participaram do estudo 31 alunos do 3º e 4º anos de graduação dos cursos de Bacharelado em Ciência da Computação e Engenharia da Computação da UFSCar, matriculados na disciplina Tópicos em Informática II.

e) Projeto do Experimento. O projeto do experimento descreve como os testes experimentais são organizados e executados. O estudo foi planejado **em blocos** (WOHLIN et al., 2000), a fim de assegurar que o efeito do fator *nível de experiência dos participantes* não interferisse nos resultados dos tratamentos do fator *processo de desenvolvimento*. Dessa forma, os alunos foram divididos em 10 grupos (blocos) homogêneos, de modo que cada grupo possuísse, tanto quanto possível, médias semelhantes de nível de experiência. A experiência dos participantes foi avaliada pelo *Formulário de caracterização dos participantes* (Apêndice A) – documento utilizado para capturar a experiência profissional e experiência nos assuntos relacionados ao estudo (e.g. Java, JSF, XHTML, CSS). Esse formulário foi entregue a cada participante semanas antes da execução do experimento, de forma que fosse possível planejar o estudo antecipadamente. O gráfico da Figura 5.12 mostra os níveis de experiência individuais de cada participante, bem como o nível médio de experiência dos grupos (rótulos sobrepostos às barras adjacentes referentes aos participantes de um mesmo grupo), conforme as informações apresentadas pelos alunos em seus respectivos formulários de caracterização. Esses níveis foram obtidos quantificando as ponderações entre o grau de conhecimento (escalas de 4 e 5 pontos) e os tempos de experiência (em meses) em cada assunto reportados pelos participantes nos formulários. A alocação dos participantes nos grupos foi realizada de maneira desbalanceada para refletir equipes com número variado de membros.

f) Tipo do Projeto. O tipo do projeto do experimento foi de **um fator** (*processo de desenvolvimento*) **com dois tratamentos** (*Model Driven RichUbi* e

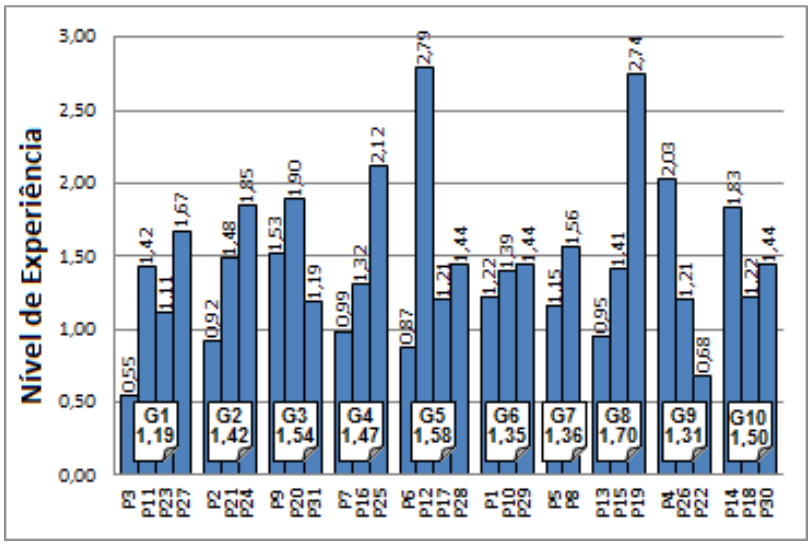


Figura 5.12 – Níveis de experiência individuais dos participantes e experiência média dos grupos.

ciclo de vida clássico) **completamente randomizado** (WOHLIN et al., 2000). Nesse tipo de projeto um mesmo objeto a ser manipulado durante o estudo é utilizado em todos os tratamentos e os participantes (ou grupos) são aleatoriamente atribuídos a cada tratamento. No caso do experimento, a aplicação *TrackMe* foi desenvolvida utilizando ambos os processos, sendo metade dos grupos atribuída randomicamente para utilizar o processo *Model Driven RichUbi* e a outra metade atribuída para utilizar o ciclo de vida clássico. A Tabela 5.1 apresenta o resultado dessa distribuição.

Tabela 5.1 – Distribuição aleatória dos grupos aos tratamentos.

		Tratamento 1: Model Driven RichUbi	Tratamento 2: Ciclo de vida clássico
Grupos	G1	X	
	G2		X
	G3		X
	G4	X	
	G5	X	
	G6		X
	G7		X
	G8		X
	G9	X	
	G10	X	

g) Instrumentação. Os materiais necessários para apoiar os participantes no experimento foram previamente planejados, compreendendo a definição dos *objetos* que seriam manipulados, as *diretrizes* para orientar os grupos na execução dos processos, bem como os *instrumentos para a coleta de dados e medição*.

5.3.2.3 Execução do Experimento

A execução do experimento foi realizada em três passos:

1) Preparação. Nesta fase os materiais definidos na instrumentação do experimento foram efetivamente elaborados, a saber:

- **Objetos.** O módulo Web da aplicação ubíqua *TrackMe* foi projetado e implementado, com exceção de suas interfaces. O projeto parcial contendo as funcionalidades internas da aplicação foi empacotado para ser distribuído aos grupos juntamente com o script *Structured Query Language (SQL)* de criação do banco de dados utilizado, as imagens que deveriam ser utilizadas nas interfaces, e os arquivos CSS e *JavaScript* referentes à biblioteca *jQuery* empregada na construção das interfaces ricas. Aos grupos do ciclo de vida clássico também foi entregue o arquivo XML da base de dados de perfis de dispositivos WURFL para a implementação da adaptação estática. Já para os grupos do *Model Driven RichUbi* foram entregues, ainda, os artefatos de apoio ao processo, que inclui o editor de modelos das interfaces ricas, as transformações M2C e o *framework UbiCon*.
- **Diretrizes.** Os seguintes documentos foram produzidos para serem utilizados no estudo: 1) *Formulário de caracterização dos participantes*, para obtenção da experiência profissional e nos assuntos relacionados diretamente ao estudo; 2) *Formulário de consentimento*, para aprovação e ciência dos participantes dos objetivos do estudo e das condições de participação; 3) *Descrição da tarefa*, com as instruções da sua execução; e 4) *Descrição da aplicação e material de apoio*, contendo o detalhamento da aplicação *TrackMe*, bem como um tutorial para a implementação das interfaces. Os documentos nº 3 e 4 foram preparados em versões especializadas para cada um dos processos. Além disso, um quinto documento foi elaborado para ser utilizado como guia pelos grupos do processo *Model Driven RichUbi*, o qual continha a descrição e o detalhamento das atividades da etapa de EA do processo.
- **Instrumentos para Coleta de Dados.** Medições em experimentos são conduzidas através dos dados que são coletados. Em experimentos envolvendo atividades executadas por seres-humanos, os dados geralmente são coletados manualmente por meio de formulários ou entrevistas. Assim, para a coleta de dados do experimento, foi elaborado o *Formulário de coleta de dados*, no qual os grupos deveriam preencher todas as informações requeridas durante a execução do experimento. Além disso, foram

elaborados também dois tipos de *Formulário de avaliação qualitativa* para cada um dos processos, nos quais cada participante individualmente, após a execução do experimento, deveria reportar sua percepção sobre a facilidade de utilização dos processos para desenvolver interfaces ricas adaptativas.

Todos os documentos elaborados para instrumentação do estudo encontram-se anexados a esta dissertação nos Apêndices A a K.

Para evitar que o tempo de aprendizagem de como utilizar o *Model Driven RichUbi* e o ciclo de vida clássico na construção de interfaces ricas adaptativas interferisse no tempo de desenvolvimento do estudo, foram realizados treinamentos como forma de aquecimento (*warm-up*) com todos os participantes do experimento ao longo das três semanas que antecederam sua realização. Conforme mostra a Tabela 5.2, durante os treinamentos o módulo Web do *ASPS* foi utilizado como objeto para o desenvolvimento das interfaces ricas adaptativas usando ambos os processos. Os treinamentos foram conduzidos de forma que, ao final da fase de treinos, cada participante estivesse suficientemente apto para executar quaisquer dos processos.

Tabela 5.2 – Fases do experimento.

			Processo	
			Ciclo de vida clássico	**Model Driven RichUbi**
Aplicação	*1ª Fase (Treinamentos)*	*ASPS*	Todos os participantes (P1 a P31) – 1ª e 2ª semanas	Todos os participantes (P1 a P31) – 3ª semana
	2ª Fase (Execução)	*TrackMe*	G2, G3, G6, G7, G8 – 4ª semana	G1, G4, G5, G9, G10 – 4ª semana

2) Execução. Os grupos executaram os testes experimentais conforme o projeto experimental. Na Tabela 5.3 são mostrados os dados coletados pelos grupos na execução do experimento. O significado dos acrônimos dos dados podem ser visualizados na Tabela 5.4.

Tabela 5.3 – Dados coletados.

	Grupo	HIAProj	HTAProj	HIAImpl	HTAImpl	HIATeste	HTATeste	LCGAuto	LCGManual	LOC Total	Proble-mas	Tempo Total (τ)	Produti-vidade (Þ)
Model Driven RichUbi	G1	16:37	17:06	17:06	17:25	17:25	18:08	289	64	353	00:43	01:31	233
	G4	16:35	17:17	17:18	17:43	17:43	17:50	200	45	245	00:11	01:15	196
	G5	16:40	17:01	17:01	17:12	17:12	17:34	282	48	330	00:00	00:54	367
	G9	16:37	16:54	16:54	17:19	17:19	17:21	125	40	165	00:02	00:44	225
	G10	16:42	16:50	16:51	17:04	17:04	17:18	252	40	292	00:11	00:36	487
	Média	**00:23**		**00:18**		**00:17**		**230**	**47**	**277**	**00:13**	**01:00**	**301**
Ciclo de vida clássico	G2	16:30	17:15	17:15	18:00	18:00	18:15	0	232	232	00:35	01:45	133
	G3	16:37	16:52	16:52	18:03	18:03	18:11	12	75	87	00:34	01:34	56
	G6	16:40	16:41	16:41	17:40	17:40	18:10	24	149	173	00:30	01:30	115
	G7	16:40	17:33	17:33	18:00	17:53	18:04	26	94	120	00:38	01:24	86
	G8	---	---	16:42	17:27	17:28	18:00	36	132	168	00:32	01:18	129
	Média	**00:22**		**00:49**		**00:19**		**20**	**136**	**156**	**00:33**	**01:30**	**104**

Tabela 5.4 – Legenda dos dados coletados.

Descrição do Dado	Acrônimo
Hora de Início da Atividade de Projeto	HIAProj
Hora de Término da Atividade de Projeto	HTAProj
Hora de Início da Atividade de Implementação	HIAImpl
Hora de Término da Atividade de Implementação	HTAImpl
Hora de Início da Atividade de Testes	HIATeste
Hora de Término da Atividade de Testes	HTATeste
Linhas de Código Geradas Automaticamente	LCGAuto
Linhas de Código Geradas Manualmente	LCGManual

Os dados mostrados na Tabela 5.3 estão organizados em dois blocos referentes aos processos usados como tratamento. Na parte superior da tabela são listados os dados dos grupos que executaram o processo *Model Driven RichUbi*, e na parte inferior têm-se os dados dos grupos que executaram o ciclo de vida clássico. As colunas "LOC Total", "Tempo Total (τ)" e "Produtividade ($Þ$)" representam, respectivamente, a quantidade total de linhas de código produzidas pelos grupos, o tempo total despendido para a construção das versões das interfaces e a produtividade de cada grupo em termos de linhas de código produzidas por hora. Essas informações foram calculadas apurando os dados coletados pelos grupos após a realização do experimento.

As linhas rotuladas como "Média" mostram as médias de tempo gasto pelos grupos do processo *Model Driven RichUbi* e do ciclo de vida clássico na execução de cada uma das atividades dos processos, bem como o número médio de linhas de código geradas tanto manualmente quanto automaticamente. Também são mostradas as médias do total geral de linhas de código, do tempo total despendido (μ_τ) e da produtividade dos grupos ($\mu_Þ$). Ressalte-se que, conforme observado na Tabela 5.3, a produtividade é a medida proporcional ao LOC total e tempo total. Dessa forma, $\mu_Þ$, que é obtida pela média aritmética das produtividades de cada grupo, pode diferir da razão entre o número médio de linhas de código totais e o tempo total médio despendido.

Os grupos foram instruídos a relatar os problemas técnicos encontrados durante a execução do experimento. Para tanto, no formulário de coleta de dados foram colocados campos apropriados para serem preenchidos com a descrição dos problemas ocorridos, bem como o horário de identificação e de resolução dos mesmos. A Tabela 5.5 detalha os dados preenchidos pelos grupos relacionados aos problemas enfrentados, indicando os tempos em que cada problema foi solucionado pelos grupos. O tempo total gasto por cada grupo para solução dos problemas

encontra-se sumarizado na coluna “Problemas” da Tabela 5.3. Esse tempo não foi descontado dos tempos totais dos grupos uma vez que problemas de ordem técnica podem ocorrer em qualquer desenvolvimento.

Tabela 5.5 – Problemas técnicos enfrentados pelos grupos.

Grupo	Descrição do Problema	Identi-ficação	Reso-lução	Tempo Gasto
G1	Erro para carregar a página inicial (*HTTP Status 404*)	17:25	18:08	00:43
G2	Problema na exibição do mapa (API JavaScript do *Google Maps*)	17:20	17:45	00:25
	Problema com o uso da biblioteca jQuery	17:50	18:00	00:10
G3	Problemas com folhas de estilo	17:20	17:54	00:34
G4	Problemas na exibição do mapa	17:39	17:45	00:06
	Bibliotecas ausentes na pasta *WEB-INF/lib*	17:37	17:42	00:05
G5	---	---	---	00:00
G6	Problemas com bibliotecas do servidor *Apache TomCat*; erros na criação do banco de dados; e problemas com a importação de folhas de estilo e da biblioteca jQuery	17:10	17:40	00:30
G7	Problemas com a classe *WURFLAdapter* para implementação da adaptação estática	17:27	17:42	00:15
	Problema na chamada da função JavaScript “loadContent” para atualizar o mapa	17:40	17:52	00:12
	Erro ao carregar o mapa e o painel de abas	17:53	18:04	00:11
G8	Bibliotecas ausentes na pasta *WEB-INF/lib*	17:28	17:43	00:15
	Erro ao carregar o mapa e o painel de abas	17:43	18:00	00:17
G9	Falha na importação do *framework* UbiCon	17:02	17:04	00:02
G10	Problemas com bibliotecas do servidor *Apache TomCat*	17:06	17:17	00:11

3) Validação dos Dados. Após a execução do experimento, verificou-se se os dados registrados pelos grupos eram razoáveis, bem como se foram coletados de forma correta. Durante a apuração, observou-se uma diferença muito grande no total de linhas de código gerada automaticamente reportado pelo grupo G10 em relação aos demais grupos. O grupo havia registrado em seu formulário que 963 linhas de código foram geradas através das transformações M2C. A média desse valor para os demais grupos do *Model Driven RichUbi* era de 273, o que resultava em uma diferença de 690 linhas de código. Após contatar o grupo, verificou-se que o mesmo havia incluído no somatório o código dos arquivos *JavaScript* e das folhas de estilo da biblioteca *jQuery* que eram importados ao projeto durante a aplicação das transformações. Conforme orientado aos grupos, apenas o código dos arquivos de script e de folhas de estilo personalizados deveria ser somado. Após nova contabilização, chegou-se ao valor de 252 linhas de código geradas automaticamente para o grupo G10. A Tabela 5.3 já encontra-se com esse valor corrigido.

Ao analisar o formulário do grupo G8, que aplicou o ciclo de vida clássico, observou-se que não constavam os dados de início e fim da atividade de *Projeto*. Isso porque seus integrantes realizaram mais rapidamente a atividade de *Projeto*, partindo logo para a *Implementação*. É interessante notar que o grupo G8 foi o que apresentou maior nível médio de experiência dentre os demais grupos do experimento (Figura 5.12). Outro grupo que também realizou o *Projeto* de forma rápida foi o grupo G6. Porém, ao contrário do grupo G8, o grupo G6 registrou em seu formulário o tempo de 1 minuto gasto com essa atividade (entre 16:40 e 16:41). Uma vez que, apesar da rápida execução, houve tarefas associadas à disciplina de Projeto em ambos os grupos, os dados tanto do grupo G8 quanto do grupo G6 foram considerados válidos.

De forma geral, constatou-se que os grupos desempenharam bem a tarefa experimental que lhes foi proposta, executando as atividades com seriedade. Os treinamentos realizados nas três semanas anteriores ao experimento colaboraram para que os participantes se habituassem à construção das interfaces ricas adaptativas com ambos os processos. Assim, durante o experimento os grupos não tiveram maiores problemas para executarem as atividades. Os tratamentos, portanto, foram aplicados corretamente conforme planejado no projeto experimental, sendo os dados coletados considerados válidos para o experimento.

5.3.2.4 Análise e Interpretação dos Resultados

A partir de uma análise preliminar dos dados apresentados na Tabela 5.3, alguns aspectos interessantes foram observados entre os grupos que utilizaram o processo *Model Driven RichUbi* e os grupos que aplicaram o ciclo de vida clássico. A Figura 5.13 apresenta a distribuição média dos esforços dos grupos entre as atividades executadas nos processos. Os grupos gastaram tempos similares no projeto e testes das interfaces desenvolvidas. No entanto, nota-se uma redução de esforços significativa para implementação nos grupos que usaram o *Model Driven RichUbi*. Enquanto os grupos do ciclo de vida clássico gastaram, em média, 49 minutos para a codificação das interfaces, os grupos do *Model Driven RichUbi* levaram, em média, apenas 18 minutos (redução de 63,3%). Apesar dos grupos do *Model Driven RichUbi* terem gasto menos tempo na implementação do que os grupos do ciclo de vida clássico, aqueles foram mais produtivos (média de 301 LOC/h) do que estes (média de 104 LOC/h). A média de linhas de código totais implementadas pelos grupos do *Model Driven RichUbi* foi de 277

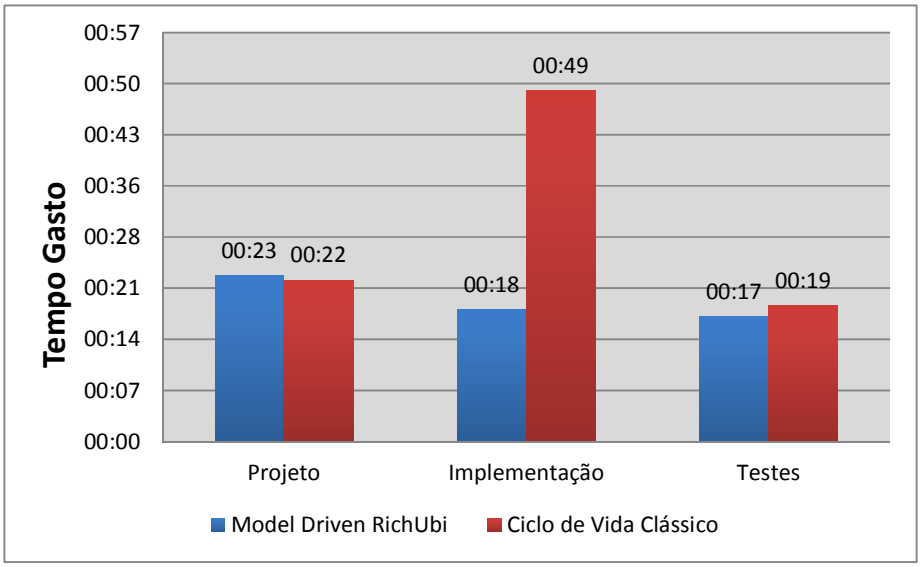


Figura 5.13 – Distribuição média dos esforços por atividades.

LOC, ao passo que os grupos do ciclo de vida clássico produziram, em média, apenas 156 LOC. Essa relação inversamente proporcional entre tempo despendido e quantidade de linhas de código geradas pode ser justificada pelo emprego das transformações M2C no processo *Model Driven RichUbi*. Os dados evidenciam que o esforço de implementação pode ser significativamente reduzido sem afetar negativamente a produtividade da equipe, uma vez que grande parte das tarefas de codificação podem ser encapsuladas nas transformações – o que é comum ocorrer em processos automatizados que se baseiam em MDD. Conforme mostra a Tabela 5.3, em média, 83% do código implementado pelos grupos do *Model Driven RichUbi* foi gerado automaticamente pelas transformações (≈ 230 LOC). Já para os grupos do ciclo de vida clássico, os quais não utilizaram geradores de código, a média de código gerado de forma automática pelo ambiente de desenvolvimento foi apenas de 13% (≈ 20 LOC). Por outro lado, analisando-se somente o tempo gasto para construir as interfaces, também é possível obter um retrato da produtividade alcançada. Em média, o tempo gasto pelos grupos que utilizaram o *Model Driven RichUbi* foi 50% menor (30 min), o que indica que esses grupos puderam desenvolver o código das interfaces em um período de tempo mais curto, o que os tornou mais produtivos.

Notou-se também que o esforço de modelagem realizado na atividade de Projeto foi mais bem aproveitado pelos grupos do processo *Model Driven RichUbi*. Conforme ilustrado na Figura 5.13, ambos os grupos gastaram tempos similares no projeto das interfaces (23 min – *RichUbi* vs. 22 min – ciclo de vida clássico), porém os grupos que executaram o *Model Driven RichUbi* puderam usufruir melhor do

esforços envidados para modelagem das interfaces, uma vez que os modelos produzidos serviram como entrada para a geração de grande parte do código.

Outro aspecto interessante que se pôde observar nos dados coletados foram as relações entre o nível médio de experiência dos grupos e os tempos totais para a conclusão da tarefa experimental e resolução dos problemas durante a construção das interfaces. Entre os grupos que executaram o *Model Driven RichUbi*, o grupo G5, com maior nível médio de experiência (1,58), não reportou nenhum problema durante a construção das interfaces. Já o grupo G1, o menos experiente (1,19), foi o que despendeu mais tempo para solucionar problemas: 43 minutos. Porém, dentre esses grupos, houve aqueles que, mesmo com níveis menores de experiência, foram mais rápidos ou gastaram o mesmo tempo para resolução de problemas do que grupos com níveis de experiência mais elevados. Este é o caso, por exemplo, dos grupos G9 (1,31/2 min) e G4 (1,47/11 min), e dos grupos G4 (1,47/11 min) e G10 (1,50/11 min). Com relação ao tempo para finalização da tarefa experimental, o grupo G10 foi o mais rápido para concluir as interfaces, gastando em torno de 36 minutos. Nota-se que, mesmo o grupo G5 tendo sido o mais experiente entre os grupos do *Model Driven RichUbi*, o mesmo não foi o que levou menos tempo para implementar as interfaces, gastando 18 minutos a mais que o grupo mais rápido, G10, e 10 minutos a mais que o grupo G9 (que gastou 44 min). Isto indica que, independentemente do nível de conhecimento da equipe, o processo *Model Driven RichUbi* colaborou para simplificar o desenvolvimento das interfaces ricas adaptativas tanto para os grupos com mais experiência, como também para aqueles menos experientes, o que sugere que o processo foi de fácil aprendizagem por todos os grupos.

Observou-se também uma diferença grande com relação à média de tempo despendido para tratar problemas técnicos pelos grupos do ciclo de vida clássico com relação à mesma média dos grupos do *Model Driven RichUbi*. Enquanto estes gastaram, em média, 13 minutos, aqueles levaram um tempo médio de 33 minutos resolvendo problemas. Esta diferença justifica-se pelo fato dos grupos do processo *Model Driven RichUbi* terem sido apoiados por transformações M2C durante o desenvolvimento, já que estas abstraíram grande parte das tarefas de codificação. Os grupos do ciclo de vida clássico, porém, que implementaram a maior parte do código manualmente, tiveram retrabalhos para a criação das duas versões das interfaces da aplicação, o que resultou numa frequência maior de erros de sintaxe e equívocos na definição de parâmetros e diretivas de configuração.

Após essas observações iniciais, passou-se para os passos de obtenção de conclusões do estudo, envolvendo a caracterização dos dados através *da estatística descritiva*, a *redução do conjunto de dados* e o *teste das hipóteses*. Esses passos são descritos a seguir:

1) Estatística Descritiva. Este passo lida com a representação e processamento numérico do conjunto de dados, representando-o graficamente para melhorar o entendimento de como os dados estão distribuídos e ajudar a identificar *outliers*, i.e., dados anormais ou atípicos para o conjunto de dados das amostras.

A Figura 5.14 apresenta os gráficos de caixa (*boxplot*) que mostram a dispersão dos tempos totais (Figura 5.14(a)) e das produtividades (Figura 5.14(b)) dos grupos participantes do experimento.

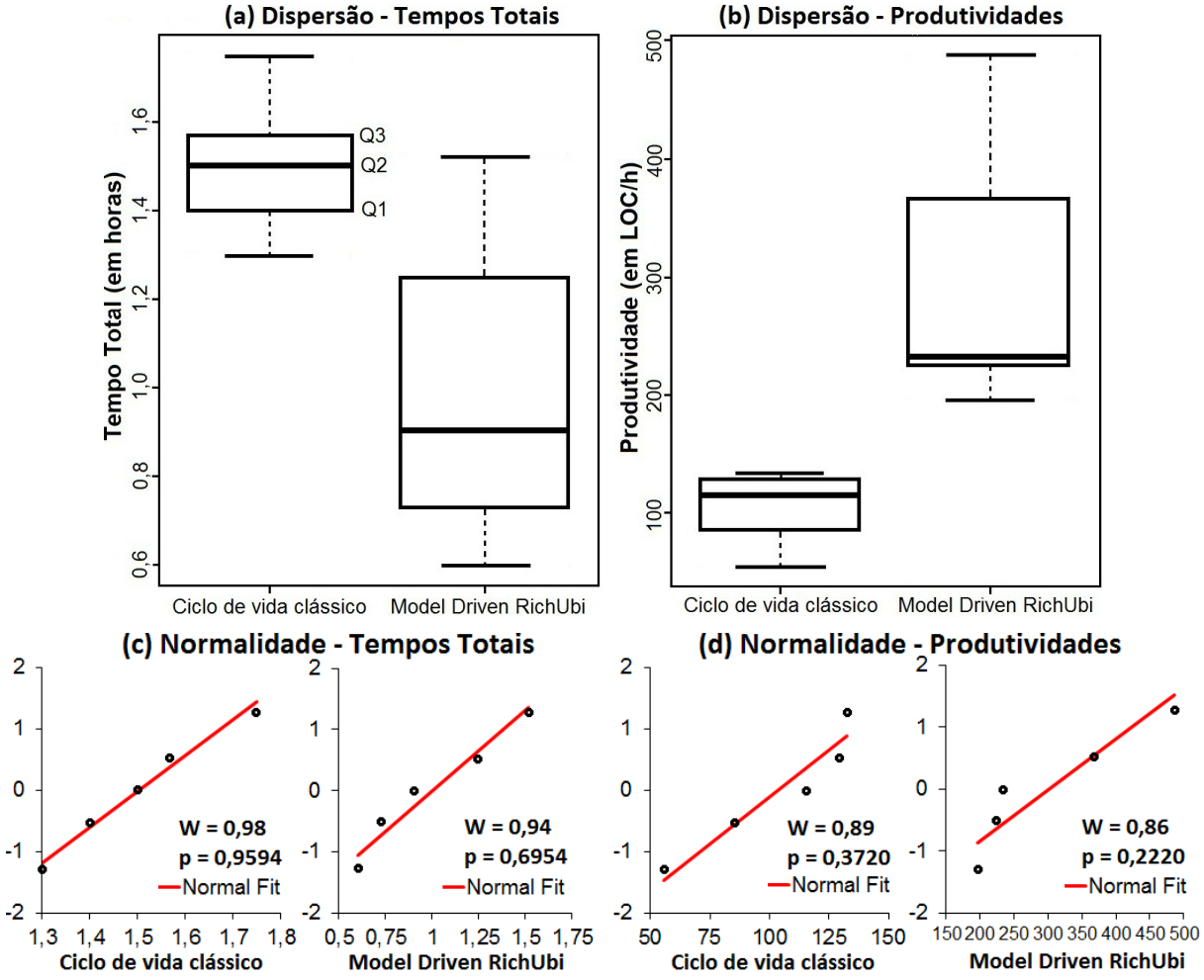


Figura 5.14 – Estatística descritiva dos dados das amostras.

No *boxplot*, a linha sobre cada caixa marca a mediana do conjunto de dados e representa o segundo quartil (Q2) da distribuição. As partes inferior e superior das caixas são delimitadas, respectivamente, pelos quartis inferior (Q1) e superior (Q3). As

hastes que se estendem acima e abaixo das caixas representam o limite teórico dentro do qual provavelmente são encontrados todos os dados da amostra se a distribuição for normal. A haste inferior se estende do quartil inferior até o menor valor não inferior a Q1 – 1,5*IQR, onde IQR é o intervalo interquartil (Q3 – Q1). Já a haste superior se estende do quartil superior até o maior valor não superior a Q3 + 1,5*IQR. Os valores que se encontram abaixo ou acima desses limites são representados como pontos individuais no gráfico, sendo os mesmos caracterizados como *outliers*. A verificação da normalidade da distribuição dos dados das amostras foi feita através do teste não paramétrico *Shapiro-Wilk* (MONTGOMERY, 2000) (Figura 5.14(c) e (d)).

Conforme se observa nos gráficos de caixa da Figura 5.14, nenhum *outlier* foi identificado no conjunto de dados. A partir desses gráficos também é possível observar claramente a tendência dos grupos do *Model Driven RichUbi* em concluir a implementação de forma mais rápida e com maior produtividade, ao contrário dos grupos que não aplicaram o *Model Driven RichUbi*, que tenderam a ser mais lentos e menos produtivos.

2) Redução do Conjunto de Dados. Os métodos estatísticos para o teste das hipóteses dependem da qualidade dos dados de entrada para obtenção de bons resultados. Se os dados sobre os quais um método estatístico é aplicado estiverem incorretos, então os resultados do método também o estarão. É necessário, portanto, remover das amostras todos os dados redundantes e os que podem levar a resultados errôneos. Falhas no conjunto de dados geralmente ocorrem por conta de erros sistemáticos (e.g. erros durante a transcrição dos dados), ou então pela presença de *outliers*, que são como “pontos fora da curva”, i.e., são muito menores ou muito maiores do que se poderia esperar em relação aos demais dados do conjunto.

Conforme constatado no passo de estatística descritiva, nenhum *outlier* foi encontrado nas amostras. Além disso, todos os formulários com os dados coletados pelos grupos foram cuidadosamente revisados e nenhum erro de transcrição foi observado. Portanto, não houve necessidade de redução do conjunto de dados.

É importante notar que este passo tem correlação com o passo “Validação dos Dados” da fase de Execução do estudo, cujo objetivo foi identificar dados inválidos com base na execução do experimento, como, por exemplo, determinar se os grupos participaram seriamente do estudo fornecendo dados confiáveis. Neste passo, a redução dos dados foca na identificação de *outliers* e baseia-se não apenas na forma

com que o estudo foi executado, como também na análise dos resultados obtidos nos formulários de coleta de dados considerando, para isso, a estatística descritiva.

3) Teste das Hipóteses. O objetivo deste passo é verificar, com algum **grau de significância**, se é possível rejeitar a hipótese nula (H_0) em favor de alguma das hipóteses alternativas (H_1 ou H_2) com base no conjunto de dados obtido.

A princípio, analisando os dados obtidos no experimento, aparentemente o uso do processo *Model Driven RichUbi* para a construção de interfaces ricas adaptativas aumentou a eficiência dos grupos se comparados com os grupos que não usaram o processo proposto. Para comprovar esse efeito de forma estatística, aplicou-se o teste chamado *t-test* (MONTGOMERY, 2000). Esse teste paramétrico é usado para comparar duas amostras independentes e checar se as médias de seus dados são estatisticamente diferentes e, portanto, mostrar que o efeito hipotético foi demonstrado. No caso do estudo realizado, as amostras foram constituídas pelos dados referentes aos tempos totais e produtividades tanto dos grupos do processo *Model Driven RichUbi*, quanto dos grupos do ciclo de vida clássico.

O *t-test* se baseia na ideia de que, quando se procura por diferenças entre duas amostras *X* e *Y*, deve-se julgar a diferença entre suas médias considerando a dispersão ou variabilidade dos dados que as compõem. Isto porque, quanto maior a variabilidade dos dados das amostras comparadas, maiores as chances desses dados estarem sobrepostos na distribuição normal, mesmo que a diferença das médias se mantenha constante. A Equação (1) apresenta a fórmula do *t-test*. Trata-se de uma razão: no numerador tem-se a diferença das médias, e no denominador tem-se a medida da variabilidade ou dispersão dos dados. Essa medida é obtida através das variâncias das amostras (S_x^2 e S_y^2) e do número de itens de dados em cada amostra (n e m), conforme cálculo apresentado na Equação (2).

$$t_0 = \frac{\bar{x} - \bar{y}}{S_p \sqrt{\frac{1}{n} + \frac{1}{m}}} \qquad (1)$$

$$S_p = \sqrt{\frac{(n-1)\,S_x{}^2 + (m-1)S_y{}^2}{n + m - 2}} \qquad (2)$$

Após o cálculo de t_0 deve-se verificar a **tabela padrão de distribuição de probabilidade estatística *t* de *Student*** (Anexo A) (GOSSET, 1947) para descobrir se a razão calculada é suficientemente grande de modo que se possa atestar que é

improvável que a diferença amostral tenha sido mera causalidade. Estabelece-se, portanto, um grau de significância α que represente o "ponto de corte" ou "nível de risco" com o qual é permitido afirmar que há diferença entre as amostras e rejeitar a hipótese nula. Por exemplo, tomando-se $\alpha = 0,05$, significa que se assume um risco de 5% em se encontrar uma diferença significativa entre as médias das amostras, mesmo que essa diferença fosse por acaso (falso positivo). O nível de confiança do resultado do teste neste caso seria de 95%. Também é necessário estabelecer o **grau de liberdade (*gl*)** para o teste, o qual corresponde à soma do número de dados nas amostras menos dois ($m + n - 2$).

Assim, tendo t_0, α e *gl*, observa-se o valor de *t* na tabela padrão para determinar se o valor de t_0 é grande o suficiente para ser significativo. O valor *t* checado na tabela reflete a probabilidade de ter sido obtida a diferença observada entre as médias das amostras numa situação onde as eficiências dos grupos de ambos os processos se equiparam, i.e., a hipótese nula é verdadeira. Se essa probabilidade for muito pequena, então pode-se concluir que o resultado observado no estudo é estatisticamente relevante. Neste caso, se o valor *t* da tabela referente ao nível de risco escolhido e ao grau de liberdade for inferior ao t_0 calculado, a hipótese nula é rejeitada. Caso contrário, a hipótese nula não é rejeitada e nenhuma conclusão poderá ser obtida do experimento.

Uma vez que o efeito na variável dependente do experimento (eficiência da equipe) envolveu dois aspectos, tempo total (τ) e produtividade ($Þ$), a aplicação do *t-test* ao conjunto amostral de dados foi realizada em duas etapas. Na primeira etapa, comparou-se as amostras relativas aos tempos totais de cada grupo. Já na segunda etapa, a comparação foi feita com as amostras referentes às produtividades dos grupos. O teste da hipótese nula se baseou na combinação dos critérios de rejeição das duas etapas do teste, de modo que H_0 apenas seria rejeitada se, e somente se, pudesse ser rejeitada nas duas etapas do teste. Para os propósitos deste estudo, em cada etapa do teste utilizou-se o menor grau de significância α com o qual fosse possível rejeitar a hipótese nula, assumindo-se um grau de significância máximo de 6%. As Tabelas 5.6 e 5.7 apresentam a organização do teste realizado.

3.1) 1ª Etapa. Com base nas amostras de τ, tem-se que $|X_{\tau RichUbi}| = 5$ e $|Y_{\tau Clássico}| = 5$. Os valores médios são $\mu_{\tau RichUbi} = 1,0000$ e $\mu_{\tau Clássico} = 1,50334$. Pode-se, então, obter $S_{X\tau RichUbi}^2 = 0,1426$ e $S_{Y\tau Clássico}^2 = 0,0292$. Assim, $S_{p\tau} = 0,2932$ e $t_{0\tau} = -2,7148$.

Tabela 5.6 – Organização da 1ª etapa do *t-test* (tempo total).

	Tempo Total (τ)
Entrada	Duas amostras independentes: $X_{\tau RichUbi}$ = {1.52, 1.25, 0.9, 0.73, 0.60} e $Y_{\tau Clássico}$ = {1.75, 1.57, 1.50, 1.40, 1.30} (dados em horas).
$H_{0\tau}$	**$H_{0\tau}$:** $\mu_{\tau RichUbi} = \mu_{\tau Clássico}$, i.e., os valores médios para os tempos totais dos grupos de ambos os processos são iguais.
Cálculo de $t_{0\tau}$	$$t_{0\tau} = \frac{\mu_{\tau RichUbi} - \mu_{\tau Clássico}}{S_{p\tau}\sqrt{\frac{1}{\lvert X\tau_{RichUbi}\rvert} + \frac{1}{\lvert Y\tau_{Clássico}\rvert}}}$$ $$S_{p\tau} = \sqrt{\frac{(\lvert X\tau_{RichUbi}\rvert - 1)\, S_{X\tau RichUbi}{}^2 + (\lvert Y\tau_{Clássico}\rvert - 1) S_{Y\tau Clássico}{}^2}{\lvert X\tau_{RichUbi}\rvert + \lvert Y\tau_{Clássico}\rvert - 2}}$$
Critério de Rejeição de $H_{0\tau}$	• **$H_{1\tau}$:** $\mu_{\tau RichUbi} < \mu_{\tau Clássico}$; • **$H_{2\tau}$:** $\mu_{\tau RichUbi} > \mu_{\tau Clássico}$; Logo, $\mu_{\tau RichUbi} \neq \mu_{\tau Clássico}$ (bicaudal): rejeitar $H_{0\tau}$ se $\lvert t_{0\tau}\rvert > t_{\alpha/2,gl\tau}$

O número de graus de liberdade é $gl_\tau = |X_{\tau RichUbi}| + |Y_{\tau Clássico}| - 2 = 5 + 5 - 2 = 8$. Tomou-se $\alpha = 0{,}055$. Na tabela padrão de distribuição de probabilidade estatística *t* de *Student*, verifica-se que $t_{0.0275,8} = 2{,}6899$. Desde que $|t_{0\tau}| > t_{0.0275,8}$ **foi possível rejeitar a hipótese nula $H_{0\tau}$ com 5,5% de significância.**

Tabela 5.7 – Organização da 2ª etapa do *t-test* (produtividade).

	Produtividade (Þ)
Entrada	Duas amostras independentes: $X_{ÞRichUbi}$ = {233, 196, 367, 225, 487} e $Y_{ÞClássico}$ = {133, 56, 115, 86, 129}.
$H_{0Þ}$	**$H_{0Þ}$:** $\mu_{ÞRichUbi} = \mu_{ÞClássico}$, i.e., os valores médios para as produtividades dos grupos de ambos os processos são iguais.
Cálculo de $t_{0Þ}$	$$t_{0Þ} = \frac{\mu_{ÞRichUbi} - \mu_{ÞClássico}}{S_{pÞ}\sqrt{\frac{1}{\lvert X_{ÞRichUbi}\rvert} + \frac{1}{\lvert Y_{ÞClássico}\rvert}}}$$ $$S_{pÞ} = \sqrt{\frac{(\lvert X_{ÞRichUbi}\rvert - 1)\, S_{XÞRichUbi}{}^2 + (\lvert Y_{ÞClássico}\rvert - 1) S_{YÞClássico}{}^2}{\lvert X_{ÞRichUbi}\rvert + \lvert Y_{ÞClássico}\rvert - 2}}$$
Critério de Rejeição de $H_{0Þ}$	• **$H_{1Þ}$:** $\mu_{ÞRichUbi} > \mu_{ÞClássico}$; • **$H_{2Þ}$:** $\mu_{ÞRichUbi} < \mu_{ÞClássico}$; Logo, $\mu_{ÞRichUbi} \neq \mu_{ÞClássico}$ (bicaudal): rejeitar $H_{0Þ}$ se $\lvert t_{0Þ}\rvert > t_{\alpha/2,glÞ}$

3.2) 2ª Etapa. Com *base nas amostras de* Þ*, tem-se que* $|X_{ÞRichUbi}| = 5$ *e* $|Y_{ÞClássico}| = 5$*. Os valores médios são* $\mu_{ÞRichUbi} = 301{,}6$ *e* $\mu_{ÞClássico} = 103{,}8$*. Pode-se, então, obter* $S_{XÞRichUbi}{}^2 = 15093{,}8$ *e* $S_{YÞClássico}{}^2 = 1053{,}7$*. Assim,* $S_{pÞ} = 89{,}8540$ *e* $t_{0Þ} = 3{,}4806$*.*

O número de graus de liberdade é $gl_Þ = |X_{ÞRichUbi}| + |Y_{ÞClássico}| - 2 = 5 + 5 - 2 = 8$. Tomou-se $\alpha = 0{,}02$. Pela tabela padrão de distribuição de probabilidade estatística *t* de

Student, tem-se que $t_{0.01,8} = 3,3554$. Desde que $|t_{0Þ}| > t_{0.01,8}$ **foi possível rejeitar a hipótese nula $H_{0Þ}$ com 2% de significância**.

A análise dos dados deste estudo foi realizada utilizando o sistema *R*[39] e o software *RKWard*[40], uma interface gráfica para o *R*.

3.3) Conclusões do Experimento. Uma vez rejeitada a hipótese nula (H_0), é possível tirar conclusões a respeito da influência das variáveis independentes sobre a variável dependente, considerando que o experimento é válido tendo sido tratadas as ameaças à validade (Seção 5.3.2.6).

Com a rejeição de H_0 no experimento, pode-se afirmar que as diferenças observadas na eficiência dos grupos que executaram o processo *Model Driven RichUbi* e dos grupos que seguiram o ciclo de vida clássico possuem significância estatística. Assim, provavelmente a mudança na eficiência dos grupos foi devido aos dois processos usados como tratamento e não a um acaso por erros aleatórios de amostragem.

Conforme mostra a Tabela 5.3, a média de tempo total dos grupos do processo *Model Driven RichUbi* foi menor do que aquela dos grupos do ciclo de vida clássico ($\mu_{\tau RichUbi} < \mu_{\tau Clássico}$). Já a média de produtividade dos grupos teve uma relação inversa: os grupos do *Model Driven RichUbi* foram, em média, mais produtivos do que os demais grupos ($\mu_{ÞRichUbi} > \mu_{ÞClássico}$). Esses dados fornecem evidências de que a hipótese alternativa H_1 pode ser validada em detrimento da hipótese H_2, ou seja, há indícios para se afirmar que equipes utilizando o *Model Driven RichUbi* na construção de interfaces ricas adaptativas são, em geral, mais eficientes do que equipes que aplicam o ciclo de vida clássico, e não o contrário. Este resultado está em conformidade com as expectativas iniciais do experimento, embora estas não tenham sido declaradas formalmente nas hipóteses. As expectativas eram de que os artefatos reutilizáveis construídos na etapa de Engenharia de Domínio do *Model Driven RichUbi* (metamodelo, transformações M2C e adaptadores de conteúdo) tornariam mais ágeis as tarefas dos desenvolvedores de aplicações.

Finalmente, considerando que o experimento foi realizado *in-vitro* sob condições controladas, é importante manter em mente que as conclusões a respeito dos resultados observados neste trabalho se restringem ao escopo de desenvolvedores de software em ambiente universitário no qual o estudo foi conduzido. Por questões de validade, para

[39] http://www.r-project.org/.

[40] http://rkward.sourceforge.net/.

expandir a generalização do fenômeno observado para um contexto mais amplo, é necessário que novos estudos com um número maior de equipes sejam realizados em ambientes *in-vivo*, comparando-se também o uso do *Model Driven RichUbi* em relação a outras abordagens de desenvolvimento, tais como *métodos ágeis* e *Linhas de Produto de Software*, por exemplo. Isto permitirá obter uma validação mais abrangente das hipóteses de pesquisa. A replicação deste experimento em ambiente industrial, portanto, torna-se importante já que outros fatores ausentes no ambiente acadêmico poderão ser controlados e estudados. Espera-se que o pacote de experimentação, disponível em *http://www.dc.ufscar.br/~carlos_cirilo/package-mdrichubi.zip*, possa ser reutilizado por pesquisadores e profissionais em novos experimentos em diferentes contextos a fim de contribuir com a descoberta de novas relações de causa e efeito potencialmente existentes em ambientes experimentais heterogêneos.

5.3.2.5 Avaliação Qualitativa

Apesar do escopo do estudo ter se restringido ao contexto universitário, a partir de um ponto de vista de validade é factível afirmar que as atividades e os artefatos de suporte do processo *Model Driven RichUbi* colaboraram para o aumento da eficiência dos grupos. A geração de código para diferentes tecnologias de implementação a partir de um mesmo modelo contribuiu para reduzir os esforços, sendo que a implementação de cada versão da interface não necessita ser realizada a partir do zero. Essa característica é inerente às abordagens de desenvolvimento dirigido a modelo. Além disso, o uso de modelos específicos do domínio de interfaces ricas também facilitou a tradução dos requisitos da aplicação em componentes de interface que atendam a esses requisitos, além de tornar os modelos mais intuitivos e fáceis de manusear. O reúso do *framework UbiCon* para adaptação híbrida de conteúdo também contribuiu para agilizar o desenvolvimento, visto que os grupos não necessitaram se preocupar com questões de manipulação de contexto e adaptação das interfaces. Dessa forma, os grupos do processo *Model Driven RichUbi* puderam desenvolver mais código em um período menor de tempo do que os grupos do ciclo de vida clássico.

Para validar essas observações junto aos participantes do estudo, após a execução do experimento os alunos receberam um formulário de avaliação apropriado, conforme o processo que haviam executado (Apêndices F e K), para reportarem sua percepção com relação ao uso do processo *Model Driven RichUbi* ou do ciclo de vida clássico para construir interfaces ricas adaptativas.

Dos alunos respondentes que aplicaram o *Model Driven RichUbi*, todos consideraram que o processo proposto auxiliou no desenvolvimento das interfaces. De acordo com as respostas desses alunos, o principal motivo do processo ter facilitado as tarefas de desenvolvimento das interfaces foi o fato de se empregar transformações para geração do código a partir dos modelos. Os alunos consideraram que a implementação foi feita de forma mais rápida, gerando interfaces mais "bonitas" e com melhor desempenho graças ao emprego da adaptação híbrida. Segundo os alunos, não houve necessidade de se preocupar com os detalhes de estruturação inicial das interfaces e da produção de folhas de estilo, já que estes já eram gerados pelas transformações. Isto também facilitou a implementação.

Com relação à dificuldade de utilização do processo, 57% dos alunos do processo *Model Driven RichUbi* responderam que não sentiram dificuldades, e 43% reportaram que sentiram parcialmente. Estes últimos indicaram que a dificuldade está relacionada ao fato do processo ser relativamente novo para o grupo, apesar de ser simples e de fácil aprendizado. Interessante foi que um dos alunos respondeu que a dificuldade estava na identificação de erros após a complementação do código. Isto porque a maior parte do código era gerada de forma automática pelas transformações, o que fazia com que o aluno tivesse que primeiro entender o código gerado para conseguir identificar a natureza do erro. Essa observação reflete a importância de se produzir transformações que gerem código bem estruturado, de maneira que fique fácil para o desenvolvedor compreender e complementar o código gerado. Isto sugere que transformações mal implementadas podem até mesmo dificultar o desenvolvimento, ao invés de simplificar, uma vez que o tempo ganho com a geração automática pode ser perdido na correção de defeitos no código e entendimento do mesmo.

Por outro lado, dentre os alunos respondentes que aplicaram o ciclo de vida clássico, 29% responderam não ter sentido dificuldades no desenvolvimento das interfaces ricas adaptativas, ao passo que 71% responderam que houve uma dificuldade parcial durante a implementação. A maior dificuldade relatada por esses alunos foi a redundância e retrabalho para a criação das duas versões da interface, o que era feito de forma manual já que nenhum desses alunos utilizaram geradores automatizados de código, dando margens para erros de sintaxe e esquecimentos na definição de parâmetros de configuração. Esses alunos responderam unanimemente que o uso de um processo adequado que fornecesse um roteiro e mecanismos de apoio especificamente voltados para o desenvolvimento de interfaces ricas adaptativas teria

facilitado sua tarefa. Eles consideraram que um processo com diretrizes e geradores parciais de código das interfaces poderia contribuir para a redução dos esforços, tornando o desenvolvimento mais simples e rápido, elevando o nível de abstração, melhorando a compreensão do problema e produzindo artefatos com mais qualidade.

Com relação à modelagem, 86% dos alunos que executaram o processo *Model Driven RichUbi* consideraram que houve ganhos de tempo com o uso de elementos específicos do domínio de interfaces ricas nos modelos. Outros 14% acharam que os ganhos foram parciais. Os alunos reportaram que os modelos se tornaram mais fáceis e rápidos de implementar, abstraindo os detalhes técnicos de implementação e padronizando a definição dos componentes, o que evitou a inserção de erros conceituais em relação ao Domínio de Interfaces Ricas nos modelos. Porém, alguns alunos consideraram que, pelo fato dos elementos dos modelos serem voltados especificamente para componentes de interfaces ricas, isso pode limitar as possibilidades dos desenvolvedores durante a especificação dos modelos, o que requer que o metamodelo esteja corretamente implementado para não restringir demasiadamente os desenvolvedores durante a modelagem.

Quanto ao emprego da adaptação híbrida ou estática, os alunos respondentes do processo *Model Driven RichUbi* em sua maioria (86%) disseram que a abordagem híbrida é viável do ponto de vista de esforço de desenvolvimento e eficiência da adaptação. Outros 14% responderam que parcialmente. Com relação ao esforço, os alunos consideram que a abordagem híbrida pode potencialmente contribuir para economizar tempo, uma vez que desenvolve-se apenas uma versão da interface que possa atender a diversos dispositivos. Segundo os alunos, isso se torna particularmente útil à medida que aumenta-se a utilização de aplicações a partir de dispositivos móveis. Com relação ao desempenho, os alunos observaram um melhor carregamento das páginas no dispositivo. Por outro lado, dentre os alunos respondentes do ciclo de vida clássico, todos responderam que a estratégia de adaptação estática é inviável (71%) ou parcialmente viável (29%) para o desenvolvimento de interfaces ricas adaptativas. O principal motivo que inviabiliza essa estratégia, segundo os alunos, é a diversidade de dispositivos atualmente disponíveis, o que torna difícil desenvolver uma interface especializada que atenda a cada um.

5.3.2.6 Ameaças à Validade

Uma das questões fundamentais envolvendo os resultados de um experimento é determinar quão válidos eles são. Por isso, é importante considerar a questão da validade

da avaliação desde a fase de planejamento de forma a guiar a execução do experimento no sentido de obter uma validade adequada dos resultados. Validade adequada, neste caso, significa que os resultados são válidos para a população de interesse, tanto para o subconjunto populacional que executou o estudo, quanto para uma população mais ampla para a qual os resultados podem ser generalizados (WOHLIN et al., 2000).

As principais ameaças que podem colocar em risco a validade de um experimento se classificam em quatro tipos (WOHLIN et al., 2000): **de conclusão**, **interna**, **de construção** e **externa**. Essas ameaças foram tratadas no experimento realizado, conforme segue:

1) Validade de Conclusão. Refere-se às questões que afetam a habilidade de tirar conclusões corretas a respeito do efeito dos tratamentos nas variáveis dependentes (e.g. escolha do método estatístico adequado para análise, cuidados tomados na execução e na medição do experimento). No caso do estudo, em se tratando de um projeto do tipo um fator com dois tratamentos, o teste estatístico *t-test* foi adotado. Esse teste é o mais adequado para projetos desse tipo, onde o objetivo é comparar as médias obtidas nos resultados de dois tratamentos distintos. O *t-test* usualmente requer dados normalmente distribuídos para comparação. Embora o teste de normalidade de *Shapiro-Wilk* (Figura 5.14(c) e (d)) tenha dado positivo para as amostras do estudo a 5% de significância, para assegurar a correção da análise em vista do pequeno tamanho do conjunto amostral de dados – o que dificulta obter uma conclusão precisa a respeito de sua normalidade – foi também realizado o teste *Mann-Whitney* (um teste não paramétrico, alternativo ao *t-test* em casos onde os dados não estão normalmente distribuídos). Com esse teste foram observados resultados semelhantes. Entretanto, conforme indicado por Wohlin et al. (2000), mesmo se as amostras não estivessem em sua distribuição normal, o *t-test* é robusto o suficiente para suportar alguns desvios dessa pré-condição. Particularmente, sendo um teste paramétrico, sua potência é, em geral, maior do que a de testes não-paramétricos, o que também justifica seu uso para obter resultados mais precisos no estudo. Além disso, as medidas que deveriam ser coletadas pelos participantes envolveram dados diretos (hora, LOC) que independem de julgamento humano, ao contrário de medidas como *pontos por função* (PRESSMAN, 2004), por exemplo. Isto aumentou a confiabilidade dos dados coletados, mesmo sob o risco de possíveis imprecisões nos dados, já que estes foram coletados pelos próprios grupos. Ademais, a heterogeneidade da experiência dos participantes do experimento que poderia

afetar os resultados foi tratada pela estratégia de blocos, com a formação de grupos homogêneos que continham, em média, níveis de experiência similares.

2) Validade Interna. Refere-se às questões que afetam a habilidade de assegurar que os resultados foram, de fato, obtidos em decorrência dos tratamentos e não por uma coincidência, ou pelo efeito de outro fator que não foi medido ou não se pôde controlar (e.g. modo como os participantes são selecionados, agrupados, recompensados e tratados; situação em que ocorre o experimento). No caso do estudo, buscou-se realizar o experimento com estudantes da área de Computação em um estágio avançado de seus cursos de graduação (3º e 4º anos), os quais já possuíam certo nível de experiência com desenvolvimento de software, conforme reportado em seus formulários de caracterização. Assim, assumiu-se que eles são representativos para a população dos desenvolvedores de software. Além disso, os estudantes foram agrupados adequadamente conforme seus níveis de experiência para que os grupos ficassem homogêneos, evitando, dessa forma, discrepâncias na velocidade com que cada grupo aprendia a executar os processos à medida que interagiam com o experimento. Nenhum tipo de recompensa ou favorecimentos quanto à nota da disciplina foi oferecida aos estudantes de modo a não criar expectativas e evitar que se comportassem com empenho anormal durante o estudo para obtenção de vantagens neste sentido. Ainda, o estudo foi realizado em uma única sessão e simultaneamente com todos os grupos, de forma que todos os participantes estivessem sob as mesmas circunstâncias durante a execução do experimento.

3) Validade de Construção. Refere-se às questões que afetam a habilidade de generalizar o resultado do experimento ao conceito ou teoria envolvidos no estudo (e.g. definição adequada das medidas e tratamentos). No estudo realizado, o objetivo era comparar dois processos de desenvolvimento com respeito a seu impacto na eficiência da equipe. Para os propósitos do estudo, uma equipe eficiente é aquela que produz mais código em uma hora e finaliza a implementação em menos tempo. Portanto, os dados para coleta e os tratamentos foram definidos de modo que fosse possível realizar adequadamente a análise do efeito na eficiência das equipes em conformidade com os objetivos do estudo. Além disso, a fim de evitar interações dos participantes voltadas para maximização ou minimização das métricas de interesse (produtividade e tempo despendido), evitou-se divulgar exatamente quais métricas seriam analisadas quando os mesmos foram informados dos objetivos. Isso também evitou que os participantes se comportassem de maneira diferente do comum,

buscando se empenhar melhor ou agir de forma negligente mais do que de costume apenas para obterem resultados que favorecessem ou prejudicassem o experimento. O tratamento dessa ameaça permitiu mitigar dados inválidos para o estudo.

4) Validade Externa. Refere-se às questões que afetam a habilidade de generalizar os resultados do experimento para um contexto mais amplo do que o que foi selecionado para o estudo. Neste caso, existem três riscos principais: a escolha errada dos participantes, conduzir o experimento em um ambiente inapropriado, e desempenhar o estudo em um período de tempo que afete os resultados. No caso, os participantes do estudo, em geral, podem ser considerados representativos para a população dos desenvolvedores de software. O experimento foi conduzido em um laboratório informatizado com equipamentos adequados, bem como com ferramentas e tecnologias de desenvolvimento atualizadas comumente empregadas em ambientes industriais, tais como o IDE Eclipse e a linguagem de programação Java. Com relação às questões temporais, o experimento foi planejado de forma que os estudantes pudessem desempenhá-lo dentro de um período de 4 horas, evitando que os resultados fossem afetados em decorrência de tédio ou desgaste excessivo dos participantes.

5.4 Considerações Finais

Este capítulo apresentou a avaliação do trabalho desenvolvido. A avaliação compreendeu um estudo de caso preliminar, em que investigou-se a aplicabilidade do processo e o comportamento da adaptação híbrida fornecida pelo *framework UbiCon*; e um estudo comparativo, seguindo a metodologia experimental, onde avaliou-se o impacto do uso do processo *Model Driven RichUbi* e de um processo baseado no ciclo de vida clássico do software na eficiência da equipe de desenvolvimento.

Os resultados favoráveis obtidos com a avaliação do trabalho vão ao encontro das expectativas iniciais do processo desenvolvido, que incluem redução dos esforços, aumento da produtividade e melhoria da adaptação das interfaces ricas desenvolvidas através do uso do contexto.

Capítulo 6

TRABALHOS CORRELATOS

6.1 Considerações Iniciais

Diversos trabalhos relacionados ao desenvolvimento de aplicações sensíveis ao contexto e interfaces adaptativas têm sido propostos pela comunidade acadêmica, o que demonstra a relevância desse tema de pesquisa. Este capítulo apresenta alguns desses trabalhos que possuem certa relação com o trabalho desenvolvido nesta pesquisa.

A organização do capítulo está conforme segue: a Seção 6.2 apresenta os principais trabalhos encontrados na literatura; a Seção 6.3 fornece uma análise comparativa dos trabalhos correlatos ao trabalho aqui desenvolvido; por fim, a Seção 6.4 conclui o capítulo tecendo algumas considerações finais.

6.2 Trabalhos Encontrados na Literatura

Algumas das soluções pesquisadas na literatura objetivam a definição de *frameworks* que disponibilizam funcionalidades de adaptação de comportamento e conteúdo (FORTE et al., 2008; VAN WOENSEL et al., 2009a, 2009b). Outras soluções concentram-se na elaboração de ferramentas que facilitam o desenvolvimento de aplicações e interfaces adaptativas (PATERNÒ et al., 2008; VIANA & ANDRADE, 2008). Há também as pesquisas que têm foco na definição de

processos de software que orientam o desenvolvimento de aplicações adaptativas sensíveis ao contexto (SANTOS, 2008; VIEIRA et al., 2009).

6.2.1 CEManTIKA

Contextual Elements Modeling and Management through Incremental Knowledge Acquisition (CEManTIKA) (SANTOS, 2008; VIEIRA et al., 2009, 2011) é uma abordagem proposta para apoiar o projeto de aplicações sensíveis ao contexto em diferentes domínios. Essa abordagem, dentre outros componentes, possui um processo que define atividades de Engenharia de Software relacionadas à especificação do contexto e projeto de aplicações sensíveis ao contexto.

Para apoiar o processo proposto, em *CEManTIKA* é definido um metamodelo de contexto independente de domínio baseado no metametamodelo MOF (OMG, 2007), o qual forma as bases para a criação de perfis UML que guiam a modelagem da estrutura de contexto (e.g. fontes de contexto, consumidores de contexto, foco, elemento contextual) e do comportamento sensível a contexto durante as atividades de projeto do da aplicação. Santos (2008) considera que, apesar de o contexto ser um conceito dinâmico construído em tempo de execução quando um foco é identificado (conjunto de elementos contextuais instanciados e que são úteis para a realização de alguma tarefa específica), os elementos contextuais usados para compor o contexto são um conceito estático e podem ser definidos em tempo de projeto.

A abordagem *CEManTIKA* também propõe uma arquitetura genérica para aplicações sensíveis ao contexto, nomeada *Arquitetura de Contexto* (Figura 6.1), a qual é subdivida em camadas que compreendem os componentes de software que manipulam as fontes de contexto (*Context Source*), o gerenciador de contexto (*Context Manager*) e os consumidores de contexto (*Context Consumer*).

Segundo Santos (2008), um uma aplicação sensível ao contexto pode ser vista a partir de duas perspectivas distintas: uma parte que é dependente do domínio (especificação e uso do contexto na aplicação) e outra que é independente de domínio (gerenciamento do contexto). A especificação e uso do contexto estão fortemente ligados à aplicação sendo construída. Diferentes domínios ou diferentes aplicações irão demandar conjuntos de elementos contextuais distintos, o que implicará em diferentes comportamentos sensíveis ao contexto. Por outro lado, o gerenciamento de contexto pode ser modularizado e tradado de uma forma

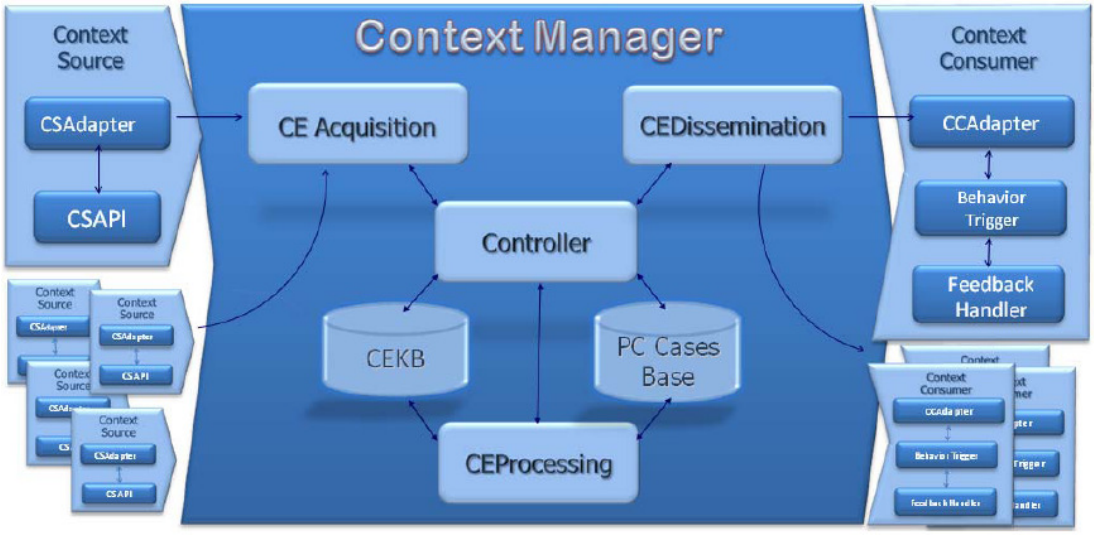


Figura 6.1 – Arquitetura de Contexto (SANTOS, 2008).

independente de domínio, uma vez que engloba os mecanismos de manipulação de contexto (aquisição, processamento e disseminação) que não estão diretamente ligados às funcionalidades de negócio da aplicação.

6.2.2 Semantic Transformer

Semantic Transformer (PATERNÒ et al., 2008) é uma ferramenta utilizada para a transformação automática de páginas Web desenvolvidas originalmente para a plataforma *desktop* em páginas Web adequadas para dispositivos móveis.

Conforme ilustrado na Figura 6.3, essa ferramenta atua como um *Proxy* que detecta as requisições HTTP oriundas de dispositivos móveis e processa a página Web requisitada colocando-a em um formato adequado para visualização no dispositivo móvel. Para isso, no momento da requisição, a ferramenta decompõe a página solicitada em uma hierarquia de modelos (etapas *Reverse* e *Redesign*) que contém suas descrições lógicas usando a linguagem TERESA XML (MORI et al., 2003). A partir desses modelos é gerado o código adaptado (etapa *Pages Generator*) para encaixar o conteúdo da página Web em uma coluna na tela do dispositivo.

6.2.3 XMobile

XMobile (VIANA & ANDRADE, 2008) é um ambiente para a geração de aplicações adaptativas baseadas em formulários para dispositivos móveis, o qual foi

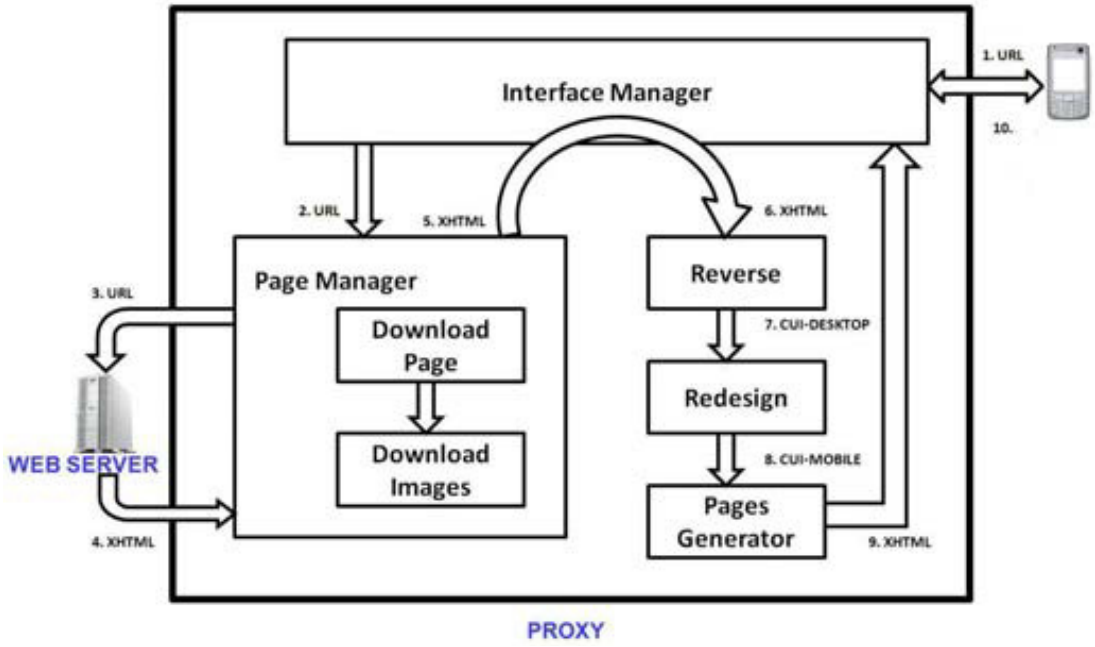


Figura 6.3 – Arquitetura do *Semantic Transformer* (PATERNÒ et al., 2008).

concebido para ser utilizado em processos de desenvolvimento orientados a prototipagem.

Conforme mostrado na Figura 6.2, *XMobile* é composto por um *framework* de componentes de interface denominado *XFormUI* e por uma ferramenta de geração de código chamada *User Interface Generator (UIG)*. O *framework XFormUI* consiste em um *toolkit* de componentes de interface independentes de plataforma de programação que permite a descrição dos diferentes aspectos da interface (e.g. componentes das interfaces, estilos, validação) em uma série de modelos. A ferramenta UIG, por sua vez, mapeia os modelos que descrevem a interface em código executável para três plataformas de programação distintas: J2ME MIDP 1.0; J2ME MIDP 2.0 e *Superwaba*.

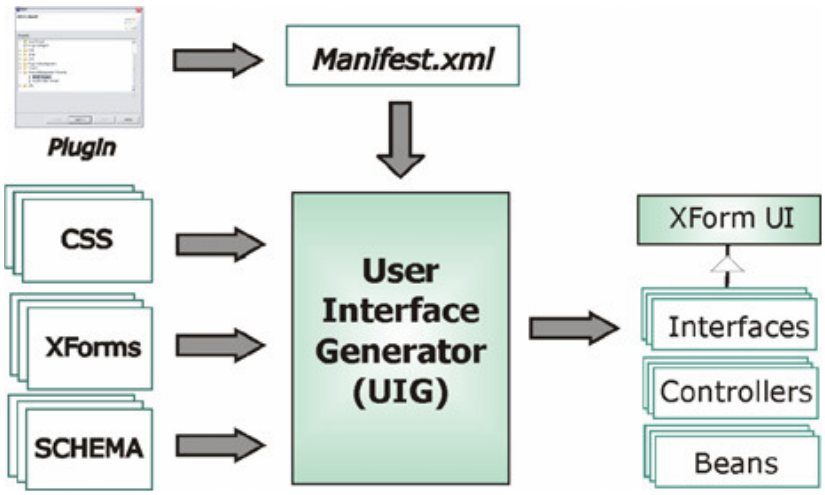


Figura 6.2 – Ambiente *XMobile* (VIANA & ANDRADE, 2008).

XMobile permite que o engenheiro de software descreva os formulários da interface da aplicação utilizando padrões do *World Wide Web Consortium (W3C)*, tais como o *XForms* (W3C, 2009) para definir os componentes; CSS para definir os estilos (e.g. cores e leiaute); e *XML Schema* (W3C, 2004c) para definir as restrições dos campos de entrada de dados do formulário. Além disso, *XMobile* faz uso de um *plugin* no IDE Eclipse, o qual permite que o engenheiro de software crie um arquivo de configuração XML (*Manifest.xml*) para definir a navegabilidade entre as interfaces e as regras de mapeamento para um código executável de uma plataforma específica (J2ME MIDP 1.0, J2ME MIDP 2.0 ou *Superwaba*). O *plugin* do Eclipse dispara a ferramenta UIG para gerar o código da interface, a qual utiliza o *framework XFormUI* para compor os componentes do formulário na interface. Esse processo é ilustrado na Figura 6.4.

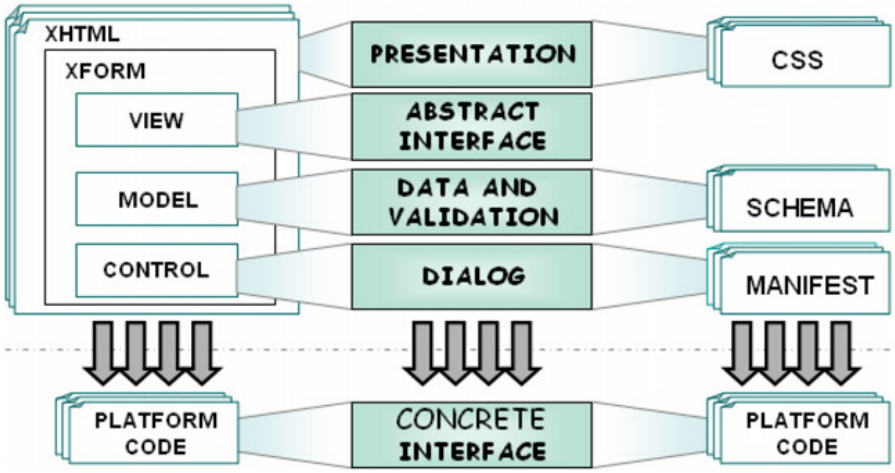


Figura 6.4 – Processo de geração de interfaces a partir de modelos no *XMobile* (VIANA & ANDRADE, 2008).

XMobile emprega a estratégia de adaptação estática, ou seja, diferentes versões da interface para cada plataforma ou dispositivo-alvo são construídas em tempo de desenvolvimento. Isso permite que os desenvolvedores verifiquem a interface gerada pela ferramenta UIG e acrescentem funcionalidades específicas pra cada versão.

6.2.4 EICAF

Extended Internet Content Adaptation Framework (EICAF) (FORTE et al., 2008) é um *framework* de adaptação de conteúdo para aplicações Web. EICAF estendeu o *Internet Content Adaptation Framework (ICAF)* (FORTE et al., 2007) com

a inclusão de componentes que permitem o uso de ontologias e *Web Services* para adaptação de conteúdo.

No *EICAF*, para que as políticas de adaptação sejam aplicadas, as informações contextuais acerca das entidades do domínio considerado (e.g. dispositivo, usuário, acordo de nível de serviço, rede, conteúdo) são disponibilizadas por meio de perfis especificados em *Web Ontology Language (OWL)* (W3C, 2004a). Além disso, as informações sobre os servidores que executam adaptação também são armazenadas em um perfil especificado em OWL, e as informações acerca dos serviços de adaptação disponíveis nesses servidores são armazenadas em um perfil especificado em *Ontology Web Language for Services (OWL-S)* (W3C, 2004b).

Para facilitar a inclusão das informações sobre os serviços de adaptação disponíveis à base de conhecimento do *EICAF*, é disponibilizada uma interface gráfica que permite que o autor do serviço insira informações que o descrevem, tais como o protocolo de comunicação utilizado (e.g. *ICAP - Internet Content Adaptation Protocol*, *SOAP - Simple Object Access Protocol*), o endereço da especificação *Web Service Description Language (WSDL)* do serviço, etc. Essas informações são traduzidas automaticamente em especificações OWL-S ao final da inclusão.

A Figura 6.5 ilustra a sequência de adaptação desempenhada pelo *EICAF*. Os

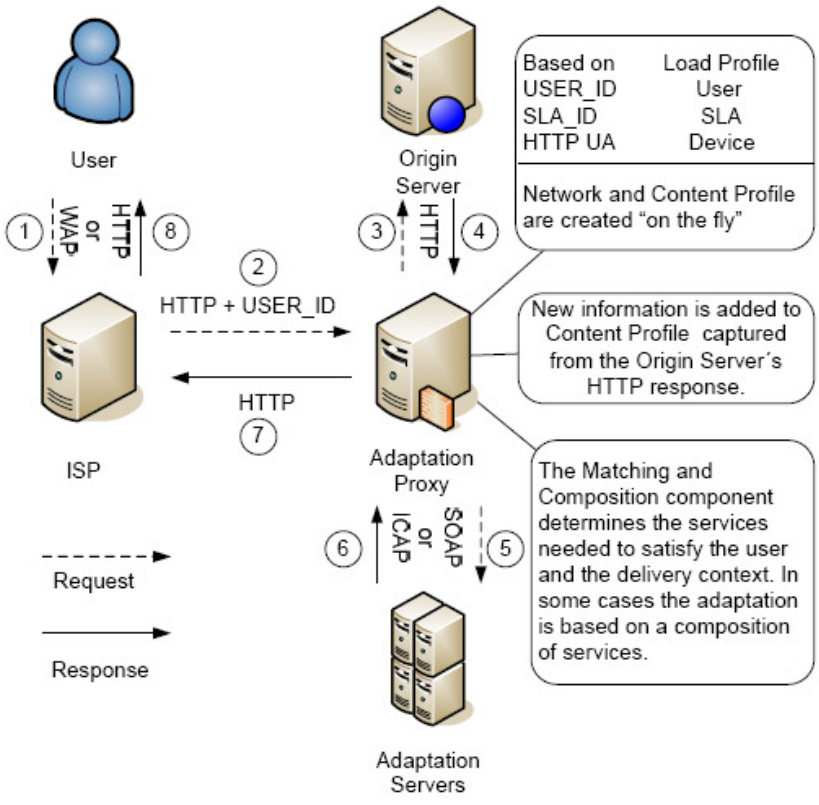


Figura 6.5 – Sequência de adaptação no EICAF (FORTE et al., 2008).

serviços necessários para a execução de uma adaptação são definidos mediante a combinação das informações contextuais contidas nos perfis do dispositivo, do usuário, da rede, dos servidores de adaptação, dos serviços disponíveis, e assim por diante. Se mais que um serviço é necessário para executar a adaptação requerida, a ordem em que os mesmos são executados é gerenciada pelo *EICAF* por meio de orquestração. Se não houver serviços disponíveis o suficiente para executar a adaptação como um todo, a mesma não é realizada.

6.2.5 SCOUT

Semantic COntext-aware Ubiquitous scouT (SCOUT) (VAN WOENSEL et al., 2009a, 2009b) é um *framework* para a construção de aplicações sensíveis ao contexto para dispositivos móveis.

De forma análoga ao projeto *CoolTown* (BARTON & KINDBERG, 2001; DEBATY et al., 2005), o *framework SCOUT* permite mapear entidades do mundo real (e.g. pessoas, lugares, objetos) para entidades virtuais na Web que fornecem informações ou serviços relacionados às entidades físicas a que estão vinculadas, o que é chamado de "presença Web". Dessa forma, por exemplo, quando um usuário móvel se aproxima de uma entidade, ele pode obter o endereço virtual (*URI - Uniform Resource Identifier*) dessa entidade por meio de alguma tecnologia de identificação (e.g. sensores, *RFID - Radio-Frequency Identification*) e acessar sua presença Web para utilizar os serviços fornecidos. As informações e serviços oferecidos são, dessa maneira, específicos da localização atual do usuário, e podem ser ainda mais personalizados através da combinação dos metadados fornecidos pela presença Web selecionada com as informações contextuais acerca do usuário (e.g. preferências, costumes, disponibilidade atual).

No *SCOUT* as presenças Web das entidades são implementadas utilizando tecnologia de Web Semântica, que fornece um grande poder expressivo. A recuperação das informações contextuais é feita de uma forma descentralizada, o que confere ao *framework* maior flexibilidade e escalabilidade, além de permitir que os usuários publiquem suas próprias presenças Web sem ter que lidar com um sistema externo e centralizado para gerenciar o contexto e o conteúdo.

Conforme ilustrado na Figura 6.6, o *framework SCOUT* emprega uma arquitetura em camadas na qual os diferentes aspectos de projeto são separados,

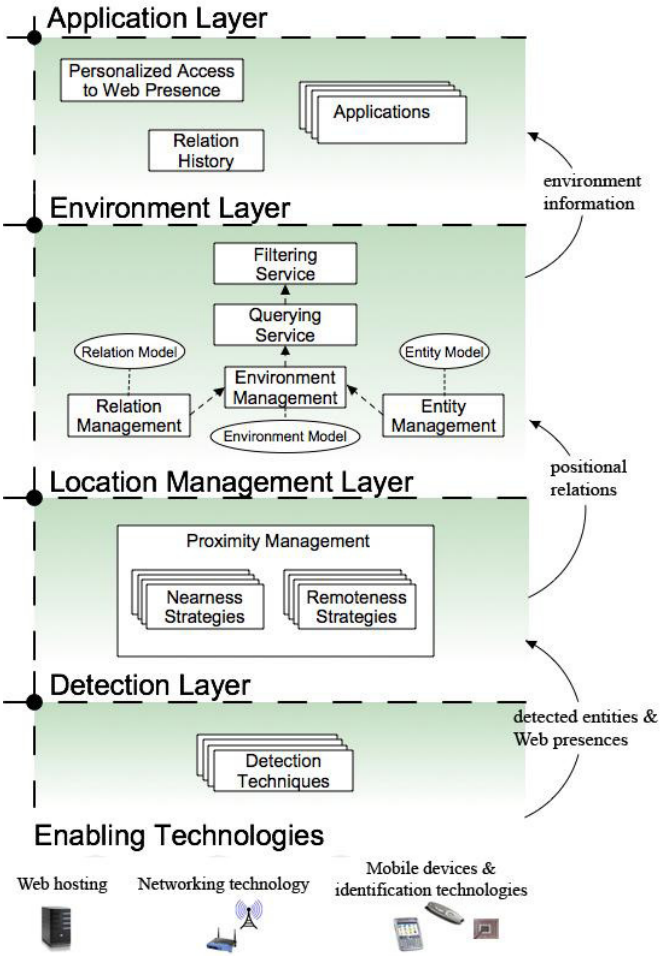


Figura 6.6 – Arquitetura em camadas do *SCOUT* (VAN WOENSEL et al., 2009a, 2009b).

garantindo independência das tecnologias das camadas subjacentes.

A Camada de Detecção (*Detection Layer*) encapsula as funcionalidades de obtenção das referências URI das presenças Web. A Camada de Gerenciamento da Localização (*Location Management Layer*) compreende as estratégias para determinar quais entidades estão próximas e quais deixaram de estar próximas usando as informações fornecidas pela Camada de Detecção. A Camada de Ambiente (*Environment Layer*) abrange modelos, funcionalidades e APIs que permitem o desenvolvimento de aplicações móveis sensíveis ao contexto. Nessa camada, o Modelo de Entidade (*Entity Model*) e o Modelo de Ambiente (*Environment Model*) são representações abstratas que armazenam informações, respectivamente, de determinada entidade e do ambiente no qual o usuário se encontra atualmente com base nas relações espaciais fornecidas pela camada anterior. Com o intuito de tornar as aplicações adaptativas, um Serviço de Filtragem (*Filtering Service*) é empregado para permitir que as aplicações sejam alertadas quando entidades de interesse tornam-se próximas. Por fim, a Camada de Aplicação

(*Application Layer*) abrange as aplicações que utilizam das APIs fornecidas pelo *framework SCOUT* para adaptar/personalizar informações e serviços de acordo com o contexto da interação.

6.2.6 DynamiCOS

O *framework Dynamic Composition of Services (DynamicCOS)* (GONÇALVES DA SILVA et al., 2010) foi criado para dar suporte ao ciclo de vida de composição dinâmica de serviços (Figura 6.7). A ideia da composição dinâmica de serviços é a de que, durante a execução de uma aplicação, serviços podem ser automaticamente descobertos e compostos para satisfazer às necessidades dos usuários à medida que transitam entre diferentes contextos e situações. Na composição dinâmica de serviços assume-se que se não houver um serviço único que satisfaça à requisição do usuário, um novo serviço pode ser automaticamente composto por outros existentes em tempo de execução considerando o contexto e as preferências do usuário a fim de personalizar o processo de criação do serviço demandado.

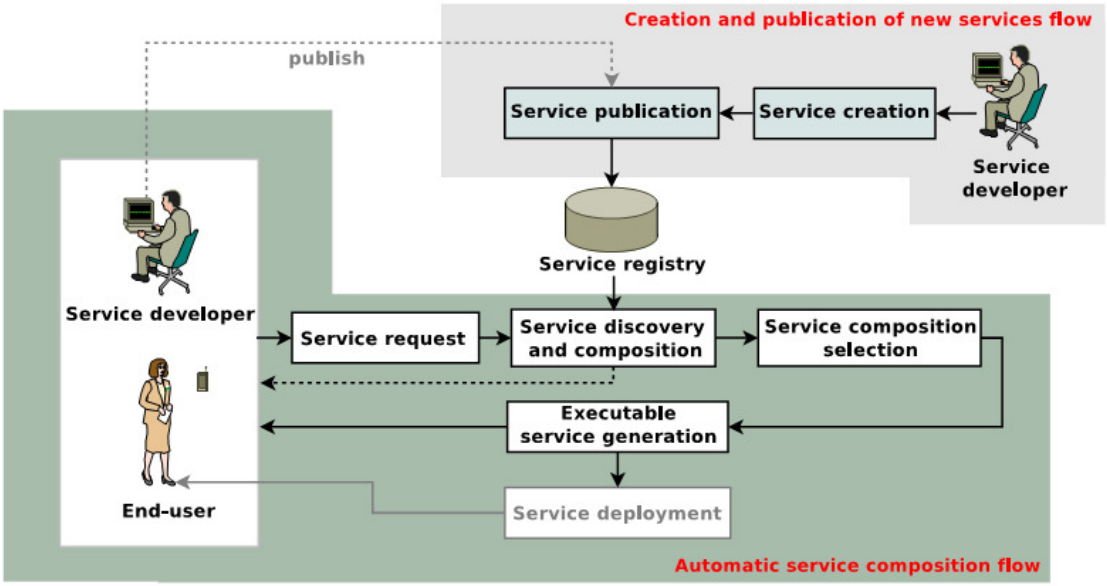


Figura 6.7 – Ciclo de vida da composição dinâmica de serviços (GONÇALVES DA SILVA et al., 2010).

Conforme ilustra a Figura 6.8, para permitir automação às tarefas de descoberta e composição de serviços, no framework *DynamiCOS* são utilizadas informações semânticas, através de ontologias, para descrever os serviços que serão publicados e posteriormente descobertos pelo *framework*. O *framework DynamiCOS* permite que os desenvolvedores publiquem os serviços

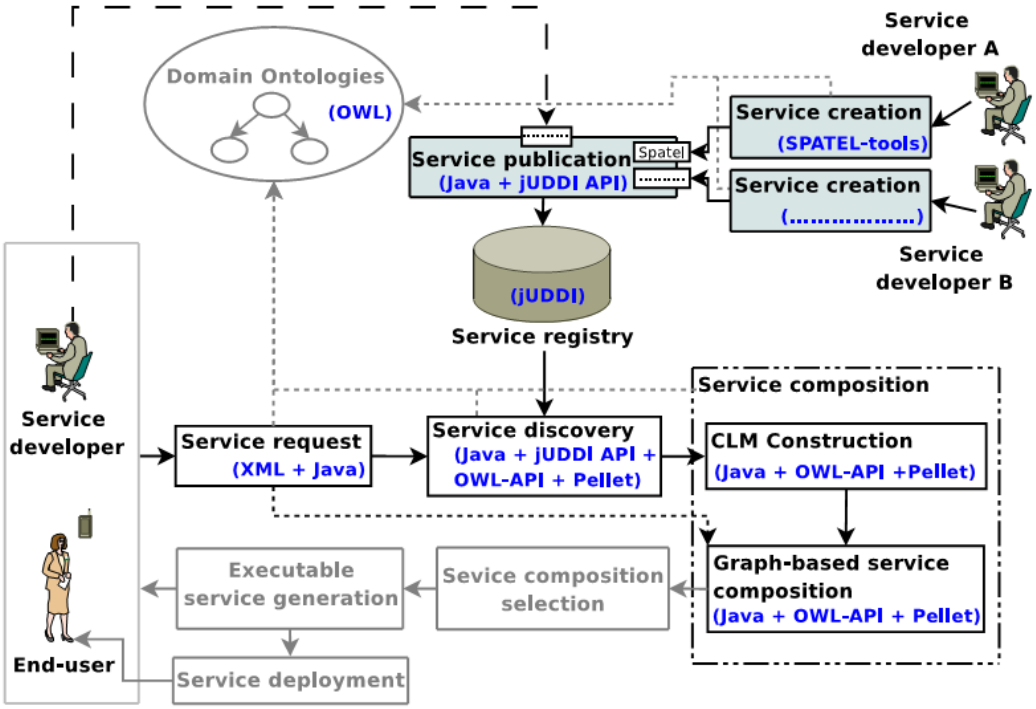


Figura 6.8 – *Framework DynamiCOS* (GONÇALVES DA SILVA et al., 2010).

semanticamente anotados de forma independente da tecnologia pela qual o serviços foi desenvolvido. Isto possibilita que serviços descritos em diferentes linguagens possam ser combinados em uma mesma composição de serviços. Por outro lado, o *DynamiCOS* também fornece suporte aos usuários de serviços, através da descoberta, seleção e composição automática dos serviços com base na requisição do usuário final.

6.3 Análise Comparativa

O trabalho desenvolvido nesta pesquisa se baseia em diversas características dos trabalhos anteriormente descritos. Contudo, apresenta suas próprias contribuições através da evolução e/ou adequação das concepções dos trabalhos correlatos.

Em geral, os trabalhos relacionados diferem-se do trabalho desta pesquisa com relação ao emprego de um processo com atividades e artefatos de apoio ao desenvolvimento de aplicações sensíveis ao contexto; à adoção de uma abordagem de MDD para o desenvolvimento; ao uso do contexto para permitir que as aplicações desenvolvidas desempenhem comportamentos sensíveis ao contexto, como o de

adaptação de conteúdo; à estratégia de adaptação adotada; e ao tipo de aplicação que o trabalho fornece suporte.

CEManTIKA (SANTOS, 2008; VIEIRA et al., 2009), por exemplo, propõe uma abordagem de âmbito genérico com recomendações para construção de aplicações sensíveis ao contexto. O processo *Model Driven RichUbi* desenvolvido neste trabalho, por outro lado, visa a especificamente apoiar a construção de interfaces ricas que se adaptam em múltiplos dispositivos. O processo estabelece o uso de componentes de software pré-fabricados que desempenham a manipulação do contexto (aquisição, processamento e disseminação) e adaptação das interfaces, a fim de contribuir para a simplificação do desenvolvimento. Além disso, por empregar uma abordagem de DSM, o processo proposto prevê o uso de modelos, construídos utilizando uma linguagem específica do domínio de interfaces ricas, os quais possibilitam modelar as interfaces de forma independente de plataforma, bem como utilizar os modelos como entrada para mecanismos de transformação para a geração automática de código para múltiplos dispositivos. Isto contribui para a redução do tempo de desenvolvimento das versões da interface.

A ferramenta *Semantic Transformer* (PATERNÒ; SANTORO; SCORCIA, 2008) realiza a adaptação da interface Web no dispositivo móvel de forma exclusivamente dinâmica. Por outro lado, o *framework UbiCon* adotado no processo proposto emprega uma estratégia de adaptação híbrida, onde apenas partes do código já construído da interface é adaptado dinamicamente. Isso também simplifica o processo de desenvolvimento, já que um número menor de versões mais genéricas podem ser desenvolvidas. Além disso, diferentemente do *Semantic Transformer*, para adaptar a interface, o *framework UbiCon* não apenas verifica a natureza móvel do dispositivo, como também recupera *todo* seu perfil do contexto a fim de obter uma adaptação mais eficiente da interface. O *UbiCon* também oferece a possibilidade de ser estendido de modo que as adaptações possam se basear na combinação dos perfis de outras entidades, tais como o usuário e a rede de acesso, além do dispositivo.

O ambiente *XMobile* (VIANA & ANDRADE, 2008), por sua vez, emprega a estratégia de adaptação puramente estática, ao passo que a estratégia adotada no processo proposto é uma estratégia de adaptação híbrida. Além disso, *XMobile*, apesar de adotar as ideias de MDD, não define explicitamente um roteiro para a

condução de um processo em que esse ambiente pode ser adotado para apoio ao desenvolvimento.

Tanto o *framework EICAF* (FORTE et al., 2008) quanto o *SCOUT* (VAN WOENSEL et al., 2009a, 2009b) e o *framework DynamiCOS* (GONÇALVES DA SILVA et al., 2010) apresentam contribuições para o reúso de software na construção de aplicações ubíquas sensíveis ao contexto. O *framework UbiCon* desenvolvido neste trabalho estende essas concepções adaptando e adequando as mesmas ao desenvolvimento de interfaces ricas adaptativas com emprego de uma abordagem de adaptação híbrida.

A Tabela 6.1 resume as principais características analisadas.

Tabela 6.1 – Análise comparativa da proposta com os trabalhos correlatos.

Características	Trabalhos						
	Proposta	***CEManTIKA***	***Semantic Transformer***	***XMobile***	**EICAF**	**SCOUT**	**DynamiCOS**
Define um processo de apoio ao desenvolvimento de aplicações sensíveis ao contexto.	✓	✓		P			
Emprega abordagem de MDD	✓	P	NA	✓	NA	NA	NA
Utiliza contexto dinamicamente	✓	✓	P		✓	✓	✓
Emprega estratégia de adaptação puramente estática		NA		✓		NA	NA
Emprega estratégia de adaptação puramente dinâmica		NA	✓		✓	NA	NA
Emprega estratégia de adaptação híbrida	✓	NA				NA	NA
Possui componentes de software de apoio à adaptação	✓	NA	✓	✓	✓	NA	NA
Possui componentes de software de apoio ao gerenciamento de contexto	✓		P		✓	✓	✓
Suporta o desenvolvimento de interfaces adaptativas	✓	P	✓	✓	✓	NA	NA
Suporta o desenvolvimento de aplicações Web sensíveis ao contexto	✓	✓	✓		✓	✓	✓
Suporta o desenvolvimento de aplicações *stand-alone* sensíveis ao contexto		✓		✓		✓	✓
Utiliza serviços		NA			✓	✓	✓

Legenda: NA = Não se aplica **P** = Parcialmente ✓ = Apresenta a característica

6.4 Considerações Finais

Este capítulo apresentou os trabalhos associados ao desenvolvimento de aplicações sensíveis ao contexto que possuem correlação ao trabalho desenvolvido nesta pesquisa.

O trabalho proposto é baseado em diversas características dos trabalhos correlatos, apresentando, porém, suas próprias contribuições por meio da adequação e evolução dos trabalhos em que está baseado. Mais precisamente, as contribuições deste trabalho estão voltadas para o suporte ao desenvolvimento de interfaces ricas de aplicações ubíquas que se adaptam em diferentes dispositivos. Para isso, no trabalho são combinadas as concepções de MDD, DSM, sensibilidade ao contexto e adaptação e conteúdo de interfaces para a formulação de um processo apropriado que define atividades e artefatos para apoio ao desenvolvimento.

Capítulo 7

CONCLUSÃO

A Computação Ubíqua tem demandado aos engenheiros de software uma série de requisitos adicionais ao desenvolvimento. Dentre esses requisitos tem-se a necessidade de adaptar as aplicações, tanto seu comportamento quanto seu conteúdo, à heterogeneidade dos dispositivos computacionais do usuário e ao ambiente no qual ele está imerso. A definição de um processo adequado, portanto, torna-se essencial para fornecer aos desenvolvedores o suporte necessário para o cumprimento dos requisitos de adaptação demandados por uma aplicação ubíqua, considerando-se os diferentes contextos em que se executa a aplicação.

Diante disso, neste trabalho foi desenvolvido o processo *Model Driven RichUbi.* Com base nas concepções de Desenvolvimento Dirigido a Modelos (MDD) e Modelagem Específica de Domínio (DSM), o processo define atividades e artefatos que auxiliam na modelagem e geração parcial de código das interfaces ricas para diferentes plataformas. Esses artefatos compreendem um metamodelo do Domínio de Interfaces Ricas que expressa a sintaxe abstrata de uma Linguagem Específica de Domínio (DSL) para apoio à modelagem, e transformações M2C para geração de código. No *Model Driven RichUbi*, também faz-se uso de adaptadores dinâmicos de conteúdo que refinam as versões produzidas das interfaces para se adequarem às peculiaridades do dispositivo de acesso em tempo de execução. Esses artefatos, construídos previamente no processo numa etapa de *Engenharia de Domínio (ED)*, por estarem associados ao Domínio de Interfaces Ricas – um domínio transversal aos domínios de aplicações – podem ser reutilizados durante a *Engenharia de Aplicação (EA)* para simplificar o desenvolvimento das interfaces ricas adaptativas

de aplicações ubíquas de diversas áreas, o que colabora para a redução de esforços e aumento da produtividade.

Este capítulo está organizado da seguinte forma: a Seção 7.1 destaca as contribuições deste trabalho, sintetizando os principais resultados obtidos; a Seção 7.2 apresenta as limitações identificadas em relação ao trabalho desenvolvido. Por fim, na Seção 7.3, são discutidos alguns dos possíveis trabalhos futuros, sinalizando o desfecho desta dissertação.

7.1 Contribuições e Síntese dos Principais Resultados

Uma contribuição importante do *Model Driven RichUbi* é a estratégia híbrida empregada para adaptação do conteúdo das interfaces ricas desenvolvidas. O processo combina geração de código a partir da modelagem em tempo de desenvolvimento (facilitada pelo reúso do metamodelo e das transformações), com a geração de código em tempo de execução (facilitada pelo reúso dos adaptadores de conteúdo). Dessa forma, versões mais genéricas da interface são construídas, cada qual adequada a um determinado grupo de dispositivos (adaptação estática). Os adaptadores de conteúdo, no acesso à aplicação, completam, quando necessário, o restante da adaptação conforme as peculiaridades do dispositivo (adaptação dinâmica). Assim, o desenvolvimento torna-se simplificado, uma vez que um número menor de versões podem ser desenvolvidas.

Para apoio à adaptação híbrida, foi também desenvolvido um *framework,* chamado *Ubiquitous Context Framework (UbiCon)*. O *UbiCon*, que é baseado na *Arquitetura de Contexto* proposta por Santos (2008), encapsula as tarefas de manipulação do contexto e fornece funcionalidades para adaptação do conteúdo de interfaces ricas de forma híbrida. Tais funcionalidades foram desenvolvidas reaproveitando-se os adaptadores de conteúdo produzidos durante a execução da etapa de ED do *Model Driven RichUbi*. A ideia do *UbiCon* é fornecer aos desenvolvedores um “esqueleto” que possa ser instanciado para o desenvolvimento de aplicações ubíquas que adaptam suas interfaces ricas de maneira híbrida através da combinação de diferentes perfis recuperados do contexto (e.g. dispositivo, usuário, rede). O *UbiCon* pode, então, ser empregado na etapa de EA do *Model*

Driven RichUbi de modo a estender a adaptação das interfaces ricas para atender a variados perfis de entidades contextuais usualmente presentes em um ambiente ubíquo.

Para validação do trabalho, dois tipos de experimentos foram conduzidos. Primeiro, foi realizado um estudo de caso preliminar utilizando o processo e o *framework UbiCon* no desenvolvimento das interfaces ricas do módulo Web de uma aplicação ubíqua experimental. O objetivo do estudo de caso foi avaliar a aplicabilidade das atividades da EA e dos artefatos reutilizáveis desenvolvidos na ED para suporte à construção das interfaces ricas adaptativas, bem como verificar o comportamento dos serviços de adaptação fornecidos pelo *UbiCon* e sua capacidade de carga. Em seguida, um experimento foi realizado com o intuito de investigar o impacto do *Model Driven RichUbi* na eficiência de equipes desenvolvendo interfaces ricas adaptativas.

Os resultados obtidos com o estudo de caso mostraram que as atividades da etapa de EA do processo colaboraram para simplificar o desenvolvimento das versões das interfaces. O uso de modelos específicos do Domínio de Interfaces Ricas facilitou a modelagem das interfaces, elevando o nível de abstração e possibilitando gerar grande parte do código de forma automatizada. A adaptação fornecida pelo *framework UbiCon* mostrou-se satisfatória, atendendo às peculiaridades dos dispositivos. Com relação à sua capacidade de carga, apesar do aumento no tempo de resposta da adaptação híbrida em decorrência de uma maior carga de processamento da adaptação por conta de um número maior de acessos simultâneos, houve uma redução da complexidade do desenvolvimento. A adaptação híbrida facilitou o trabalho de autoria de interfaces adaptativas, permitindo atender uma gama maior de dispositivos com um número reduzido de versões das interfaces em vista do refinamento realizado em tempo de execução. Além disso, o refinamento dinâmico das interfaces com o uso do contexto permite obter uma adaptação mais precisa das interfaces, o que também colabora para fornecer aos usuários finais um grau maior de satisfação.

Durante a experimentação do processo, grupos de estudantes executaram a etapa de EA do *Model Driven RichUbi*, enquanto outros grupos seguiram as disciplinas do ciclo de vida clássico, sem aplicar MDD, para a construção das interfaces ricas adaptativas de uma aplicação ubíqua. Os dados coletados durante o estudo referentes ao tempo gasto e ao total de linhas de código implementadas

pelos grupos de ambos os processos serviram de base para análise. Os resultados obtidos, mesmo que em contexto universitário, evidenciaram o potencial do processo proposto em colaborar para o aumento da eficiência de equipes no desenvolvimento de interfaces ricas na Computação Ubíqua em termos de tempo despendido e produtividade. Em geral, os grupos que aplicaram o processo *Model Driven RichUbi* foram mais rápidos e produtivos do que os grupos que não usaram o processo proposto. Os dados evidenciaram que, com o uso do *Model Driven RichUbi*, os esforços de desenvolvimento podem ser significativamente reduzidos sem afetar negativamente a produtividade das equipes. Isto graças ao emprego de modelos específicos do Domínio de Interfaces Ricas, que facilita a tradução dos requisitos da aplicação em componentes de interfaces, e também ao uso de transformações M2C, que encapsulam grande parte das tarefas de codificação das interfaces para diferentes plataformas. Além disso, o reúso *framework UbiCon* possibilitou abstrair as tarefas de aquisição e manipulação do contexto do perfil do dispositivo, permitindo que o foco dos grupos fosse mantido apenas no desenvolvimento das interfaces.

Em síntese, ao explorar as concepções de MDD, DSM e sensibilidade ao contexto no desenvolvimento de interfaces ricas na Computação Ubíqua, o processo definido neste trabalho contribui para a Engenharia de Software nas seguintes áreas:

- **Reúso**. Através dos artefatos construídos na ED, é possível desenvolver mais facilmente e rapidamente as interfaces ricas adaptativas de aplicações de diferentes áreas durante a EA. Pelo fato do Domínio de Interfaces Ricas se tratar de um domínio transversal ao domínio de aplicações, o metamodelo, as transformações M2C e os adaptadores de conteúdo podem ser reutilizados no desenvolvimento das interfaces ricas de aplicações ubíquas em diferentes domínios de problema;
- **Modelagem Específica de Domínio (DSM)**. O metamodelo específico do Domínio de Interfaces Ricas facilita a modelagem das aplicações ubíquas com interfaces ricas. A geração parcial do código colabora para a redução dos esforços de desenvolvimento. A grande vantagem da DSM é o reúso no nível da modelagem, o que possibilita que as interfaces de uma aplicação possam ser implementadas para

diferentes dispositivos da Computação Ubíqua a partir dos mesmos modelos aplicando-se as transformações M2C apropriadas;

- **Sensibilidade ao Contexto**. Durante a ED são desenvolvidos os adaptadores de conteúdo que desempenham a adaptação dinâmica das interfaces conforme o perfil do dispositivo de acesso recuperado do contexto. O reúso desses artefatos na EA possibilita aos desenvolvedores focarem-se mais em tarefas associadas aos requisitos de negócio da aplicação;
- **Adaptação de Conteúdo**. Os adaptadores de conteúdo produzidos na ED foram evoluídos e incorporados ao *framework UbiCon*, de forma a apoiar a adaptação híbrida das interfaces atendendo aos variados perfis de entidades contextuais (e.g. dispositivo, usuário, rede) presentes num ambiente ubíquo;
- **Apoio Computacional.** O metamodelo e as transformações M2C desenvolvidos na ED foram incorporados à ferramenta MVCASE, que por sua vez encontra-se integrada ao IDE Eclipse para apoiar grande parte das atividades dos Engenheiros de Domínio e de Aplicação.

7.2 Limitações

Algumas limitações deste trabalho foram identificadas a partir de uma análise crítica do trabalho desenvolvido.

O processo *Model Driven RichUb*i proposto neste trabalho visa a apoiar o desenvolvimento de interfaces ricas de aplicações ubíquas sensíveis ao contexto. Considerando a ampla abrangência da Computação Ubíqua e da adaptabilidade, o escopo deste trabalho restringiu-se ao domínio de aplicações ubíquas desenvolvidas para a Web 2.0 que adaptam suas interfaces considerando apenas o perfil do dispositivo de acesso recuperado do contexto. Entende-se que elementos de interferem no contexto da interação são inesgotáveis, porém questões como preferências do usuário e largura de banda de comunicação seriam aspectos relevantes ao foco deste trabalho.

Outra limitação deste trabalho refere-se às conclusões obtidas no estudo experimental apresentado no capítulo de avaliação. Apesar de o estudo ter apontado ganhos de eficiência para as equipes que desenvolveram as interfaces ricas usando o *Model Driven RichUbi*, em conformidade com os benefícios que já eram esperados para o processo, deve-se considerar que o fenômeno observado limita-se ao escopo de desenvolvedores de software em ambiente universitário, no qual o experimento foi realizado. Por questões de validade, para estender a amplitude dos resultados obtidos para um contexto mais amplo, é necessário que novos estudos com um número maior de equipes sejam realizados em ambientes *in-vivo*, comparando-se também o uso do *Model Driven RichUbi* em relação a outras abordagens de desenvolvimento, e avaliando outros efeitos, como usabilidade e qualidade das interfaces desenvolvidas.

Na avaliação do *framework UbiCon*, considerou-se o comportamento da adaptação, verificando se a mesma ocorre em conformidade com o perfil do dispositivo identificado do contexto, bem como sua capacidade de carga no servidor, em termos de tempo de resposta, usando emuladores e usuários virtuais simultâneos. Uma outra avaliação mais detalhada, que não foi realizada neste trabalho, se refere ao estudo do desempenho da adaptação híbrida das interfaces utilizando um conjunto maior de aplicações e dispositivos reais em comparação com a adaptação puramente dinâmica.

7.3 Trabalhos Futuros

Ao longo do desenvolvimento deste trabalho, algumas oportunidades de melhoria, tanto do processo quanto do suporte computacional, foram destacadas. Além disso, novas oportunidades de pesquisa foram identificadas. Os possíveis trabalhos futuros envolvem:

- A sistematização dos testes dos artefatos reutilizáveis produzidos na ED de forma a maximizar a correção de defeitos e evitar que estes se propaguem nas aplicações desenvolvidas na EA;
- A inclusão de novos componentes de interface rica ao metamodelo e a construção de um *plugin* com notações gráfica para tornar a modelagem mais abrangente e amigável;

- A extensão do *framework UbiCon* de forma a contemplar outras características contextuais mais dinâmicas acerca do dispositivo durante a adaptação (e.g. estado da bateria, nível de conectividade), além da inclusão de fontes de contexto associadas a outras entidades contextuais, tais como o usuário e a rede de acesso;
- A replicação do estudo experimental com o processo *Model Driven RichUbi* em ambiente industrial para a construção das interfaces ricas adaptativas de aplicações de maior porte, onde outros fatores potencialmente ausentes no ambiente acadêmico poderão ser controlados e estudados. Além disso, outros efeitos também poderão ser estudados, tais como qualidade e usabilidade das interfaces desenvolvidas, por exemplo;
- A realização de testes de usabilidade com usuários em relação às interfaces ricas adaptativas desenvolvidas através do *Model Driven RichUbi*;
- A realização de estudos mais detalhados do desempenho da adaptação híbrida fornecida pelo *framework UbiCon*, de modo a obter subsídios para otimização das funcionalidades de adaptação fornecidas pelo *framework*; e
- A incorporação de um módulo de composição de serviços de adaptação de conteúdo ao *framework UbiCon*, visando à redução dos esforços de implementação das extensões do *framework* através da reutilização de serviços de adaptação fornecidos por terceiros.

Por fim, observou-se que, apesar do foco deste trabalho ter sido mantido no Domínio de Interfaces Ricas, o processo *Model Driven RichUbi* pode ser generalizado para atender a outros domínios. Com exceção da atividade *Construir Adaptadores de Conteúdo dos Componentes de Interface Rica*, intimamente relacionada com o domínio de interfaces, as demais atividade da etapa de Engenharia de Domínio do processo, se generalizadas, constituem-se como atividades para o desenvolvimento de DSLs e transformações que podem ser aplicadas em qualquer domínio de aplicação. A definição de entradas, saídas, mecanismos e controles genéricos para essas atividades, portanto, colaborará para definir um roteiro mais abrangente que apoiará os desenvolvedores na construção de DSLs e transformações M2C para suporte ao desenvolvimento de aplicações com base em MDD e DSM.

PUBLICAÇÕES

A partir da pesquisa apresentada nesta dissertação, foram publicados, em colaboração com diferentes pesquisadores, os seguintes artigos:

CIRILO, C. E., PRADO, A. F., SOUZA, W. L., ZAINA, L. A. M. *A Hybrid Approach for Adapting Web Graphical User Interfaces to Multiple Devices using Information Retrieved from Context*. In: 16th International Conference on Distributed Multimedia Systems - Globalization and Personalization, 2010, Chicago, Illinois, USA. Proceedings of the 16th International Conference on Distributed Multimedia Systems. Skokie, IL, USA: Knowledge Systems Institute Graduate School, 2010. p.168 – 173 (***first place best paper award***).

CIRILO, C. E., PRADO, A. F., SOUZA, W. L., ZAINA, L. A. M. *Model Driven RichUbi - Processo Dirigido a Modelos para a Construção de Interfaces Ricas de Aplicações Ubíquas Sensíveis ao Contexto*. In: XXIV Simpósio Brasileiro de Engenharia de Software - SBES, 2010, Salvador, Bahia, Brasil. Anais do Congresso Brasileiro de Software: Teoria e Prática. , 2010. v.1. p.101 – 110.

CIRILO, C. E., PRADO, A. F., SOUZA, W. L., ZAINA, L. A. M. *Model Driven RichUbi – A Model Driven Process for Building Rich Interfaces of Context-Sensitive Ubiquitous Applications*. In: 28th ACM International Conference on Design of Communication, 2010, São Carlos. Proceedings of the 28th ACM International Conference on Design of Communication. New York: ACM, 2010. p.207 - 214

CIRILO, C. E., PRADO, A. F., ZAINA, L. A. M. *Model Driven RichUbi - Processo Dirigido a Modelos para a Construção de Interfaces Ricas de Aplicações Ubíquas Sensíveis ao Contexto*. In: XV Workshop de Teses e Dissertações em Engenharia de Software - WTES, 2010, Salvador, Bahia, Brasil. Anais do Congresso Brasileiro de Software: Teoria e Prática. , 2010. v.6. p.1 – 6.

CIRILO, C. E.; BELLINI, A.; PRADO, A. F.; ZAINA, L. A. M. *Desenvolvimento de Sistemas Sensíveis ao Contexto usando Web Services*. In: VII Escola Regional de Informática – São Paulo/Oeste. Bauru: SBC; Avalon, 2010, p. 83-112. Minicurso: Livro Texto, 2010. Disponível em: http://www2.fc.unesp.br/erispo2010/MC-01.pdf.

CIRILO, C. E., PRADO, A. F., SOUZA, W. L., ZAINA, L. A. M. *Experimentação do Processo Model Driven RichUbi no Desenvolvimento de Interfaces Ricas Adaptativas*. In: XXV Simpósio Brasileiro de Engenharia de Software - SBES, 2011, São Paulo, Brasil. Anais do II Congresso Brasileiro de Software: Teoria e Prática. , 2011 (em fase de publicação).

SOUZA, W. L., PRADO, A. F., FORTE, M., **CIRILO, C. E.** *Content Adaptation in Ubiquitous Computing*. In: Ubiquitous Computing.1 ed.Rijeka : InTech, 2011, v.1, p. 67-94. Disponível em: http://www.intechopen.com/articles/show/title/content-adaptation-in-ubiquitous-computing.

MENEZES, A. L., **CIRILO, C. E.**, MORAES, J. L. C., SOUZA, W. L., PRADO, A. F. *Using Archetypes and Domain Specific Languages on Development of Ubiquitous*

Applications to Pervasive Healthcare. In: 23rd IEEE International Symposium on Computer-Based Medical Systems (CBMS), 2010, Perth, Austrália. Proceedings of the 23rd IEEE International Symposium on Computer-Based Medical Systems. IEEE, 2010. p.395 – 400.

BELLINI, A., **CIRILO, C. E.**, FERRAZ, V. R. T., DUQUE, J. L., ANNIBAL, L. P., ARAUJO, J. G., DURELLI, R. S., MARCONDES, C. A. C. *A Low Cost Positioning and Visualization System using Smartphones for Emergency Ambulance Service.* In: ACM/IEEE 32nd International Conf. on Software Engineering / 2nd International Workshop on Software Engineering in Health Care, 2010, Cape Town. Proceedings of the 2nd International Workshop on Software Engineering in Health Care. New York, NY, USA: ACM, 2010. p.12 – 18.

BELLINI, A.; **CIRILO, C. E.**; PRADO, A. F.; SOUZA, W. L.; ZAINA, L. A. M. *A Service Layer for building GSM Positioning Systems in e-Health Domain*. In: 4th International Conference on Ubi-media Computing (em fase de publicação).

REFERÊNCIAS

ABI-ANTOUN, M. Making frameworks work: a project retrospective. In: Companion to the 22nd ACM SIGPLAN Conference on Object-Oriented Programming Systems and Applications Companion, 2007. **Proceedings…** ACM, 2007. p. 1004-1018.

ABOWD, G. D.; DEY, A. K.; BROWN, P. J.; DAVIES, N.; SMITH, M.; STEGGLES, P. Towards a better understanding of context and context-awareness. In: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing, 1999, Karlsruhe, Germany. **Proceedings…** Springer-Verlag, LNCS, v. 1707, 1999. p. 304-307.

ABRAMS, M.; PHANOURI, C.; BATONGBACAL, A.; SHUSTER, J. UIML: an appliance-independent XML user interface language. In: Proceedings of the 8th International World Wide Web Conference, 1999, Toronto, Canada. **Proceedings…** Elsevier, 1999. p. 617–630.

AHMED, S.; ASHRAF, G. Model-based user interface engineering with design patterns. **Journal of Systems and Software**, v. 80, n. 8, p. 1408-1422, 2007.

ARAÚJO, R. B. Computação ubíqua: princípios, tecnologias e desafios. In: XXI Simpósio Brasileiro de Redes de Computadores, 2003, Natal, RN, Brazil. **Minicurso: Livro Texto**. Natal, RN: UFRN/DIMAp: UnP, 2003. p. 45-115.

ARES, J.; DIESTE, O.; GARCÍA, R.; LÓPEZ, M.; RODRÍGUEZ, S. Formalising the Software Evaluation Process. In: Proceedings of the XVIII International Conference of the Chilean Computer Science Society, 1998. **Proceedings…** IEEE Computer Society, 1998, 15-24.

BAHNOT, V.; PANISCOTTI, D.; ROMAN, A.; TRASK, B. Using domain-specific modeling to develop software defined radio components and applications. In: Proceedings of the 5th OOPSLA Workshop on Domain-Specific Modeling, 2005, San Diego, USA. **Proceedings…** [S.l.: s.n.] 2005.

BALDAUF, M.; DUSTDAR, S.; ROSENBERG, F. A survey on context-aware systems. **International Journal of Ad Hoc and Ubiquitous Computing**, v. 2, n. 4, p. 263-277, 2007.

BARTON, J.; KINDBERG, T. The cooltown user experience. Hewllet-Packard Laboratories. **Technical Report**, n. 22, 2001. Disponível em: <http://www.hpl.hp.com/techreports/2001/HPL-2001-22.html>.

BASILI, V.; GREEN, S. Software Process Evolution at the SEL. IEEE Software, v. 11, n. 4, p. 58-66, 1994.

BAZIRE, M.; BRÉZILLON, P. Understanding context before using it. In: Proceedings of the 5th International and Interdisciplinary Conference on Modeling and Using Context, 2005, Paris, France. **Proceedings…** Springer-Verlag, LNCS, v. 3554, 2005. p. 29-40.

BELLINI, A.; CIRILO, C. E.; FERRAZ, V. R. T.; ARAUJO, J. G.; DUQUE, J. L.; ANNIBAL, L. P.; DURELLI, R. S.; MARCONDES, C. A. C. A low cost positioning and visualization system using smartphones for emergency ambulance service, In: Proceedings of 2nd ICSE Workshop on Software Engineering in Health Care, 2010, Cape Town, South Africa. **Proceedings…** 2010, p. 12-18.

BITTAR, T. J.; FORTES, R. P.; LOBATO, L. L.; WATANABE, W. M. Web communication and interaction modeling using model-driven development. In: Proceedings of the 27th ACM International Conference on Design of Communication, 2009, Bloomington, Indiana, USA. **Proceedings…** ACM, 2009. p. 193-198.

BLOIS, A. P.; WERNER, C. M. L.; BECKER, K. Towards a components grouping technique within a Domain Engineering process. In: Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications, 2005, Porto, Portugal. **Proceedings…** IEEE Computer Society, 2005, p. 18-25.

BUCHHOLZ, S.; SCHILL, A. Adaptation-aware web caching: caching in the future pervasive web. In: Proceedings of the 13th GI/ITG Conference Kommunikation in verteilten Systemen, 2003, Leipzig, Alemanha. **Proceedings…** 2003, p. 55-66.

BRÉZILLON, P. Context in problem solving: a survey. **The Knkowledge Engineering Review**, v. 18, n. 2, p. 1-40, 1999.

BRÉZILLON, P.; POMEROL, J.-C. Contextual knowledge sharing and cooperation in intelligent assistant systems. **Le Travail Humain**, v. 62, n. 3, p. 223-246, 1999.

BUCHHOLZ, S.; SCHILL, A.; SCHILL, E. Adaptation-aware web caching: caching in the future pervasive web. In: Proceedings of the 13th GI/ITG Conference Kommunikation in verteilten Systemen, Leipzig, Alemanha. **Proceedings…** 2003, p. 55-66.

CHAVARRIAGA, E.; MACÍAS A. A model-driven approach to building modern semantic web-based user interfaces. **Advances in Engineering Software**, v. 40, n. 12, p. 1329-1334, 2009.

CHEN, H. An intelligent broker architecture for pervasive context-aware systems, **Tese de Doutorado**, Universidade de Maryland, Baltimore County, USA, 2004.

CLEAVELAND, J. C. Separating concerns of modeling from artifact generation using XML. In: Proceedings of the 1st OOPSLA Workshop on Domain-Specific Visual Languages, 2001, Tampa Bay, Florida, USA. Proceedings… 2001.

CLEMENTS, P.; NORTHROP, L. **Software Product Lines: Practices and Patterns**. Addison-Wesley, 2002.

COUTAZ, J.; CROWLEY, J. L.; DOBSON, S.; GARLAN, D. Context is key. **Communications of the ACM**, v. 48, n. 3, p. 49-53, 2005.

CZARNECKI, K.; EISENECKER, U. W. **Generative Programming: Methods, Tools, and Applications**. Addison-Wesley Professional, 2000.

DA SILVA, P. P. User interface declarative models and development environments: a survey. In: Proceedings of the 7th International Workshop on the Design, Verification and Specification of Interactive Systems, 2000, Limerick, Ireland. **Proceedings…** Springer-Verlag, LNCS, v. 1946, 2000. p. 207-226.

DEBATY, P.; GODDI, P.; VORBAU, A. Integrating the physical world with the web to enable context-enhanced mobile services. **Mobile Networks and Applications**, v. 10, n. 4, p. 385-394, 2005.

DEITEL, P. J.; DEITEL, H. M. **AJAX, Rich Internet Applications, and Web Development for Programmers**. 1 ed. Prentice Hall, 2008.

DESMET, B.; VALLEJOS, J., CONSTANZA, P.; MEUTER, W.; D'HONDT, T. Context-oriented domain analysis. In: Proceedings of the 6th International and Interdisciplinary Conference on Modeling and Using Context, 2007, Roskilde, Denmark. **Proceedings…** Springer-Verlag, LNCS, v. 4635, 2007. p. 178-191.

DEURSEN, A. v.; KLINT, P. Little languages: little maintenance. **Journal of Software Maintenance**, v. 10, n. 2, p. 75-92, 1998.

DEURSEN, A. v.; KLINT, P.; VISSER, J. Domain-specific languages: an annotated bibliography. **ACM SIGPLAN Notices**, v. 35, n. 6, p. 26-36, 2000.

DEY, A. Understanding and using context. **Personal Ubiquitous Computing**, v. 5, n. 1, p. 4-7, 2001.

DEY, A. K.; SALBER, D.; ABOWD, G. D. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. **Human-Computer Interaction Journal**, v. 16, n. 2, p. 97-166, 2001.

DOURISH, P. **Where the action is: the foundation of embodied interaction**. MIT Press, Cambridge, 2004.

ECLIPSE DOCUMENTATION. **EMFT JET Developer Guide**. Disponível em: <http://help.eclipse.org/help32/index.jsp?topic=/org.eclipse.jet.doc/references/syntax/jetSyntaxXMLTags.xhtml>. Acesso em: 02 fev 2010.

EISENSTEIN, J.; VANDERDONCKT, J.; PUERTA, A. Adapting to mobile contexts with user-interface modeling. In: Proceedings of the 3rd IEEE Workshop on Mobile Computing Systems and Applications, 2000, Monterey, California. **Proceedings…** IEEE Computer Society, 2000. p. 83-92.

FIELDING, R. T.; TAYLOR, R. N. Principled design of the modern Web architecture. In: Proceedings of the 22nd International Conference on Software Engineering, 2000, Limerick, Ireland. **Proceedings…** ACM, 2000. p. 407-416.

FORTE, M.; SOUZA, W. L.; PRADO, A. F. Using ontologies and Web services for content adaptation in Ubiquitous Computing. **Journal of Systems and Software**, v. 81, n. 3, p. 368-381, 2008.

FORTE, M.; CLAUDINO, R. A.; SOUZA, W. L.; PRADO, A. F.; SANTANA, L. H. A component-based framework for the internet content adaptation domain. In: Proceedings of the 22nd Annual ACM Symposium on Applied Computing, 2007, Seoul, Korea. **Proceedings…** ACM, 2007. p. 1450-1455.

FRANCE, R.; RUMPE, B. Model-driven development of complex software: a research roadmap. In: Proceedings of the 29th International Conference on Software Engineering - Future of Software Engineering, 2007, Minneapolis, USA. **Proceedings…** IEEE Computer Society, 2007. p. 37-54.

GAJOS, K.; WELD, D. S. SUPPLE: automatically generating user interfaces. In: Proceedings of the 9th International Conference on Intelligent User Interfaces, 2004, Island of Madeira, Portugal. **Proceedings…** ACM, 2004, p. 93-100.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. M. **Design Patterns: Elements of Reusable Object-Oriented Software**. Addison-Wesley Professional, 1994.

GARLAN, D.; SCHMERL, B. Component-Based Software Engineering in a Pervasive Computing Environment. In: Proceedings of the 4th ICSE Workshop on Component-Based Software Engineering: Component Certification and System Prediction, Toronto, Canada. **Proceedings...** 2001, p. 14-15.

GARTNER. **Gartner Highlights Key Predictions for IT Organisations and Users in 2010 and Beyond**, 2010. Disponível em: http://www.gartner.com/it/page.jsp?id=1278413. Acesso em: 31 jan 2011.

GASPAR, T. C.; YAGUINUMA, C. A.; PRADO, A. F. Desenvolvimento de aplicações colaborativas síncronas na Web 2.0. In: XV Simpósio Brasileiro de Sistemas Multimídia e Web, Fortaleza, CE, Brazil. **Minicurso: Livro Texto**. Fortaleza, CE, 2009. p. 168-207.

GONÇALVES DA SILVA, E.; FERREIRA PIRES, L.; v. SINDEREN, M. v. Towards runtime discovery, selection and composition of semantic services. **Computer Communications**, v. 34, n. 2, p. 159-168, 2011.

GOSSET, W. S. **"Student's" Collected Papers**. Biometrika Office, 1947.

GRISS, M. L.; FAVARO, J.; ALESSANDRO, M. d. Integrating feature modeling with the RSEB. In: Proceedings of the 5th international Conference on Software Reuse, 1998, Victoria, BC, Canada. **Proceedings…** IEEE Computer Society, 1998, p. 76-85.

GRONBACK, R. C. **Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit**. Addison-Wesley, 2009.

GUIZZARDI, G.; PIRES, L. F.; SINDEREN, M. J. V. On the role of domain ontologies in the desing of domain-specific visual modeling languages. In: Proceedings of the 2nd OOPSLA Workshop on Domain-Specific Visual Languages, 2002, Seattle, Washington, USA. **Proceedings…** 2002.

HANSMANN U.; MERK, L.; NICKLOUS, M. S. Pervasive Computing Handbook. Springer-Verlag, 2003.

HOLDENER, A. T. **Ajax: The Definitive Guide**. O'Reilly, 2008.

IEEE, IEEE standard glossary of software engineering terminology. **IEEE Std 610.12-1990**, 1990.

JAVA 2 SDK - Standard Edition Documentation. **Common DOM API**. Disponível em: <http://java.sun.com/j2se/1.4.2/docs/guide/plugin/dom/index.html>. Acesso em: 31 mar 2010.

KASHYAP, V.; SHETH, A. Semantic and schematic similarities between database objects: a context-based approach. **The VLDB Journal**, v. 5, p. 276-304, 1996.

KELLY, S.; TOLVANEN, J. **Domain-Specific Modeling: Enabling Full Code Generation**. Wiley-IEEE Computer Society Press, 2008.

KLEPPE, A. G. A language description is more than a metamodel. In: Proceedings of the 4th International Workshop on Software Language Engineering, 2007, Nashville, EUA. **Proceedings…** Grenoble, France: megaplanet.org, 2007.

KLEPPE, A.; WARMER, J.; BAST, W. **MDA Explained – The Model Driven Architecture: Practice and Promise**. Addison-Wesley, 2003. (Object Technology Series).

KOBRYN, C. Modeling components and frameworks with UML. **Communications of the ACM**, v. 43, n. 10, p 31-38, 2000.

LAFORCADE, P.; ZENDAGUI, B.; BARRÉ, V. A domain-specific-modeling approach to support scenarios-based instructional design. In: Proceedings of the 3rd European Conference on Technology Enhanced Learning: Times of Convergence: Technologies Across Learning Contexts, 2008. **Proceedings…** 2008, p. 185-196.

LIMBOURG, Q.; VANDERDONCKT, J.; MICHOTTE, B.; BOUILLON, L.; LÓPEZ-JAQUERO, V. USIXML: A language supporting multi-path development of user interfaces. In: Proceedings of the 9th IFIP Working Conference on Engineering for Human-Computer Interaction, 2004, Hamburg, Germany. **Proceedings...** Springer-Verlag, LNCS, v. 3425, 2005. p. 200-220.

LINDEN, F. V. D.; SCHMID, K.; ROMMES, E. **Software Product Lines in Action: The Best Industrial Practice In Product Line Engineering**. Springer, 2007.

LUCRÉDIO, D. Uma abordagem orientada a modelos para reutilização de software. **Tese de Doutorado**. Universidade de São Paulo. Instituto de Ciências Matemáticas e de Computação, São Carlos, SP, 2009.

LUCRÉDIO, D.; ALVARO, A.; ALMEIDA, E. S.; PRADO, A. F. MVCASE tool - working with design patterns. In: Proceedings of Latin American Conference on Pattern Languages of Programming 2003, Porto de Galinhas, PE, Brasil. **Proceedings…** 2003.

LUOMA, J.; KELLY, S.; TOLVANEN, J. Defining domain-specific modeling languages: collected experiences, In: Proceedings of the 4th OOPSLA Workshop on Domain-Specific Modeling, 2004, Vancouver, British Columbia, Canada. **Proceedings…** 2004.

MELLOR, S. J; SCOTT, K.; UHL, A.; WEISE, D. **MDA Distilled: Principles of Model-Driven Architecture**. Addison Wesley, 2004.

MELLOR, S. J.; CLARK, A. N.; FUTAGAMI, T. Model-driven development. **IEEE Software**, v. 20, n. 5, p. 14-18, 2003.

MERNIK, M.; HEERING, J.; SLOANE, A. M. When and how to develop domain-specific languages. **ACM Computing Surveys**, v. 37, n. 4, p. 316-344, 2005.

MONTGOMERY, D. C. **Design and Analysis of Experiments**. 5 ed., Wiley, 2000.

MORI, G.; PATERNÒ, F.; SANTORO, C. Tool support for designing nomadic applications. In: Proceedings of the 8th International Conference on Intelligent User Interfaces, 2003, Miami, Florida, USA. **Proceedings…** ACM, 2003. p. 141-148.

NORRIE, M. C. PIM Meets Web 2.0. In: Proceedings of the 27th International Conference on Conceptual Modeling, 2008, Barcelona, Spain. **Proceedings…** Springer-Verlag, LNCS, v. 5231, 2008. p. 15-25.

OMA. **WAG UAProf**. 2001. Disponível em: <http://www.openmobilealliance.org>. Acesso em: 25 mar 2010.

OMG. **MDA Guide Version 1.0.1**. 2003. Disponível em: <http://www.omg.org>. Acesso em: 23 mar 2010.

OMG. **Meta Object Facility Core Specification Version 2.0**. 2006. Disponível em: <http://www.omg.org>. Acesso em: 23 mar 2010.

OMG. **Unified Modeling Language (OMG UML) Infrastructure Version 2.2**. 2009. Disponível em: <http://www.omg.org>. Acesso em: 23 mar 2010.

OMG. **MOF 2.0/XMI Mapping Version 2.1.1**. 2007. Disponível em: <http://www.omg.org>. Acesso em: 23 mar 2010.

O'REILLY, T. **What is Web 2.0: Design patterns and business models for the next generation of software**. 2005. Disponível em: <http://oreilly.com/web2/archive/what-is-web-20.html>. Acesso em: 10 set 2009.

PATERNÒ, F.; SANTORO, C.; SCORCIA, A. Automatically adapting web sites for mobile access through logical descriptions and dynamic analysis of interaction resources. In: Proceedings of the Working Conference on Advanced Visual Interfaces, 2008, Napoli, Italy. **Proceedings…** ACM, 2008. p. 260-267.

PAULK, M. C.; WEBER, C. V.; CURTIS, B., CHRISSIS, M. B. **The Capability Maturity Model: Guidelines for Improving the Software Process. Addison-Wesley Professional**, 1994.

PFLEEGER, S. L. ; ATLEE, J. M. **Software Engineering**. 3 ed., Prentice Hall, 2005.

PRESSMAN, R. S. **Software Engineering: A Practitioner's Approach**. 6 ed., McGraw-Hill Science, 2004.

PUERTA, A. EISENSTEIN, J. Towards a general computational framework for model-based interface development systems. In: Proceedings of the 4th International Conference on Intelligent User interfaces, 1999, Los Angeles, California, USA. **Proceedings…** ACM, 1999. p. 171-178.

PUERTA, A; EISENSTEIN, J. XIML: a common representation for interaction data. In: Proceedings of the 7th International Conference on Intelligent User interfaces, 2002, San Francisco, USA. **Proceedings…** ACM, 2002. p. 214-215.

ROSS, D. T. Structured analysis: a language for communicating ideas. **IEEE Transactions on Software Engineering**, v. 3, n. 1, p. 16-34, 1977.

SADILEK, D. A. Prototyping domain-specific language semantics. In: Proceedings of the Companion to the 23rd ACM SIGPLAN Conference on Object-Oriented Programming Systems Languages and Applications, 2008, Orlando, Florida. **Proceedings…** ACM, 2008. p. 895-896.

SANTOS, V. V. CEManTIKA: A domain-independent framework for designing context-sensitive systems. **Tese de Doutorado**. Universidade Federal de Pernambuco. Centro de Informática, Recife, PE, 2008.

SATO, K. Context-sensitive approach for interactive systems design: modular scenario based methods for context representation. **Journal Physiol. Anthropol. Appl. Human. Sci.**, v. 23, n. 6, p. 277-281, 2004.

SBC. **Grandes Desafios da Pesquisa em Computação no Brasil (2006 - 2016)**. Relatório sobre o Seminário realizado em 8 e 9 de maio de 2006. Disponível em: <http://www.sbc.org.br/index.php?option=com_jdownloads&Itemid=195&task=finish&cid=11&catid=50>. Accesso em: 24 jan 2011.

SCHILIT, B.; THEIMER, M. Disseminating active map information to mobile hosts. **IEEE Network**, v. 8, p. 22–32, 1994.

SCHMIDT, D. C. Guest editor's introduction: Model-driven engineering. **IEEE Computer**, v. 39, n. 2, p. 25-31, 2006.

SERRAL, E.; VALDERAS, P.; PELECHANO, V. Towards the model driven development of context-aware pervasive systems. **Pervasive Mobile Comp.**, v. 6, n. 2, p. 254-280, 2010.

SINGH, G. Guest Editor's Introduction: Content Repurposing. **IEEE MultiMedia**, v. 11, n. 1, p. 20-21, 2004.

SOUZA, W. L.; PRADO, A. F.; FORTE, M.; CIRILO, C. E. Content adaptation in ubiquitous computing. BABKIN, E. (Ed.). In: Ubiquitous Computing. Rijeka: InTech, p. 67-94, 2011.

SPÍNOLA, R. O.; SILVA, J. L. M.; TRAVASSOS, G. H. Checklist to characterize ubiquitous software projects. Anais do XXI Simpósio Brasileiro de Engenharia de Software, João Pessoa, PB. **Anais...** 2007, p. 39-55.

STEINBERG, D.; BUDINSKY, F.; PATERNOSTRO, M.; MERKS, E. **EMF – Eclipse Modeling Framework**, Addison-Wesley, 2008.

SUN Developer Network. **JavaServer Faces Technology**. Disponível em: <http://java.sun.com/>. Acesso em: 23 mar 2010a.

SUN Developer Network. **The Source for Java Developers**. Disponível em: <http://java.sun.com/>. Acesso em: 23 mar 2010b.

TAGSOUP. **TagSoup - Just Keep On Truckin'**. Disponível em: <http://home.ccil.org/~cowan/XML/tagsoup/>. Acesso em: 02 fev 2010.

TRAVASSOS, G. H. ; GUROV, D.; AMARAL, E. A. G. Introdução à Engenharia de Software Experimental. **Relatório Técnico RT-ES-590/02**, Programa de Engenharia de Sistemas e Computação, UFRJ, 2002.

VAN WOENSEL, W.; CASTELEYN, S.; DE TROYER, O. A framework for decentralized, context-aware mobile applications using semantic web technology. In: Proceedings of the Confederated International Workshops and Posters on On the Move to Meaningful Internet Systems, 2009, Vilamoura, Portugal. **Proceedings…** Springer-Verlag, LNCS, v. 5872, 2009a. p. 88-97.

VAN WOENSEL, W.; CASTELEYN, S.; DE TROYER, O. SCOUT: A framework for personalized context-aware mobile applications. In: Proceedings of the Doctoral Consortium of the International Conference on Web Engineering, 2009, San

Sebastian, Spain. **Proceedings…** Disponível on-line em: <http://ceur-ws.org/Vol-484>, 2009b.

VELLIS, G. Model-based development of synchronous collaborative user interfaces. In: Proceedings of the 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems, 2009, Pittsburgh, PA, USA. **Proceedings...** ACM, 2009. p. 309-312.

VIANA, W.; ANDRADE, R. M. C. XMobile: a MB-UID environment for semi-automatic generation of adaptive applications for mobile devices. **Journal of Systems and Software**, v. 81, n. 3, p. 382-394, 2008.

VIEIRA, V.; TEDESCO, P.; SALGADO, A. C. A process for the design of context-sensitive systems. In: Proceedings of the 13th International Conference on Computer Supported Cooperative Work in Design, 2009, Santiago, Chile. **Proceedings…** IEEE Computer Society, 2009. p. 143-148.

VIEIRA, V.; TEDESCO, P.; SALGADO, A. C. Designing context-sensitive systems: an integrated approach. **Expert Systems with Applications: An International Journal**, v. 38, n. 2, p. 1119-1138, 2011.

VÖLTER, M. MD* Best Practices. Dezembro 2008. Disponível on-line em: <http://www.voelter.de/>. Acesso em: 15 mar 2010.

VÖLTER, M; GROHER, I. Product line implementation using aspect-oriented and model-driven software development. In: Proceedings of the 11th International Software Product Line Conference, 2007, Kyoto, Japan. **Proceedings...** IEEE Computer Society, 2007. p. 233-242.

W3C, **XHTML™ 1.0 The Extensible HyperText Markup Language**. 2002. Disponível em: < http://www.w3.org/TR/xhtml1/>. Acesso em: fev. 2010.

W3C. **OWL Web Ontology Language**. 2004a. Disponível em: < http://www.w3.org>. Acesso em: 24 mar 2010.

W3C. OWL-S: **Semantic Markup for Web Services**. 2004b. Disponível em: <http://www.w3.org>. Acesso em: 24 mar 2010.

W3C. **XForms Specification Version 1.1**. 2009. Disponível em: <http://www.w3.org>. Acesso em: 24 mar 2010.

W3C. **XML Schema**. 2004c. Disponível em: < http://www.w3.org>. Acesso em: 24 mar 2010.

WEISER, M. The computer for the 21st century. **ACM SIGMOBILE Mobile Computing and Communications Review** - Special issue dedicated to Mark Weiser, v. 3, n. 3, p. 3-11, 1999.

WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. **Experimentation in Software Engineering: an introduction**. Kluwer Academic Publishers, 2000.

ZAKAS, N.; MCPEAK, J; FAWCETT, J. **Professional AJAX**. Wiley, 2007.

Apêndice A

FORMULÁRIO DE CARACTERIZAÇÃO DO PARTICIPANTE

Este formulário tem por objetivo caracterizar sua experiência acadêmica, pessoal e profissional com relação à Ciência da Computação. Por favor, responda a TODAS as questões o mais fielmente possível. Toda informação fornecida é confidencial, sendo que seu nome e quaisquer outros meios de identificação não serão divulgados em nenhuma hipótese.

1) Dados do participante

Registro acadêmico: _____

Idade: ____

2) Formação acadêmica

( ) Graduação

( ) Mestrado

( ) Doutorado

Ano de Ingresso: _____ Mês/Ano de Conclusão (ou previsão de conclusão): ____/_____

3) Experiência profissional

3.1) Assinale a opção que melhor reflita seu grau atual de experiência com as tecnologias listadas a seguir, considerando a escala de 5 pontos:

0 = nenhum

1 = estudei em aula ou em livro

2 = pratiquei em projetos em sala de aula

3 = usei em projetos pessoais ou na indústria

4 = uso em grande parte dos projetos que realizo

3.1.1)	Linguagem de Programação Java	0	1	2	3	4
3.1.2)	Framework Java Server Faces (JSF)	0	1	2	3	4
3.1.3)	Linguagem de Marcação XHTML/HTML	0	1	2	3	4
3.1.4)	Folhas de Estilo (CSS)	0	1	2	3	4
3.1.5)	Unified Modeling Language (UML)	0	1	2	3	4
3.1.6)	Biblioteca jQuery	0	1	2	3	4
3.1.7)	Banco de Dados MySQL	0	1	2	3	4
3.1.8)	IDE Eclipse	0	1	2	3	4
3.1.9)	IDE NetBeans	0	1	2	3	4
3.1.10)	Wireless Universal Resource File (WURFL)	0	1	2	3	4
3.1.11)	Java Emitter Templates (JET)	0	1	2	3	4
3.1.12)	Eclipse Modeling Framework (EMF)	0	1	2	3	4

3.2) Qual das opções a seguir melhor descreve sua experiência anterior com relação ao desenvolvimento de software na prática?

Tenho desenvolvido software por conta própria (sozinho) ( )	Tenho desenvolvido software como membro de equipe durante cursos que realizo ( )	Tenho desenvolvido software como membro de equipe, na indústria, há menos de 2 anos ( )	Tenho desenvolvido software como membro de equipe, na indústria, há 2 anos ou mais ( )

3.2.1) Por favor, detalhe sua resposta indicando o número de meses de experiência relevantes tanto em projetos desenvolvidos durante algum curso, como em projetos desenvolvidos na indústria.

a) Quantos meses de experiência você teve como gerente de projeto de software:
Meses de experiência: ____ (como aluno em um curso) ____ (na indústria)

b) Quantos meses de experiência você teve como analista de sistemas:
Meses de experiência: ____ (como aluno em um curso) ____ (na indústria)

c) Quantos meses de experiência você teve como desenvolvedor (programação):
Meses de experiência: ____ (como aluno em um curso) ____ (na indústria)

3.2.2) Como você distribuiria o tempo gasto em sua experiência com programação?

_____ % em esforço individual (sozinho)

_____ % em desenvolvimento conjunto com outras pessoas (equipe)

_____ % na supervisão de outros programadores (gerência)

100% ← TOTAL

3.3) Qual sua habilidade em...

a) ... desenvolver software para Web?
( ) Especialista ( ) Avançado ( ) Médio ( ) Básico ( ) Nenhum
Meses de experiência: ____ (como aluno em um curso) ____ (na indústria)

b) ... desenvolver interfaces gráficas para Web?
( ) Especialista ( ) Avançado ( ) Médio ( ) Básico ( ) Nenhum
Meses de experiência: ____ (como aluno em um curso) ____ (na indústria)

c) ... modelar interfaces gráficas para Web?
( ) Especialista ( ) Avançado ( ) Médio ( ) Básico ( ) Nenhum
Meses de experiência: ____ (como aluno em um curso) ____ (na indústria)

4) Experiência com os conceitos e técnicas empregados no experimento

Esta seção será utilizada para compreender sua familiaridade com os conceitos e técnicas que serão utilizados nas atividades do experimento.

4.1) Assinale na tabela a seguir a opção que melhor reflita seu grau atual de experiência com os itens relacionados, considerando a escala de 4 pontos:

0 = Eu não tenho familiaridade com este assunto. Eu nunca fiz isto.
1 = Já li ou estudei sobre isto. Conheço os conceitos e/ou técnicas.
2 = Eu utilizo isto algumas vezes em projetos pessoais ou na indústria, mas não sou um(a) especialista.
3 = Eu sou muito familiar com este assunto. Eu me sentiria confortável fazendo isto.

4.1.1)	Desenvolvimento Dirigido a Modelos (MDD)	0	1	2	3
4.1.2)	Interfaces Ricas para a Web	0	1	2	3
4.1.3)	Aplicações Ubíquas Sensíveis ao Contexto	0	1	2	3

Obrigado por sua colaboração!

Apêndice B

FORMULÁRIO DE CONSENTIMENTO

Experimento

Este experimento visa avaliar a aplicação do processo *Model Driven RichUbi* para apoio à construção de interfaces ricas de aplicações ubíquas sensíveis ao contexto.

Idade

Eu declaro ser maior que 18 (dezoito) anos de idade e concordar em participar do experimento conduzido por Carlos Eduardo Cirilo na Universidade Federal de São Carlos (UFSCar).

Procedimento

Este experimento ocorrerá em uma única sessão, que incluirá o desenvolvimento de interfaces ricas adaptativas de uma aplicação ubíqua para diferentes dispositivos. O desenvolvimento das interfaces se dará com ou sem a aplicação do *processo Model Driven RichUbi*, conforme determinado pelo experimentador. Eu entendo que, uma vez que o experimento tenha sido concluído, os trabalhos que desenvolvi, bem como os dados coletados, serão estudados visando analisar a aplicação dos procedimentos e técnicas propostos.

Confidencialidade

Estou ciente de que toda informação coletada neste experimento é confidencial, e meu nome ou quaisquer outros meios de identificação não serão divulgados. Da mesma forma, me comprometo a não comunicar meus resultados aos demais participantes e/ou a outros grupos enquanto não terminar o experimento, bem como manter sigilo das técnicas e documentos apresentados que fazem parte do experimento.

Benefícios e liberdade de desistência

Eu entendo que os benefícios que receberei deste experimento se limitam ao aprendizado do material que é distribuído e apresentado. Eu compreendo que sou livre para realizar perguntas a qualquer momento, solicitar que qualquer informação relacionada à minha pessoa não seja incluída no experimento, ou comunicar minha desistência de participação. Eu entendo que participo livremente com o único intuito de contribuir para o avanço e desenvolvimento de técnicas e processos para a Engenharia de Software.

Pesquisador responsável

Carlos Eduardo Cirilo
Programa de Pós-Graduação em Ciência da Computação – PPG-CC/DC/UFSCar

Professor responsável

Prof. Dr. Antonio Francisco do Prado
Programa de Pós-Graduação em Ciência da Computação – PPG-CC/DC/UFSCar

Ao preencher e assinar este formulário, dou plena ciência e consentimento com os termos acima expostos.

Nome (em letra de forma): _____ **RA:** _____

Assinatura: _____

Data: ____/_____/_____

Apêndice C

DESCRIÇÃO DA TAREFA – GRUPO DO PROCESSO MODEL DRIVEN RICHUBI

Instruções

Este experimento avaliará a aplicação do processo *Model Driven RichUbi* no desenvolvimento de interfaces ricas adaptativas para aplicações ubíquas. A análise dos resultados será baseada nos dados coletados durante a execução das atividades propostas no experimento. Assim, de forma que o experimentador possa melhor avaliar os resultados obtidos, nos formulários entregues ao seu grupo preencha corretamente todos os dados relacionados ao horário de início e término de cada atividade, bem como os dados relacionados ao número de linhas de código implementadas.

Da mesma maneira, caso encontre algum problema de ordem técnica durante a execução de determinada atividade, como configuração do IDE, configuração de máquina, etc., anote os horários de identificação e solução do problema juntamente com sua descrição no formulário apropriado.

Pergunte e comente tudo o que julgar necessário.

Contextualização

A demanda por software na Computação Ubíqua, onde o acesso às aplicações ocorre de qualquer lugar, a qualquer hora e a partir de diferentes dispositivos, fez surgir novos desafios para a Engenharia de Software. Um desses desafios está relacionado com a adaptação dos conteúdos de uma aplicação para os inúmeros dispositivos que podem acessá-la em diferentes contextos. A construção e manutenção de versões específicas das aplicações para cada tipo de dispositivo torna-se difícil devido à diversidade de dispositivos e plataformas existentes. Apesar disso, é necessário que as aplicações sejam desenvolvidas de modo que a compatibilidade às características dos dispositivos de acesso seja mantida, não prejudicando a interação do usuário com a aplicação.

Neste sentido, o uso de informações extraídas do contexto da interação, como, por exemplo, o perfil do dispositivo de acesso, torna-se essencial para permitir que as aplicações se adaptem de acordo com a característica do ambiente em que operam. Além disso, abordagens de Desenvolvimento Dirigido a Modelos (MDD) podem contribuir para reduzir os esforços na construção de aplicações ubíquas sensíveis ao contexto. O MDD foca na modelagem e na transformação de modelos em artefatos de implementação para diferentes tecnologias com o intuito de prover maior automação e produtividade aos processos de desenvolvimento de software.

Tarefa

Você recebeu a descrição de uma aplicação ubíqua sensível ao contexto denominada *TrackMe*. Uma vez que o foco do processo avaliado está no desenvolvimento da camada de apresentação da aplicação, as regras de negócio da aplicação a ser desenvolvida já encontram-se implementadas. **Sua tarefa é desenvolver as versões da interface Web do *TrackMe* para *smartphones* e *desktops* aplicando a etapa de Engenharia de Aplicação do processo *Model Driven RichUbi*.**

Utilize o formulário de descrição do processo *Model Driven RichUbi* e o material de apoio para auxiliá-lo na execução das atividades. Tendo que os requisitos da aplicação já foram identificados, inicie a execução do processo a partir da atividade *Projetar*. Para cada atividade realizada anote os dados temporais e de linhas de código no formulário apropriado.

Lembre-se que o processo *Model Driven RichUbi* emprega uma estratégia de adaptação de interfaces híbrida, combinando geração de código a partir da modelagem em tempo de desenvolvimento (adaptação estática) com geração de código em tempo de execução (adaptação dinâmica), sendo essas atividades facilitadas pelo reúso de artefatos previamente construídos na etapa de Engenharia de Domínio do processo (metamodelo para apoio à modelagem, transformações para geração de código e adaptadores dinâmicos de conteúdo).

Ao final do experimento, compacte o projeto da aplicação desenvolvida e envie por email para ducirilo@gmail.com, usando como assunto o seguinte padrão: *"GRUPO [Nº de seu grupo] – Experimento"*.

Apêndice D

DESCRIÇÃO DA APLICAÇÃO E MATERIAL DE APOIO – GRUPO DO PROCESSO MODEL DRIVEN RICHUBI

Descrição da Aplicação *TrackMe*

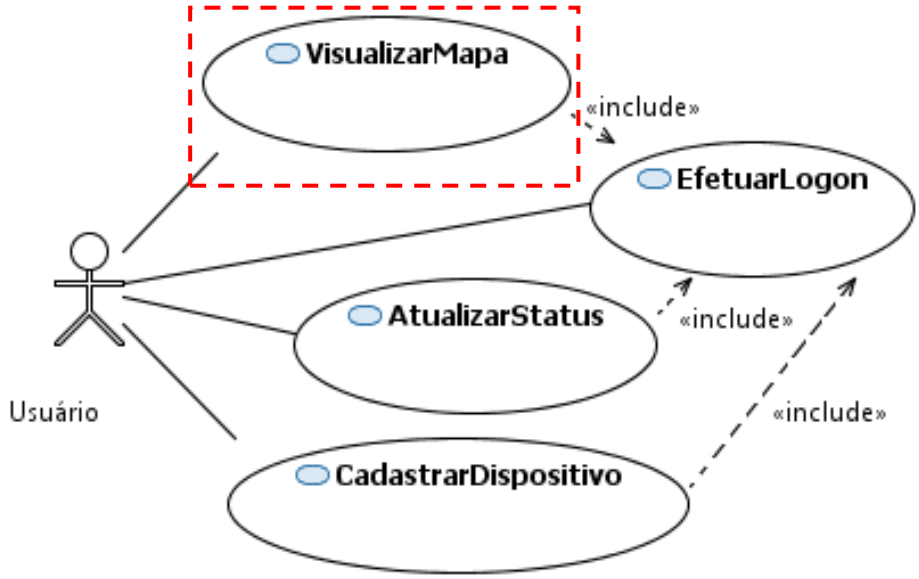
O *TrackMe* é uma aplicação que permite o rastreamento de usuários previamente cadastrados. Esses usuários, munidos de um dispositivo móvel com suporte para *Bluetooth*, ao serem detectados por um *Acess Point (AP)*, têm seu registro armazenado em um banco de dados. Esse registro contém o identificador do dispositivo do usuário, a data e hora em que o registro foi realizado, e o identificador do AP em que o registro foi feito. Cada AP tem sua localização (em coordenadas latitude e longitude) previamente armazenadas no banco de dados, de forma que é possível determinar qual a posição mais recente do usuário na qual o mesmo registrou-se pela última vez.

Cada usuário pode cadastrar na aplicação um ou mais dispositivos de sua propriedade. Para tanto, o usuário necessita efetuar *logon* na aplicação. Além disso, o usuário poderá também indicar seu *status* atual (ocupado, disponível, ausente) a qualquer momento.

Sendo de natureza ubíqua, o *TrackMe* é constituído por três partes: a primeira, executada no dispositivo do usuário, realiza o registro no AP mais próximo; a segunda, executada no servidor, processa os registros dos usuários e os armazena em uma base de dados; e a terceira, também executada no servidor, trata-se de um módulo Web que disponibiliza uma interface para a visualização das posições dos usuários no mapa da localidade em que os APs estão distribuídos. Em curtos períodos de tempo, o módulo Web realiza requisições assíncronas ao servidor, utilizando AJAX, para atualizar o mapa com as posições mais recentes dos usuários.

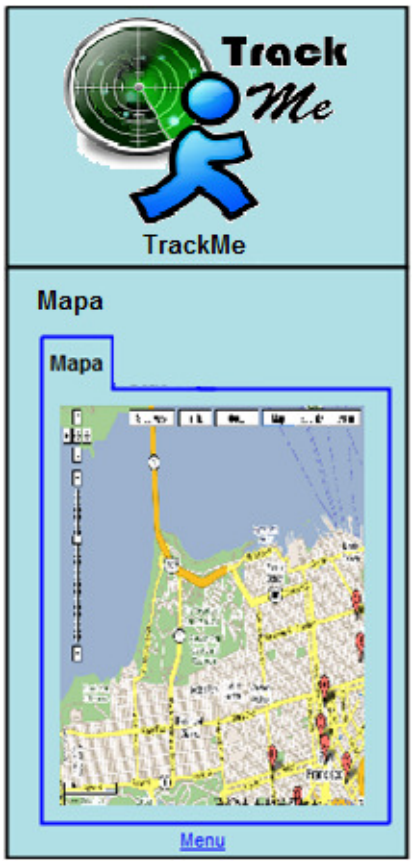
Uma vez que o módulo Web do *TrackMe* pode ser acessado a partir de diferentes dispositivos (*smartphones* e *desktops*), é desejável que suas interfaces sejam adaptadas conforme as características dos dispositivos de acesso, de forma que a interação dos usuários não seja prejudicada.

Alguns dos casos de uso identificados para o *TrackMe* são apresentados no diagrama a seguir. Para atender aos propósitos deste experimento, foque no desenvolvimento das interfaces que atendam ao caso de uso "*VisualizarMapa*".



Material de apoio para implementação das interfaces do *TrackMe*

***Wireframes* das versões *Desktop* e *Smartphone* para atender ao caso de uso "*VisualizarMapa*"**



Importando o projeto no Eclipse

Importe o projeto *TrackMe_Hibrido* que se encontra no diretório que acompanha este material:

1. No IDE Eclipse vá no menu *File* e escolha a opção *Import*...
2. Na janela que se abre, vá à categoria *General* e selecione a opção *Existing Projects Into Workspace*. Pressione o botão *next*.
3. Na próxima janela, clique no botão *Browse* e selecione a pasta raiz do projeto *TrackMe_Hibrido* que acompanha este material.
4. Nessa mesma janela selecione a opção *Copy Projects Into Workspace*.
5. Finalmente, pressione o botão *Finish*. O IDE Eclipse importará o projeto Web de nome *TrackMe_Hibrido* ao *workspace* atual.
6. Corrija quaisquer erros de importação de bibliotecas. Clique com o botão direito sobre a pasta raiz do projeto. Selecione a opção Build *Path > Configure Build Path...* Na janela que se abre remova as bibliotecas conflitantes e adicione novamente (Bibliotecas com possíveis erros: Apache TomCat e JSF).

Criando o banco de dados utilizado pela aplicação

Antes de iniciar a implementação das interfaces do *TrackMe*, crie o banco de dados utilizado pela aplicação:

1. Na plataforma Windows, para acessar o terminal do MySQL e criar o banco, deve-se ir em Iniciar *> Programas > MySQL > MySQL Server 5.x > MySQL Command Line Client*.
2. Com o terminal aberto, deve-se digitar a senha de acesso (*root*) e em seguida colar o *script* SQL que se encontra no arquivo *TrackMe_BD.sql* que acompanha este material.

As regras de negócio para acesso ao banco de dados e recuperação da posição das ambulâncias já encontram-se implementadas nos pacotes de código fonte *br.ufscar.dc.trackme.{beans, db, servlet}.*

Criando o Modelo da Interface

Para criar o modelo da interface rica, utilize o *plugin* editor de modelos construído na etapa de Engenharia de Domínio do processo *Model Driven RichUbi*:

1. Na pasta raiz do projeto *TrackMe_Hibrido* crie uma pasta chamada "*Modelo*" (clique com o botão direito sobre a pasta raiz, opção *New > Folder*)
2. Clique com o botão direito sobre a pasta *Modelo* recém-criada e selecione a opção *New > Other...*
3. Na janela que se abre, selecione a opção *Rich Interface Model* da categoria *MVCASE*. Em seguida clique em *Next*.
4. Na próxima janela, nomeie o modelo com o nome *TrackMe.rich_interface_model*. Clique em *Next*.
5. Na janela seguinte escolha a opção Portal para o campo *Model Object*. Em seguida pressione o botão *Finish*. O modelo será criado, tendo como nó raiz o elemento *Portal*.

Executando as Transformações Modelo para Código

Antes de executar as transformações Modelo para Código, é necessário importar o projeto das mesmas para o seu ambiente de desenvolvimento. Siga os passos do item *"Importando o projeto no Eclipse"* deste documento e importe para seu *workspace* os projetos *br.ufscar.dc.richinterface.m2c.desktop* e *br.ufscar.dc.richinterface.m2c.smartphone* que acompanham este material.

1. Configure o modelo de forma com que as páginas da interface geradas sejam de extensão JSP. Para tanto, clique com o botão direito sobre o elemento *Portal* e selecione a opção *Show Properties View*. Na barra de propriedades do elemento *Portal* selecione a opção *jsp* para a propriedade *Documents Extension*.

   Dica 1: sempre salve todas as alterações no modelo antes de executar as transformações, de forma que essas alterações sejam refletidas no código gerado.

   Dica 2: a fim de verificar a diferença obtida com a adaptação de conteúdo, gere a interface para *desktops* com o logotipo apontando para a imagem *trackme_menor.png*; e a interface para *smartphones* com o logotipo apontando para a imagem *trackme_maior.png*. Essas imagens acompanham este material.
2. Clique com o botão direito sobre o modelo da interface e selecione a opção *Run As > Run Configurations*.
3. Na caixa do lado esquerdo da janela que se abre, dê um clique duplo sobre a opção *JET Transformation*. Do lado direito aparecerão as opções para execução de transformações.
4. No campo *Transformation Input*, clique sobre o botão *Browse...* e selecione o arquivo do modelo da interface.
5. No campo *ID* selecione a transformação que deseja aplicar sobre o modelo (*br.ufscar.dc.richinterface.m2c.desktop* ou *br.ufscar.dc.richinterface.m2c.smartphone*) e em seguida pressione o botão *Run*. A transformação será executada, sendo o código gerado na pasta *WebContent* do projeto.

Incluindo o mapa na interface

Para uso da API do *Google Maps* e das rotinas *JavaScript* que atualizam a posição dos usuários no mapa, insira o seguinte trecho de código na seção *<head>* da página de visualização do mapa:

```
<script type="text/javascript"
src="http://maps.google.com/maps?file=api&amp;v=2&amp;sensor=false&amp;key=ABQIAAAA8bGZtNbInGR-
i63nV1Tb1RSrjhsvlsvZ1QyEYJompjSXtwip6RQ7wz9BwSfh4Upy2T_4I9Utf6BWtA"></script>
<script type="text/javascript" src="js/scripts-googlemaps.js"></script>
<script type="text/javascript">
        $(function() {
                //demais códigos javascript
                initialize();
        });
</script>
```

O mapa deve ser inserido em uma * * cujo atributo *id* possui o valor “map_canvas”, da seguinte forma:

Versão Desktop:

```
<div id="map_canvas" class="map_canvas" align="center" style="height: 210px;"></div>
```

Versão Smartphone:

```
<div id="map_canvas" class="map_canvas" align="center" style="height: 250px"></div>
```

Implementação da adaptação híbrida

A adaptação híbrida é realizada pelos adaptadores de conteúdo implementados previamente na Engenharia de Domínio do processo *Model Driven RichUbi*. Para reutilizar os adaptadores de conteúdo na aplicação, copie o arquivo *jar* do *framework UbiCon* (*ubicon.jar*) que acompanha este material para o diretório *WebContent/WEB-INF/lib*. A base de dados de perfis de dispositivos utilizada pelos adaptadores para desempenharem a adaptação das interfaces é o WURFL. Copie os arquivos *wurfl.zip* e *web_browser_patch.xml* que acompanham este material para o diretório *C:/Users* de seu computador, de forma que os adaptadores de conteúdo funcionem apropriadamente.

A página *index.jsp* do diretório *WebContent* é o ponto de entrada da aplicação. A partir dela é que é realizada a seleção da versão da interface que melhor se encaixa ao perfil do dispositivo recuperado do contexto da interação (adaptação estática) e feito o redirecionamento para tal versão. Essa seleção é realizada reutilizando os adaptadores de conteúdo implementados durante a etapa de Engenharia de Domínio do processo *Model Driven RichUbi*. O código da página *index.jsp* do diretório *WebContent* deverá ficar semelhante ao trecho de código abaixo:

```
<%@page import="br.ufscar.dc.ubicon.contentadaptation.InterfaceID"%>
<%
        InterfaceID interfaceIDObject = new InterfaceID(request);
        String pageToRedirect = interfaceIDObject.getInterfaceID() + "/index.jsp";
%>
<jsp:forward page="<%=pageToRedirect%>"/>
```

Os adaptadores de conteúdo devem ler um arquivo XML que contém as regras que determinam a escolha de determinada versão. Esse arquivo deve ter o nome *context-rules.xml* e estar localizado no diretório *WebContent* da aplicação. No caso do *TrackMe*, a versão para *smartphones* (que deverá estar dentro do diretório *WebContent/Mobile*) será escolhida caso o dispositivo do usuário for um dispositivo móvel. Caso contrário, a versão selecionada deverá ser a versão para desktops (que deverá estar dentro do diretório *WebContent/Desktop*). Logo, o conteúdo do arquivo *context-rules.xml* deverá ser como segue:

```
<?xml version="1.0" encoding="UTF-8"?>
<contextSpecification webApp="TrackMe">
        <context interfaceID="Desktop" dynamicAdaptation="false">
                <contextRule contextualElement="DEVICE_PRODUCT_INFO_IS_MOBILE_DEVICE"
                                matchOperator="eq" matchValue="false" CERelevance="3"/>
        </context>
        <context interfaceID="Mobile" dynamicAdaptation="true">
                <contextRule contextualElement="DEVICE_PRODUCT_INFO_IS_MOBILE_DEVICE"
                                matchOperator="eq" matchValue="true" CERelevance="3"/>
        </context>
</contextSpecification>
```

Uma vez selecionada a versão mais apropriada da interface, os adaptadores de conteúdo refinam a mesma conforme as características particulares do dispositivo no momento da execução (adaptação dinâmica) antes de enviá-la para o usuário. Para a parte da adaptação dinâmica, os

adaptadores de conteúdo implementam o conceito de *Filtros*. Um *Filtro* intercepta dinamicamente requisições e respostas HTTP para transformar ou usar a informação contida nas requisições ou respostas. Para configurar o uso do filtro dos adaptadores de conteúdo no *TrackMe*, deve-se adicionar ao final do arquivo *web.xml* (localizado no diretório *WebContent/WEB-INF)* o seguinte trecho de código destacado:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
        ... demais conteúdos do arquivo web.xml ...
        <filter>
         <display-name>ContentAdapterFilter</display-name>
         <filter-name>ContentAdapterFilter</filter-name>
         <filter-class>br.ufscar.dc.ubicon.contentadaptation.ContentAdapterFilter</filter-class>
        </filter>
        <filter-mapping>
         <filter-name>ContentAdapterFilter</filter-name>
         <url-pattern>/*</url-pattern>
         <dispatcher>REQUEST</dispatcher>
         <dispatcher>INCLUDE</dispatcher>
         <dispatcher>FORWARD</dispatcher>
         <dispatcher>ERROR</dispatcher>
        </filter-mapping>
</web-app>
```

Testando a aplicação

Ao executar a aplicação a partir de um PC, a versão entregue será aquela desenvolvida para desktops. Lembre-se que para a visualização do mapa do *Google Maps* na aplicação é necessária uma conexão com a Internet, uma vez que as definições do mapa são baixadas diretamente dos servidores do Google.

Para simular a execução a partir de um dispositivo móvel, instale no Mozilla Firefox *o plugin User-Agent Switcher FireFox* (http://chrispederick.com/work/user-agent-switcher/). Após instalar o *plugin*, no Mozilla FireFox aparecerá o item *"Default User Agent*" no menu *Ferramentas*. Selecione a opção "*iPhone*" para simular o browser de um *iPhone*.

No IDE Eclipse vá ao meu *Window > Web Browser* e selecione o *FireFox* como browser padrão para execução da aplicação Web. Execute novamente a aplicação Web. Observe agora que a versão entregue é aquela desenvolvida para *smartphones* adaptada conforme as características do *iPhone*.

Apêndice E

FORMULÁRIO DE COLETA DE DADOS – GRUPO DO PROCESSO MODEL DRIVEN RICHUBI

Grupo nº: ____

Anote na tabela abaixo os horários de início e fim de cada atividade realizada.

Atividade	Hora início	Hora Fim
Projetar (Modelagem da Interface)	___:___h	___:___h
Implementar	___:___h	___:___h
Testar	___:___h	___:___h

Anote as quantidades de linha de código geradas automaticamente e manualmente. Para a contagem do código gerado de forma automática, efetue a contagem logo após a aplicação das transformações Modelo para Código. Em seguida, complemente as interfaces até a finalização das mesmas. Realize novamente a contagem. A diferença entre a contagem atual e a contagem anterior será a quantidade de linhas de código implementadas manualmente.

Linhas de Código geradas **automaticamente** pelas transformações*: _____	*Inclua na contagem apenas as linhas de código das páginas Web, arquivos JavaScript e arquivos CSS customizados que não fazem parte de bibliotecas reutilizadas. Não inclua, por exemplo, arquivos da biblioteca jQuery e CSS de temas do jQuery, dentre outros.
Linhas de Código implementadas **manualmente***: _____	

Anote os problemas técnicos encontrados durante a execução do experimento, indicando o horário de identificação e resolução do problema, bem como sua breve descrição.

Descrição do Problema	Hora de Identificação	Hora de Resolução
	___:___h	___:___h

Descrição do Problema	Hora de Identificação	Hora de Resolução
	___:___h	___:___h
	___:___h	___:___h
	___:___h	___:___h
	___:___h	___:___h

Apêndice F

FORMULÁRIO DE AVALIAÇÃO – GRUPO DO PROCESSO MODEL DRIVEN RICHUBI

Grupo nº: ____ RA: _____

1) Você considera que a aplicação do processo Model Driven RichUbi auxiliou no desenvolvimento das interfaces ricas adaptativas? Justifique.	☐ Sim ☐ Não ☐ Parcialmente

2) Você sentiu dificuldades na realização das atividades do processo? Especifique.	☐ Sim ☐ Não ☐ Parcialmente

3) Quais sugestões você teria para melhorar o processo?

4) Você acredita que o uso da linguagem específica de domínio usada para a modelagem das interfaces ricas facilitou o desenvolvimento das interfaces da aplicação? Justifique.	☐ Sim ☐ Não ☐ Parcialmente

5) Você considera viável, sob o ponto de vista de esforço de desenvolvimento e eficiência de adaptação de interfaces do usuário, adotar a estratégia híbrida de adaptação? Justifique.	☐ Sim ☐ Não ☐ Parcialmente

Obrigado por sua colaboração!

Apêndice G

DESCRIÇÃO DA ETAPA DE EA DO PROCESSO MODEL DRIVEN RICHUBI

A etapa de Engenharia de Aplicação do processo *Model Driven RichUbi* é apresentada na Figura F.1.

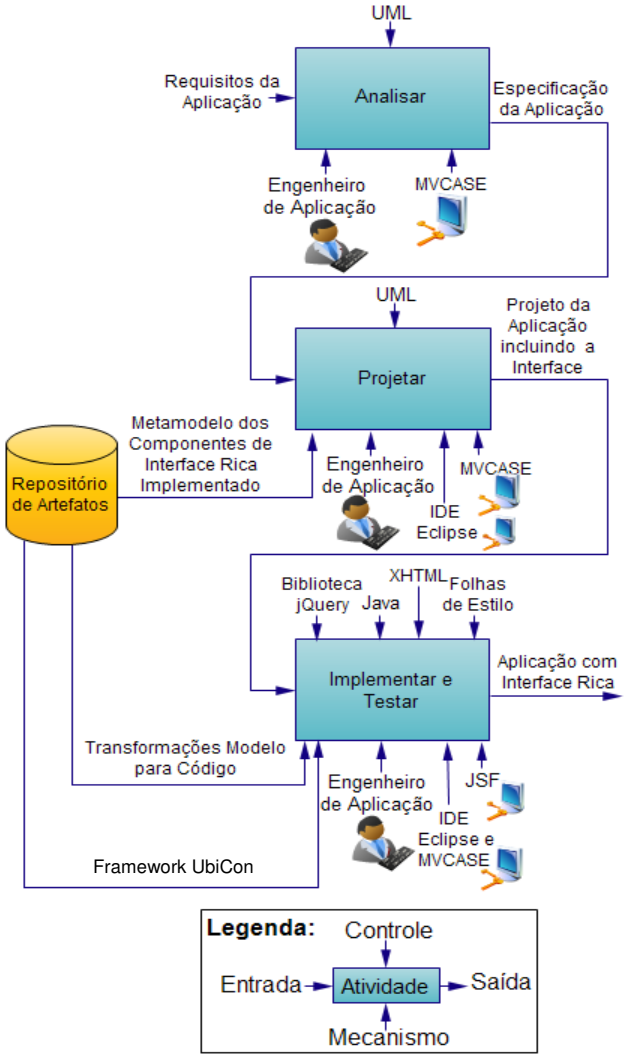


Figura F.1 – Engenharia de Aplicação

Cada uma das atividades são descritas a seguir ilustradas com desenvolvimento do módulo Web, denominado *WebRES,* de uma aplicação ubíqua do domínio de Registros Eletrônicos de Saúde. Essa aplicação permite que médicos cardiologistas acessem os dados pressóricos de seus pacientes utilizando tanto um *desktop* quanto um *smartphone,* e é constituída por três partes: a primeira, que é executada no dispositivo móvel do paciente, registra e persiste os dados pressóricos informados pelo paciente, e os transmite a um servidor; a segunda, que é executada no servidor, recebe os dados e os persiste numa base de dados; a terceira, também executada no servidor, trata-se do módulo Web (*WebRES*) que disponibiliza uma interface para que o profissional de saúde visualize os dados pressóricos de seus pacientes.

Atividade Analisar

A aplicação é especificada a partir de seus requisitos. Essa especificação é realizada através de técnicas UML, como diagramas de classes e diagramas de casos de uso, com auxílio da MVCASE. A MVCASE é uma ferramenta *Computer-Aided Software Engineering (CASE)* de apoio à modelagem e geração de código disponibilizada como um *plugin* do IDE Eclipse. A Figura F.2(a) mostra, por exemplo, um diagrama de casos de uso que especifica os requisitos identificados para o *WebRES*, como os relacionados à autenticação do usuário e à recuperação dos registros pressóricos dos pacientes. A Figura F.2(b) mostra um diagrama de classes que especifica as entidades *Paciente, Médico, Pessoa, Prontuário* e *RegistroPressório* associadas ao *WebRES*.

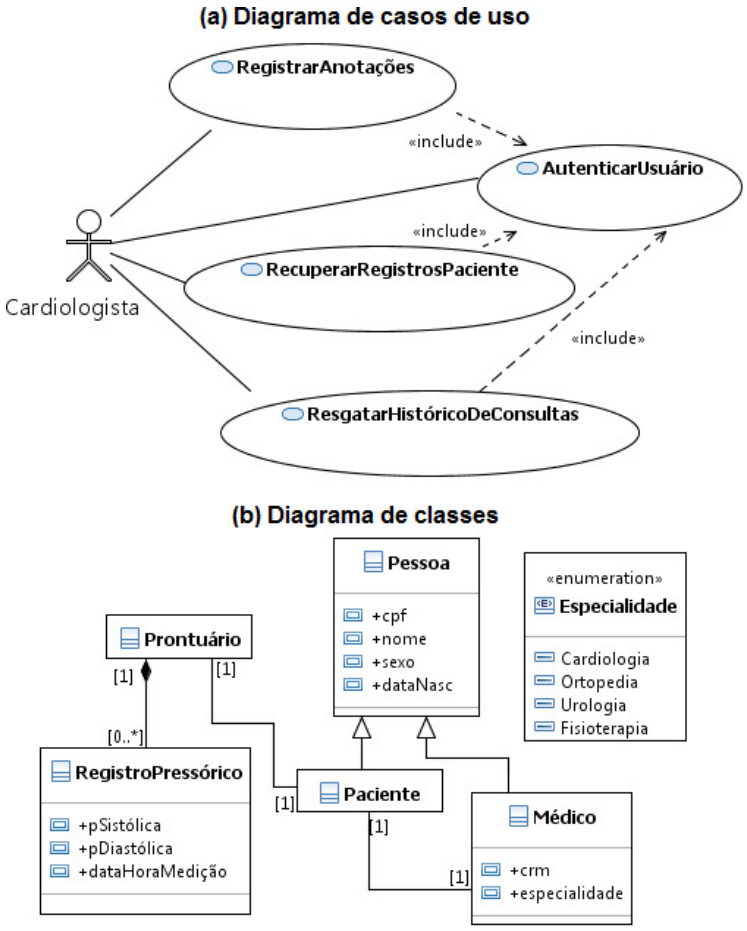


Figura F.2 – Especificação da Aplicação

Atividade Projetar

A especificação da aplicação é refinada com as tecnologias e plataformas de hardware e software que permitem a implementação da aplicação, como, por exemplo, Java EE e o *framework* JSF. Além disso, baseado nos diagramas de casos de uso, o Engenheiro de Aplicação realiza nesta atividade a modelagem das interfaces da aplicação como instância do metamodelo (utilizando o *plugin* editor de modelos disponibilizado no IDE Eclipse). Cada caso de uso da especificação da aplicação é mapeado para componentes de interface que o satisfaçam. Por exemplo, a Figura F.3

apresenta o modelo construído para as interfaces do *WebRES*. É possível observar no modelo a definição de componentes para atender aos casos de uso "*AutenticarUsuário*" e "*RecuperarRegistrosPaciente*" apresentados no diagrama da Figura F.2(a), tais como o formulário para entrada de dados de autenticação numa página de *login*, e o formulário para busca dos registros pressóricos definido dentro de um painel de abas numa página de consultas.

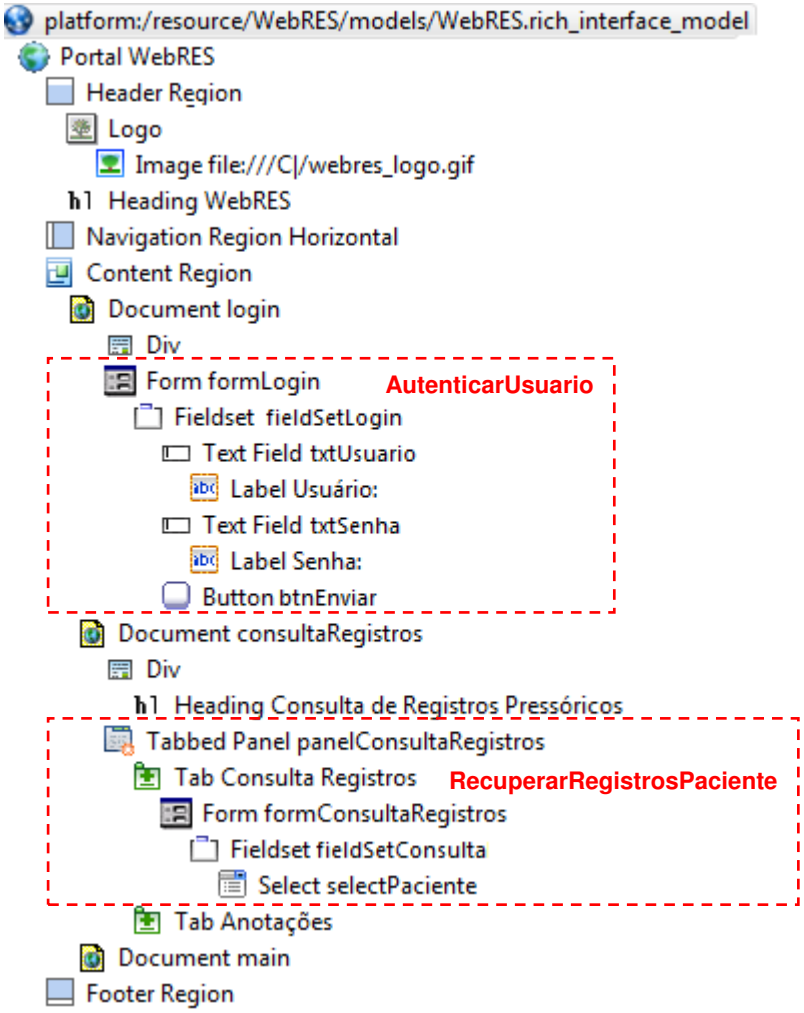


Figura F.3 – Modelo da Interface

Atividade Implementar e Testar

São realizados a codificação e testes das interfaces da aplicação. No IDE Eclipse são executadas as transformações Modelo para Código (*JET Transformations*) sobre os modelos de interface, produzidos na atividade de Projeto, para a geração parcial do código das versões da interface. O código parcial é complementado pelo Engenheiro de Aplicação até a finalização da interface, com a inclusão de características não abrangidas nos modelos, tais como código para recuperação de dados de fontes externas, folhas de estilo personalizadas, métodos de funções *JavaScript* e implementação da lógica de negócios.

Além disso, são incorporados à aplicação os adaptadores de conteúdo (disponíveis através do framework UbiCon) construídos na Engenharia de Domínio para conferir às interfaces uma característica adaptativa.

Apêndice H

DESCRIÇÃO DA TAREFA – GRUPO DO CICLO DE VIDA CLÁSSICO

Instruções

Este experimento avaliará a aplicação do processo *Model Driven RichUbi* no desenvolvimento de interfaces ricas adaptativas para aplicações ubíquas. A análise dos resultados será baseada nos dados coletados durante a execução das atividades propostas no experimento. Assim, de forma que o experimentador possa melhor avaliar os resultados obtidos, nos formulários entregues ao seu grupo preencha corretamente todos os dados relacionados ao horário de início e término de cada atividade, bem como os dados relacionados ao número de linhas de código implementadas.

Da mesma maneira, caso encontre algum problema de ordem técnica durante a execução de determinada atividade, como configuração do IDE, configuração de máquina, etc., anote os horários de identificação e solução do problema juntamente com sua descrição no formulário apropriado.

Pergunte e comente tudo o que julgar necessário.

Contextualização

A demanda por software na Computação Ubíqua, onde o acesso às aplicações ocorre de qualquer lugar, a qualquer hora e a partir de diferentes dispositivos, fez surgir novos desafios para a Engenharia de Software. Um desses desafios está relacionado com a adaptação dos conteúdos de uma aplicação para os inúmeros dispositivos que podem acessá-la em diferentes contextos. A construção e manutenção de versões específicas das aplicações para cada tipo de dispositivo torna-se difícil devido à diversidade de dispositivos e plataformas existentes. Apesar disso, é necessário que as aplicações sejam desenvolvidas de modo que a compatibilidade às características dos dispositivos de acesso seja mantida, não prejudicando a interação do usuário com a aplicação.

Neste sentido, o uso de informações extraídas do contexto da interação, como, por exemplo, o perfil do dispositivo de acesso, torna-se essencial para permitir que as aplicações se adaptem de acordo com a característica do ambiente em que operam.

Dentre as estratégias para adaptação de interfaces do usuário, tem-se a adaptação estática. Nessa estratégia, diferentes versões da interface, cada qual com conteúdos apropriados para um determinado dispositivo, são construídas durante o processo de desenvolvimento (isto é, antes da execução da aplicação). Durante o acesso à aplicação, a versão que melhor se encaixa ao perfil do dispositivo, recuperado do contexto, é selecionada e enviada ao usuário.

Tarefa

Você recebeu a descrição de uma aplicação ubíqua sensível ao contexto denominada *TrackMe*. Uma vez que o foco do processo avaliado está no desenvolvimento da camada de apresentação da aplicação, as regras de negócio da aplicação a ser desenvolvida já encontram-se implementadas. **Sua tarefa é desenvolver as versões especializadas da interface Web do *TrackMe* para *iPhones* e *desktops* seguindo as disciplinas convencionais de Análise, Projeto, Implementação e Testes da Engenharia de Software. Empregue a adaptação estática como estratégia de adaptação das interfaces**.

Utilize o material de apoio para auxiliá-lo durante a construção das versões da interfaces. Tendo que os requisitos da aplicação já foram identificados, inicie a execução do processo a partir da

disciplina *Projeto*. Para cada atividade realizada anote os dados temporais e de linhas de código no formulário apropriado.

Ao final do experimento, compacte o projeto da aplicação desenvolvida e envie por email para ducirilo@gmail.com, usando como assunto o seguinte padrão: *"GRUPO [Nº de seu grupo] – Experimento"*.

Apêndice I

DESCRIÇÃO DA APLICAÇÃO E MATERIAL DE APOIO – GRUPO DO CICLO DE VIDA CLÁSSICO

Descrição da Aplicação *TrackMe*

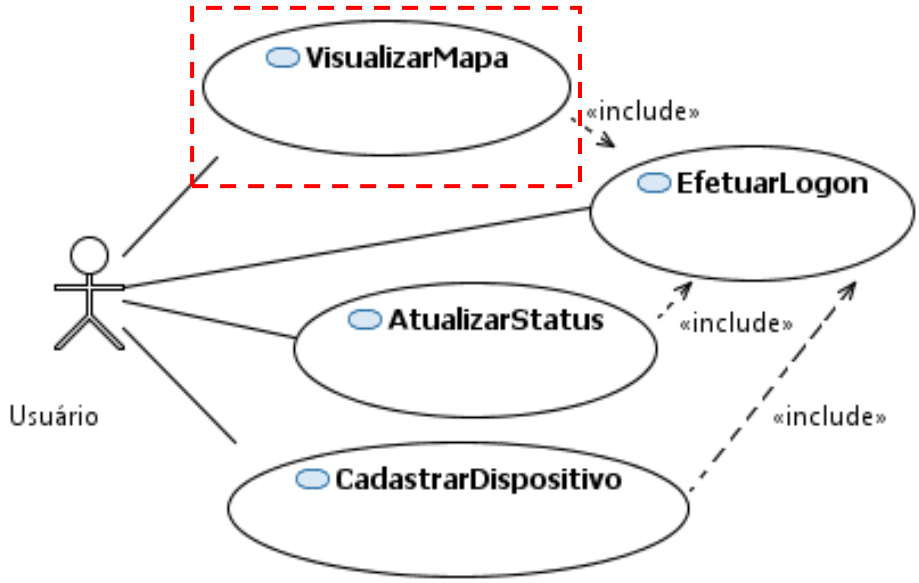
O *TrackMe* é uma aplicação que permite o rastreamento de usuários previamente cadastrados. Esses usuários, munidos de um dispositivo móvel com suporte para *Bluetooth*, ao serem detectados por um *Acess Point (AP)*, têm seu registro armazenado em um banco de dados. Esse registro contém o identificador do dispositivo do usuário, a data e hora em que o registro foi realizado, e o identificador do AP em que o registro foi feito. Cada AP tem sua localização (em coordenadas latitude e longitude) previamente armazenadas no banco de dados, de forma que é possível determinar qual a posição mais recente do usuário na qual o mesmo registrou-se pela última vez.

Cada usuário pode cadastrar na aplicação um ou mais dispositivos de sua propriedade. Para tanto, o usuário necessita efetuar *logon* na aplicação. Além disso, o usuário poderá também indicar seu *status* atual (ocupado, disponível, ausente) a qualquer momento.

Sendo de natureza ubíqua, o *TrackMe* é constituído por três partes: a primeira, executada no dispositivo do usuário, realiza o registro no AP mais próximo; a segunda, executada no servidor, processa os registros dos usuários e os armazena em uma base de dados; e a terceira, também executada no servidor, trata-se de um módulo Web que disponibiliza uma interface para a visualização das posições dos usuários no mapa da localidade em que os APs estão distribuídos. Em curtos períodos de tempo, o módulo Web realiza requisições assíncronas ao servidor, utilizando AJAX, para atualizar o mapa com as posições mais recentes dos usuários.

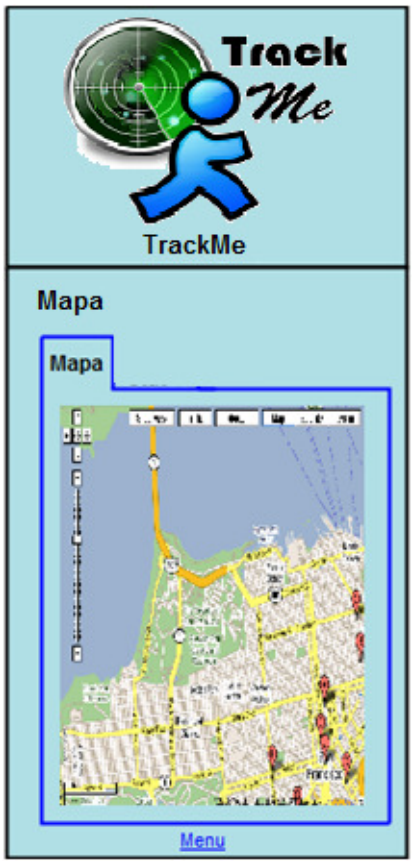
Uma vez que o módulo Web do *TrackMe* pode ser acessado a partir de diferentes dispositivos (*smartphones* e *desktops*), é desejável que suas interfaces sejam adaptadas conforme as características dos dispositivos de acesso, de forma que a interação dos usuários não seja prejudicada.

Alguns dos casos de uso identificados para o *TrackMe* são apresentados no diagrama a seguir. Para atender aos propósitos deste experimento, foque no desenvolvimento das interfaces que atendam ao caso de uso "*VisualizarMapa*".



Material de apoio para implementação das interfaces do TrackMe

***Wireframes* das versões *Desktop* e *Smartphone* para atender ao caso de uso "*VisualizarMapa*"**



Dica 1: as imagens do logotipo para as versões *desktop* e *smartphone* acompanham este material.

Dica 2: para organização do leiaute da interface, conforme ilustrado nos *wireframes*, utilize tabelas e/ou *divs*, conforme julgar mais conveniente.

Dica 3: para a criação do componente painel de abas (*tabbed panel*), utilize a biblioteca *jQuery*. Os arquivos de script, assim como os arquivos CSS dos temas da biblioteca *jQuery* encontram-se disponíveis nos materiais que acompanham este documento.

Importando o projeto no Eclipse

Importe o projeto *TrackMe_Estatico* que se encontra no diretório que acompanha este material:

7. No IDE Eclipse vá no menu *File* e escolha a opção *Import*...
8. Na janela que se abre, vá à categoria *General* e selecione a opção *Existing Projects Into Workspace*. Pressione o botão *next*.
9. Na próxima janela, clique no botão *Browse* e selecione a pasta raiz do projeto *TrackMe_Estatico* que acompanha este material.
10. Nessa mesma janela selecione a opção *Copy Projects Into Workspace*.
11. Finalmente, pressione o botão *Finish*. O IDE Eclipse importará o projeto Web de nome *TrackMe_Estatico* ao *workspace* atual.
12. Corrija quaisquer erros de importação de bibliotecas. Clique com o botão direito sobre a pasta raiz do projeto. Selecione a opção Build *Path > Configure Build Path...* Na janela que se abre remova as bibliotecas conflitantes e adicione novamente (Bibliotecas com possíveis erros: Apache TomCat e JSF).

Criando o banco de dados utilizado pela aplicação

Antes de iniciar a implementação das interfaces do *TrackMe*, crie o banco de dados utilizado pela aplicação:

3. Na plataforma Windows, para acessar o terminal do MySQL e criar o banco, deve-se ir em Iniciar > *Programas* > *MySQL* > *MySQL Server 5.x* > *MySQL Command Line Client*.
4. Com o terminal aberto, deve-se digitar a senha de acesso (*root*) e em seguida colar o *script* SQL que se encontra no arquivo *TrackMe_BD.sql* que acompanha este material.

As regras de negócio para acesso ao banco de dados e recuperação da posição das ambulâncias já encontram-se implementadas nos pacotes de código fonte *br.ufscar.dc.trackme.{beans, db, servlet}.*

Incluindo o mapa na interface

Para uso da API do *Google Maps* e das rotinas *JavaScript* que atualizam a posição dos usuários no mapa, insira o seguinte trecho de código na seção *<head>* da página de visualização do mapa:

```
<script                                                                    type="text/javascript"
src="http://maps.google.com/maps?file=api&amp;v=2&amp;sensor=false&amp;key=ABQIAAAA8bGZtNbInGR
-i63nV1TblRSrjhsvlsvZ1QyEYJompjSXtwip6RQ7wz9BwSfh4Upy2T_4I9Utf6BWtA"></script>
<script type="text/javascript" src="js/scripts-googlemaps.js"></script>
<script type="text/javascript">
        $(function() {
                //demais códigos javascript
                initialize();
        });
</script>
```

O mapa deve ser inserido em uma * * cujo atributo *id* possui o valor “map_canvas”, da seguinte forma:

Versão Desktop:

```
<div id="map_canvas" class="map_canvas" align="center" style="height: 210px;"></div>
```

Versão Smartphone:

```
<div id="map_canvas" class="map_canvas" align="center" style="height: 250px"></div>
```

Implementação da adaptação estática

A adaptação estática é realizada em tempo de desenvolvimento, construindo-se versões especializadas da interface para cada tipo de dispositivo alvo. No caso do *TrackMe* apenas duas versões serão desenvolvidas: uma para *smartphones* (que deverá estar dentro do diretório *WebContent/Mobile*) e outra para *desktops desktops* (que deverá estar dentro do diretório *WebContent/Desktop*).

A página *index.jsp* do diretório *WebContent* é o ponto de entrada da aplicação. Nela é realizada a seleção da versão da interface que melhor se encaixa ao perfil do dispositivo de acesso e feito o redirecionamento para tal versão. A versão para *smartphones* será escolhida caso o dispositivo do usuário for um dispositivo móvel. Caso contrário, a versão selecionada deverá ser a versão para *desktops*. O código da página *index.jsp* do diretório *WebContent* deverá ficar semelhante ao trecho de código abaixo:

```
<%@page import="br.ufscar.dc.trackme.interfaceversion.WURFLAdapter" %>
<%
        WURFLAdapter wurfl = new WURFLAdapter(request);
        String pageToRedirect;
        if (wurfl.getDevice().getCapability("is_wireless_device").equals("false")) {
                pageToRedirect = "Desktop/index.jsp";
        } else {
                pageToRedirect = "Mobile/index.jsp";
        }
%>
<jsp:forward page="<%=pageToRedirect%>"/>
```

Nesse trecho de código, a classe *WURFLAdapter* é utilizada para determinar a escolha da versão da interface conforme o perfil do dispositivo recuperado do contexto da interação. Para tanto, a base de dados XML *Wireless Universal Resource File (WURFL)* é utilizada. O WURFL armazena os perfis, isto é, as características, de milhares de dispositivos de diferentes marcas e modelos (cerca de 13.600). Possui uma API Java própria para seu acesso e manipulação (pode ser baixada pelo site http://wurfl.sourceforge.net/).

A recuperação do perfil do dispositivo a partir do WURFL é realizada com base no campo *User-Agent* da requisição HTTP enviada pelo dispositivo do usuário. A implementação da obtenção do perfil do dispositivo deverá ser feita na classe *WURFLAdapter* localizada no pacote de código fonte *br.ufscar.dc.asps.interfaceversion*. O conteúdo da classe *WURFLAdapter* deverá ser semelhante ao seguinte:

```
package br.ufscar.dc.trackme.interfaceversion;

import java.io.File;
import javax.servlet.http.HttpServletRequest;
import net.sourceforge.wurfl.core.DefaultDeviceProvider;
import net.sourceforge.wurfl.core.DefaultWURFLManager;
import net.sourceforge.wurfl.core.DefaultWURFLService;
import net.sourceforge.wurfl.core.Device;
import net.sourceforge.wurfl.core.DeviceProvider;
import net.sourceforge.wurfl.core.WURFLManager;
import net.sourceforge.wurfl.core.WURFLService;
import net.sourceforge.wurfl.core.cache.NullCacheProvider;
import net.sourceforge.wurfl.core.handlers.matchers.MatcherManager;
import net.sourceforge.wurfl.core.request.DefaultWURFLRequestFactory;
import net.sourceforge.wurfl.core.request.WURFLRequestFactory;
import net.sourceforge.wurfl.core.resource.DefaultWURFLModel;
import net.sourceforge.wurfl.core.resource.WURFLModel;
import net.sourceforge.wurfl.core.resource.WURFLResource;
import net.sourceforge.wurfl.core.resource.WURFLResources;
import net.sourceforge.wurfl.core.resource.XMLResource;

public class WURFLAdapter {
        private String wurflPath = "C:/wurfl.xml";
        private String patchPath = "C:/web_browsers_patch.xml";
        private WURFLManager manager;
        private WURFLResource root;
        private WURFLResources patches;
        private WURFLModel model;
        private MatcherManager matcherManager;
        private DeviceProvider deviceProvider;
        private WURFLService service;
        private WURFLRequestFactory requestFactory;
        private WURFLResource patch;
        private Device deviceProfile;

        public WURFLAdapter (HttpServletRequest request) {
                root = new XMLResource(new File(wurflPath));
                patches = new WURFLResources();
                patch = new XMLResource(new File(patchPath));
            patches.add(patch);
                model = new DefaultWURFLModel(root, patches);
                matcherManager = new MatcherManager(model);
                deviceProvider = new DefaultDeviceProvider(model);
                service    =    new    DefaultWURFLService(matcherManager,deviceProvider,    new
NullCacheProvider());
                requestFactory = new DefaultWURFLRequestFactory();
                manager = new DefaultWURFLManager(service, requestFactory);
                deviceProfile = manager.getDeviceForRequest(request.getHeader("User-Agent"));
        }

        public Device getDevice() {
                return this.deviceProfile;
        }
}
```

Na classe *WURFLAdapter* deve-se apontar o diretório no qual o WURFL e o arquivo de *patch* (que auxilia na detecção do Web Browser do usuário) encontram-se salvos:

```
private String wurflPath = "C:/wurfl.xml";
private String patchPath = "C:/web_browsers_patch.xml";
```

Copie os arquivos *wurfl.xml* e *web_browser_patch.xml* que acompanham este material para o disco *C:* de seu computador ou para o diretório que desejar (lembre-se de indicar corretamente na classe *WURFLAdapter* o caminho real dos arquivos).

Atualizando o contexto das páginas redirecionadas

Para que as rotinas *JavaScript* e as regras CSS externas incorporadas nas páginas Web funcionem corretamente, é necessário indicar o novo contexto do diretório para o qual a aplicação é redirecionada. Para tanto, na seção *<head>* de cada página deve-se incluir o seguinte trecho de código:

```
<base href="<%=request.getRequestURL()%>"/>
```

Testando a aplicação

Ao executar a aplicação a partir de um PC, a versão entregue será aquela desenvolvida para *desktops*. Para simular a execução a partir de um dispositivo móvel, instale no Mozilla Firefox *o plugin User-Agent Switcher FireFox* (http://chrispederick.com/work/user-agent-switcher/).

Após instalar o *plugin*, no Mozilla FireFox aparecerá o item *"Default User Agent*" no menu *Ferramentas*. Selecione a opção "*iPhone*" para simular o browser de um *iPhone*. No IDE Eclipse vá ao meu *Window > Web Browser* e selecione o *FireFox* como browser padrão para execução da aplicação Web. Execute novamente a aplicação Web. Observe agora que a versão entregue é aquela desenvolvida para *smartphones*.

Apêndice J

FORMULÁRIO DE COLETA DE DADOS – GRUPO DO CICLO DE VIDA CLÁSSICO

Grupo nº: ____

Anote na tabela abaixo os horários de início e fim de cada atividade realizada.

Atividade	Hora início	Hora Fim
Projeto (modelagem; decisões de tecnologias, padrões, frameworks...)	___:___h	___:___h
Implementação (codificação)	___:___h	___:___h
Testes	___:___h	___:___h

Anote as quantidades de linha de código geradas automaticamente e manualmente. Considere como linhas de código geradas automaticamente todo código que é criado automaticamente pelo IDE (exemplo: *templates* de páginas Web, *templates* de CSS, etc) ou por qualquer mecanismo de geração automática de código que possa ter sido empregado pelo grupo.

Linhas de Código geradas **automaticamente**[*]: _____	[*]Inclua na contagem apenas as linhas de código das páginas Web, arquivos JavaScript e arquivos CSS customizados que não fazem parte de bibliotecas reutilizadas. Não inclua, por exemplo, arquivos da biblioteca jQuery e CSS de temas do jQuery, dentre outros.
Linhas de Código implementadas **manualmente**[*]: _____	

Anote os problemas técnicos encontrados durante a execução do experimento, indicando o horário de identificação e resolução do problema, bem como sua breve descrição.

Descrição do Problema	Hora de Identificação	Hora de Resolução
	___:___h	___:___h

Descrição do Problema	**Hora de Identificação**	**Hora de Resolução**
	___:___h	___:___h
	___:___h	___:___h
	___:___h	___:___h
	___:___h	___:___h

Apêndice K

FORMULÁRIO DE AVALIAÇÃO – GRUPO DO CICLO DE VIDA CLÁSSICO

Grupo nº: ____ RA: _____

1) Você utilizou algum mecanismo e/ou ferramenta para suporte à geração automática das interfaces? Esse mecanismo/ferramenta atendeu satisfatoriamente suas necessidades?

2) Você sentiu dificuldades na realização das atividades para o desenvolvimento das interfaces ricas adaptativas? Especifique.	☐ Sim ☐ Não ☐ Parcialmente

3) Você considera que um processo que forneça um roteiro e mecanismos de apoio especificamente voltados para o desenvolvimento de interfaces ricas adaptativas facilitaria sua tarefa? Justifique.	☐ Sim ☐ Não ☐ Parcialmente

4) Você acha que é viável, sob o ponto de vista de esforço de desenvolvimento e eficiência de adaptação de interfaces do usuário, construir uma versão especializada da interface para cada tipo de dispositivo existente? Justifique.	☐ Sim ☐ Não ☐ Parcialmente

Obrigado por sua colaboração!

Anexo A

TABELA PADRÃO DE DISTRIBUIÇÃO DE PROBABILIDADE ESTATÍSTICA *T* DE *STUDENT**

Área contida nas duas caudas laterais (bicaudal) da distribuição *t* de *Student*									
***gl*/α**		**Graus de Significância**							
		0,9900	**0,9500**	**0,9000**	**0,1000**	**0,0500**	**0,0275**	**0,0100**	**0,0050**
Graus de Liberdade	**1**	0,0157	0,0787	0,1584	6,3138	12,7062	23,1354	63,6567	127,3213
	2	0,0141	0,0708	0,1421	2,9200	4,3027	5,9051	9,9248	14,0890
	3	0,0136	0,0681	0,1366	2,3534	3,1824	4,0281	5,8409	7,4533
	4	0,0133	0,0667	0,1338	2,1318	2,7764	3,3912	4,6041	5,5976
	5	0,0132	0,0659	0,1322	2,0150	2,5706	3,0790	4,0321	4,7733
	6	0,0131	0,0654	0,1311	1,9432	2,4469	2,8954	3,7074	4,3168
	7	0,0130	0,0650	0,1303	1,8946	2,3646	2,7749	3,4995	4,0293
	8	0,0129	0,0647	0,1297	1,8595	2,3060	**2,6899**	**3,3554**	3,8325
	9	0,0129	0,0645	0,1293	1,8331	2,2622	2,6269	3,2498	3,6897
	10	0,0129	0,0643	0,1289	1,8125	2,2281	2,5782	3,1693	3,5814
	11	0,0128	0,0642	0,1286	1,7959	2,2010	2,5396	3,1058	3,4966
	12	0,0128	0,0640	0,1283	1,7823	2,1788	2,5082	3,0545	3,4284
	13	0,0128	0,0639	0,1281	1,7709	2,1604	2,4821	3,0123	3,3725
	14	0,0128	0,0638	0,1280	1,7613	2,1448	2,4602	2,9768	3,3257
	15	0,0127	0,0638	0,1278	1,7531	2,1314	2,4414	2,9467	3,2860
	16	0,0127	0,0637	0,1277	1,7459	2,1199	2,4252	2,9208	3,2520
	17	0,0127	0,0636	0,1276	1,7396	2,1098	2,4111	2,8982	3,2224
	18	0,0127	0,0636	0,1274	1,7341	2,1009	2,3987	2,8784	3,1966
	19	0,0127	0,0635	0,1274	1,7291	2,0930	2,3877	2,8609	3,1737
	20	0,0127	0,0635	0,1273	1,7247	2,0860	2,3778	2,8453	3,1534
	21	0,0127	0,0635	0,1272	1,7207	2,0796	2,3690	2,8314	3,1352
	22	0,0127	0,0634	0,1271	1,7171	2,0739	2,3610	2,8188	3,1188
	23	0,0127	0,0634	0,1271	1,7139	2,0687	2,3538	2,8073	3,1040
	24	0,0127	0,0634	0,1270	1,7109	2,0639	2,3472	2,7969	3,0905

* *"Student's" Collected Papers* (Ed. Egar S. Pearson e J. Wishart, University College, Londres, 1942).