UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

UMA ABORDAGEM PARA A PERSONALIZAÇÃO AUTOMÁTICA DE INTERFACES DE USUÁRIO PARA DISPOSITIVOS MÓVEIS EM AMBIENTES PERVASIVOS

DISSERTAÇÃO DE MESTRADO

Ricardo Giuliani Martini

Santa Maria, RS, Brasil

2012

UMA ABORDAGEM PARA A PERSONALIZAÇÃO AUTOMÁTICA DE INTERFACES DE USUÁRIO PARA DISPOSITIVOS MÓVEIS EM AMBIENTES PERVASIVOS

por

Ricardo Giuliani Martini

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de

Mestre em Computação

Orientador: Prof. Dr. Giovani Rubert Librelotto (UFSM)

Santa Maria, RS, Brasil

2012

Universidade Federal de Santa Maria
Centro de Tecnologia
Programa de Pós-Graduação em Informática

A Comissão Examinadora, abaixo assinada,
aprova a Dissertação de Mestrado

UMA ABORDAGEM PARA A PERSONALIZAÇÃO AUTOMÁTICA DE INTERFACES DE USUÁRIO PARA DISPOSITIVOS MÓVEIS EM AMBIENTES PERVASIVOS

elaborada por
Ricardo Giuliani Martini

como requisito parcial para obtenção do grau de
Mestre em Computação

COMISSÃO EXAMINADORA:

Giovani Rubert Librelotto (UFSM), Dr.
(Presidente/Orientador)

Adenauer Corrêa Yamin, Prof. Dr. (UFPel)

Giliane Bernardi, Profª. Drª. (UFSM)

Santa Maria, 13 de Abril de 2012.

DEDICATÓRIA

Dedico este trabalho a minha mãe **Nilva Ines Giuliani** e ao meu irmão **Alexandre Martini** que sempre me incentivaram a continuar na vida acadêmica e sempre me ajudaram a enfrentar as dificuldades que apareceram pela frente. Minha mãe e meu irmão, que nunca me deixaram de lado e sempre foram e serão meus melhores amigos. Com os incentivos nos estudos desde pequeno, meu irmão fez com que eu chegasse onde estou hoje. Meu muito obrigado!

AGRADECIMENTOS

Agradeço, em primeiro lugar, a minha família. Minha mãe **Nilva Ines Giuliani** e meu irmão **Alexandre Martini**, os quais me apoiaram durante toda trajetória deste trabalho de mestrado. Sei que muitas vezes fui uma pessoa incomunicável por dias de estresse. Também agradeço a Deus pela força em momentos difíceis.

Ao meu orientador e amigo prof. Dr. **Giovani Rubert Librelotto**, por ter me conduzido neste trabalho de mestrado e muitas vezes me indicado o caminho certo a seguir. Também pelas conversas e idéias trocadas para que este trabalho fosse finalizado com qualidade.

Aos meus amigos e colegas, os quais sempre me incentivaram a continuar em frente e a não desistir nunca dos meus objetivos, além dos trabalhos em conjunto e momentos de descontração. Em especial ao meu grande amigo **Jonas B. Gassen**, que além das conversas e idéias trocadas, me ensinou muito para que eu finalizasse este trabalho da melhor forma possível. Sem tua ajuda, meu amigo, o trabalho seria mais árduo.

“Depois de ter eliminado o impossível,
aquilo que permanece,
ainda que improvável,
deve ser a verdade.”
— SPOCK

RESUMO

Dissertação de Mestrado
Programa de Pós-Graduação em Informática
Universidade Federal de Santa Maria

UMA ABORDAGEM PARA A PERSONALIZAÇÃO AUTOMÁTICA DE INTERFACES DE USUÁRIO PARA DISPOSITIVOS MÓVEIS EM AMBIENTES PERVASIVOS

AUTOR: RICARDO GIULIANI MARTINI
ORIENTADOR: GIOVANI RUBERT LIBRELOTTO (UFSM)
Local da Defesa e Data: Santa Maria, 13 de Abril de 2012.

O grande avanço na indústria de semicondutores possibilitou um aumento no desenvolvimento e comercialização de dispositivos eletrônicos móveis. Juntamente com este mercado, cresceu a necessidade de novos métodos de programação e uma visão diferente para criação de interfaces. Interfaces que antes só eram utilizadas em *desktops* com base de interação teclado e mouse, hoje são utilizadas em diferentes tipos de dispositivos, como celulares, *smartphones* e *tablets*, seja utilizadas em telas sensíveis ao toque como também por comando de voz. Levando em conta estes aspectos de multiplataforma e diferentes usabilidades, torna-se visível a importância de interfaces que se adaptem "ao meio". Com o aparecimento dos dispositivos móveis, a área em questão passou a ser de fundamental importância, pois estes dispositivos possuem características particulares fundamentais para a composição de uma interface satisfatória ao usuário. Os dispositivos móveis estão abrangendo uma diversidade grande de características, o que torna o desenvolvimento de uma interface um processo complexo. Uma das formas de desenvolver e adaptar interfaces de usuário de forma a facilitar o manuseio e diminuir o estresse no momento da utilização do dispositivo é através do uso de perfis de usuários e capacidades de dispositivos, fazendo com que a interface se adapte às necessidades e preferências do usuário e consiga se adaptar totalmente às funcionalidades do dispositivo. Considerando isto, este trabalho tem como objetivo apresentar a arquitetura PIDIM, a fim de ajudar na personalização e adaptação de interfaces de usuário para dispositivos móveis em ambientes pervasivos. As interfaces de usuários adaptadas por este processo da arquitetura PIDIM visam facilitar a utilização de dispositivos móveis. A abordagem proposta apresenta uma arquitetura que utiliza conceitos de Computação Pervasiva possibilitando acesso à informação a qualquer hora, lugar, e dispositivo computacional, além de representar dados relativos ao perfil de usuários, para que a adaptação das interfaces seja totalmente focada no usuário final. A representação do conhecimento sobre o perfil do usuário necessário para a modelagem da arquitetura PIDIM é feita através de ontologias devido a possibilidade de reuso das informações armazenadas. A fim de validar e demonstrar o fluxo de funcionamento da abordagem proposta, é apresentado um estudo de caso, encontrado na literatura, o qual possui como cenário a adaptação de interfaces de usuários quando o mesmo se encontra em movimento.

Palavras-chave: Computação Pervasiva, Interfaces de Usuário, Interface de Usuário Adaptativa, Perfil de Usuário, Independência de Dispositivos, Ontologias, Dispositivos Móveis.

ABSTRACT

Master's Dissertation
Program in Computer Science
Universidade Federal de Santa Maria

AN APPROACH FOR AUTOMATIC CUSTOMIZING USER INTERFACE FOR MOBILE DEVICES IN PERVASIVE ENVIRONMENTS

AUTHOR: RICARDO GIULIANI MARTINI
ADVISOR: GIOVANI RUBERT LIBRELOTTO (UFSM)
Defense Place and Date: Santa Maria, April 13th, 2012.

The great advance in the semiconductor industry allowed a increase in the development and marketing of mobile electronic devices. With the expansion of this market, the need for new programming methods and a different view for the development of user interfaces increased. Interfaces that were used before only in desktops and relied on keyboard and mouse interaction are now used in a variety of devices, including cell phones, smartphones and tablets. Often making the use of touch screens as well as by voice commands. Taking into account these aspects of cross-platform and different usability, it becomes apparent the importance of interfaces that adapt "to the environment." With the advent of mobile devices, this particular area became of fundamental importance because this kind of devices has specifics characteristics that are essential to the composition of a satisfactory user interface. So, mobile devices are covering a large variety of features, which makes the interfaces development a very complex task. One way to develop and adapt user interfaces in order to facilitate handling and to reduce stress at the time of use of the device is through the use of user profiles and capabilities of devices. Therefore, that interface is adapted to the user needs and preferences, as well be able to fully adapt to the device features. Considering this assumption, this dissertation aims to present the architecture PIDIM. This architecture goal to assist in the customization and adaptation of user interfaces for mobile devices in pervasive environments. The user interfaces adapted for this process plans to facilitate the use of mobile devices. The proposed approach presents an architecture that uses concepts of Pervasive Computing enabling information access anytime, anyplace, and in any computing device. Besides, it represents data on the user's profile, so that adaptation of the interfaces is entirely focused on the end user. The knowledge representation about the user profile needed for PIDIM architecture modeling is done through ontologies due to the possibility of reuse of stored information. In order to validate and demonstrate the flow of operation of the proposed approach is presented a case study in the literature, which has as scenario the adaptation of user interfaces when it is in motion.

Keywords: Pervasive Computing, User Interfaces, Adaptive User Interface, User Profile, Device Independence, Ontologies, Mobile Devices.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
MIME	*Multipurpose Internet Mail Extensions*
URI	*Uniform Resource Identifier*
URL	*Uniform Resource Locator*
OWL	*Web Ontology Language*
XML	*eXtensible Markup Language*
RDF	*Resource Description Framework*
RDFS	*Resource Description Framework Schema*
XUL	*XML User Interface Language*
UIML	*User Interface Markup Language*
XAML	*Extensible Application Markup Language*
UsiXML	*User Interface eXtensible Markup Language*
W3C	*World Wide Web Consortium*
IHC	*Interface Humano-Computador*
CC/PP	*Composite Capability / Preference Profiles*
UAProf	*User Agent Profiling Specification*
XHTML	*eXtensible HyperText Markup Language*
WSDL	*Web Service Description Language*
RSS	*RDF Site Summary ou Really Simple Syndication*
HTML	*HyperText Markup Language*
UCD	*User-Centered Design*
WHO	*World Health Organization*
ISO	*International Organization for Standardization*
SWRL	*Semantic Web Rule Language*
SQWRL	*Semantic Query-Enhanced Web Rule Language*
LDIU	*Linguagem Descritiva de Interface de Usuário*
UIDL	*User Interface Description Language*
SDK	*Software Development Kit*
JVM	*Java Virtual Machine*
DVM	*Dalvik Virtual Machine*
IDE	*Interactive Development Environment*

RFID	*Radio Frequency Identification Device*
HTTP	*Hypertext Transfer Protocol*
GUI	*Graphical User Interface*

SUMÁRIO

1 INTRODUÇÃO

Com o passar dos anos, após a invenção das interfaces gráficas de usuário (GUI) para *desktop*, as quais possuíam formas de interação baseadas em *mouse* e teclado e telas com tamanhos padronizados, o *design* não precisava se preocupar muito com adaptação de interface. Com o aparecimento dos dispositivos móveis, a área em questão passou a ser de fundamental importância, pois estes dispositivos possuem características particulares fundamentais para a composição de uma interface satisfatória ao usuário. Os dispositivos móveis estão abrangendo uma diversidade grande de características, o que torna o desenvolvimento de uma interface um processo complexo.

Com o avanço da computação pervasiva, surgem diversas necessidades, tais como a independência de dispositivo, mapeamento do contexto, adaptação de conteúdo, entre outros. Além destas necessidades, é preciso que a computação pervasiva seja invisível aos olhos do usuário final. Para esta invisibilidade ser alcançada, devem existir interfaces de usuário intuitivas, além de poderem ser aplicadas em diferentes contextos.

Um problema comum para os geradores automáticos de interface tem sido que seus projetos não se conformam aos padrões de *design* específicos do domínio que os usuários estão acostumados (GALDO; NIELSEN, 1996).

Na forma de desenvolver interfaces de usuário centradas no sistema, projetistas desenvolvem a interface e o usuário aprende a usá-la, o que pode não ser satisfatório, visto que o usuário muitas vezes não consegue alcançar total adaptação, não conseguindo realizar suas tarefas com bom desempenho. Com o surgimento da computação pervasiva juntamente com a computação móvel, onde o uso de dispositivos computacionais não se prende mais somente nas estações de trabalho, é necessário que a interface se adapte ao perfil do usuário, levando em conta as capacidades do dispositivo, bem como o contexto no qual estão inseridos.

As interfaces gráficas de usuário (GUI), possuem elementos e formas de interação que permitem uma adaptação baseada em usuários individuais. Neste trabalho será apresentada uma abordagem para personalização de interfaces de usuário para dispositivos móveis em ambientes pervasivos. Para alcançar o objetivo da personalização da interface, são utilizados perfis de usuários, onde existem dados relevantes à adaptação da interface, bem como perfis de dispositivos, os quais descrevem as capacidades dos mesmos, além da necessidade de levar em conta o contexto do ambiente.

Para descrever o perfil do usuário utilizaram-se ontologias, onde estão descritos dados como idade, sexo, dados culturais, preferências, deficiências, entre outros. Já para a descrição dos dispositivos utilizou-se um vocabulário chamado UAProf (ALLIANCE, 2001), o qual detalha as capacidades de *hardware* e *software* do dispositivo. O contexto do ambiente é descrito através do uso de sensores, como sensores de temperatura, localização, ruídos, etc.

Com a união destes dados, é possível personalizar a interface final do usuário conforme seja mais oportuno pelo mesmo. O principal desafio deste trabalho é fazer com que o usuário final, tenha ele o perfil que tiver, consiga desenvolver suas tarefas em seu dispositivo móvel da melhor forma possível, com melhor desempenho, e, principalmente, com maior satisfação. Para isso, a abordagem proposta neste trabalho utiliza a computação sensível ao contexto de forma a capturar dados relevantes ao ambiente pervasivo em que o usuário se encontra através de sensores e alcançar o objetivo do trabalho.

A proposta em questão de criar uma arquitetura para personalização automática de interfaces de usuário em um ambiente pervasivo veio através da necessidade de sistemas interativos com o usuário serem projetados conforme seja mais oportuno para o usuário.

A fim de introduzir a abordagem proposta, esta dissertação está organizada da seguinte forma. O Capítulo 2 conceitua e mostra vantagens e características da computação pervasiva, bem como descreve a computação sensível ao contexto, necessária para este trabalho. Ainda neste capítulo é apresentado como a independência de dispositivo e a computação móvel são importantes.

No Capítulo 3 são detalhados todos os dados relevantes para a personalização das interfaces de usuário baseados no perfil do usuário e perfil de dispositivos móveis. Neste capítulo também são apresentadas algumas linguagens descritivas de interface de usuário (LDIUs) baseadas em XML. Uma comparação entre estas linguagens também é feita, a fim de escolher qual usar para testes realizados na corrente dissertação. Além das LDIUs, este capítulo apresenta o SDK (*Software Development Kit*) do Android, detalhando como são criadas as interfaces de usuário nesta ferramenta, e também como utilizar certos recursos.

No Capítulo 4 são apresentados conceitos e características, bem como formas de representação de ontologias. Também são expostos neste capítulo os elementos que compõem uma ontologia e as vantagens de utilizá-la. Além disso, é descrita uma ferramenta para o processamento de ontologias, como são realizadas as inferências e consultas, e para que servem.

O Capítulo 5 apresenta a construção de uma ontologia para representação do conhecimento

do domínio de perfil do usuário, descrevendo classes e dados relativos ao perfil do usuário como preferências, deficiências, dados demográficos (idade, sexo, e nacionalidade), experiência com computadores, entre outros.

A arquitetura para personalização automática de interfaces de usuário para dispositivos móveis em ambientes pervasivos é apresentada no Capítulo 6, onde estão descritos cada elemento e fluxo de comunicação entre os mesmos. Com o intuito de validar a abordagem proposta, no Capítulo 7, é apresentado um caso de estudo já abordado na literatura, mostrando como interfaces de usuários devem ser adaptadas para usuários em movimento, além dos resultados obtidos.

O Capítulo 8 compara o trabalho corrente com os trabalhos relacionados existentes na literatura. Por fim, o Capítulo 9 apresenta as conclusões relativas ao trabalho desenvolvido e detalha trabalhos futuros que podem ser seguidos por outros pesquisadores, a fim de melhorar o atual trabalho.

2 COMPUTAÇÃO PERVASIVA

A computação pervasiva (*Pervasive Computing*) tem como principal objetivo fazer com que a interação homem-computador seja invisível, ou seja, fazer com que as pessoas utilizem métodos computacionais sem ter a necessidade do conhecimento de sua funcionalidade. Desta forma, o computador tem a capacidade de capturar informação do ambiente no qual ele está e utilizá-la para construir de forma dinâmica modelos computacionais, ou seja, controlar, configurar e ajustar a aplicação para melhor atender as necessidades do dispositivo ou usuário. O ambiente também pode e deve ser capaz de detectar outros dispositivos que venham a fazer parte dele. Desta interação surge a capacidade de computadores agirem de forma "inteligente" no ambiente no qual nos movemos, um ambiente povoado por sensores e serviços computacionais (CHEN; FININ; JOSHI, 2003).

A Computação Pervasiva originou-se da proposta de Mark Weiser, a qual é dar acesso ao usuário no ambiente computacional, em qualquer lugar e a todo momento, por meio de qualquer dispositivo (WEISER, 1991). Mark Weiser conceitua a mesma como um conjunto de ambientes computacionais que interagem com o usuário de forma tão natural no qual são considerados invisíveis e imperceptíveis aos usuários. Ainda pode-se descrever a Computação Pervasiva como um novo paradigma computacional com tecnologia de comunicação e informação em qualquer lugar, acessível por qualquer pessoa, disponível todo o tempo (SAHA; MUKHERJEE, 2003).

A visão de computação pervasiva implica que em todos os lugares o ambiente é preenchido com recursos de rede (*software* e *hardware*) que podem ser descobertos e integrados para a realização de tarefas diárias (LIBRELOTTO et al., 2011).

A Computação Pervasiva é uma área nova de pesquisa na computação que tende a prover uma computação que não se prende somente nas estações de trabalho e torna-se transparente no dia a dia das pessoas. O usuário não necessita ter grandes conhecimentos sobre a tecnologia, ele deve apenas desfrutar dos benefícios que a mesma pode oferecer. Devido à ligação crescente da tecnologia da informação com o cotidiano das pessoas, a computação pervasiva desenvolveu-se e conseguiu espaço na área da computação.

Muitas áreas da computação englobam a computação pervasiva. As áreas em que a computação pervasiva mais está presente, segundo BARBOSA (2007), são as áreas da saúde (*healthcare*), lar (*homecare* e casas inteligentes), transporte (localização), educação e meio ambiente.

2.1 Características da Computação Pervasiva

A Computação Pervasiva foca na combinação entre a computação e comunicação com o ambiente físico, além de necessitar ser uma comunicação transparente para os usuários em suas atividades do cotidiano.

No estudo de AHAMED; ZULKERNINE; HAQUE (2008) fez-se um resumo de características estudadas por outros pesquisadores e chegou-se a conclusão de que as características necessárias para se obter a computação pervasiva são as seguintes:

- Desaparecimento de limites: a computação pervasiva distribui a computação além de qualquer limite físico, desta forma, o ambiente de trabalho da computação pervasiva inclui diferentes tipos de dispositivos;

- Ambiente volátil: dispositivos podem deixar e juntar-se em ambientes pervasivos de forma arbitrária e independente, produzindo uma topologia de rede extremamente volátil;

- Sensibilidade de contexto: ambientes pervasivos devem ser capazes de coletar informações sobre o seu redor (informações sobre o ambiente) e usar estas informações no processo de decisão. Como resultado, esse ambiente será capaz de servir diferentes usuários em diferentes situações e momentos;

- Escalabilidade: muitos dispositivos diferentes podem estar em um ambiente pervasivo, podendo conectarem-se entre si. Isso traz a necessidade da computação pervasiva suportar esta enorme gama de dispositivos heterogêneos;

- Serviço Contínuo: o ambiente pervasivo precisa assegurar uma conexão persistente entre os dispositivos conectados. Essa garantia de conexão deve se dar até o momento que o usuário desconecte seu dispositivo manualmente;

- Mobilidade: em um ambiente pervasivo, os usuários estão se movendo constantemente. Essa característica deve ser considerada no desenvolvimento de aplicações pervasivas;

- Espaço Ativo: a fim de incorporar poder computacional e suporte a características de sensibilidade de contexto, a computação pervasiva necessita incorporar muitos dispositivos, como por exemplo, sensores. Estes dispositivos podem coletar informações como identidade de um usuário, temperatura, som (ruído), entre outras, baseado no que se re-

quer. Esta característica torna um espaço físico em um espaço ativo que tem o papel de monitorar e ser capaz de tomar decisões.

2.2 Computação Sensível ao Contexto

Aplicações na área da computação pervasiva devem ser sensíveis ao contexto e possibilitar a inferência de fatos com base no comportamento do mesmo, tomando assim uma decisão autônoma. Quais decisões podem ser tomadas autonomamente devem possuir maior atenção, pois elas não são desejáveis em determinados contextos.

De acordo com DEY (2001), uma das noções fundamentais da computação pervasiva é a sensibilidade ao contexto, onde, por contexto entende-se qualquer informação relevante que possa ser utilizada para caracterizar a situação de alguma entidade, (por exemplo identidade e localização). YAMIN (2004) define contexto como qualquer informação gerada pela infra-estrutura computacional do ambiente que pode ser relevante para uma aplicação na qual uma alteração pode disparar um processo para a adaptação desta aplicação.

A computação sensível ao contexto tem como objetivo permitir que aplicações sejam capazes de acessar e capturar informações relacionadas aos processos que realizam em um determinado ambiente, a fim de aperfeiçoar seu desempenho (FREITAS, 2011).

Existem várias formas de implementar aplicações com sensibilidade ao contexto. A utilização de sensores, etiquetas RFID (*Radio Frequency Identification Device*) (RFID, 2011), entre outras são algumas das formas. A abordagem depende de requisitos e condições existentes no ambiente, como a localização dos sensores, o número de usuários, a disponibilidade dos recursos a serem utilizados e a facilidade para a extensão do sistema (CHEN, 2003).

2.3 Independência de Dispositivos na Computação Pervasiva

Para possibilitar que conteúdos desenvolvidos para a *Web* ou qualquer *software* hoje em dia na computação pervasiva sejam visualizados em qualquer dispositivo, é necessário que haja independência do mesmo. Criar uma interface para cada dispositivo de forma diferente e manual é inviável de forma administrativa e econômica, tendo em vista a demanda de dispositivos diferentes que existem. Essas diferenças vão desde o tamanho da tela até processamento, memória, funcionalidades, etc.

Uma independência de dispositivo pode ser alcançada combinando a formatação do conteúdo com as capacidades do dispositivo (FORTE, 2006). Com essa combinação, é possível

oferecer muitos benefícios aos usuários (por exemplo acessibilidade, onde os usuários são capazes de interagir com o sistema mesmo com suas limitações).

O Capítulo 3 descreve de forma mais detalhada estas questões, onde para alcançar uma independência de dispositivo, é necessário conseguir um perfil de dispositivo que descreva tudo que o dispositivo possui de características. Para isso, um vocabulário recomendado pela W3C chamado UAProf é utilizado.

2.4 Computação Pervasiva e Computação Móvel

A Computação Pervasiva é uma computação onde recursos dispostos em um ambiente inteligente se auto ajustam de forma a melhor atender as necessidades do usuário, ou seja, é uma computação onipresente (WEISER; GOLD; BROWN, 1999).

Ainda que a idéia de Weiser sobre a Computação Pervasiva esteja distante em relação a prática do dia a dia com dispositivos de mercado, a realização de sua proposta está cada vez mais vindo à tona com a disponibilização de tecnologias como PDAs, *Smartphones*, e a consolidação de redes sem fio como o *bluetooth* e o IEEE 802.11 (GASSEN et al., 2008).

A possibilidade dos usuários poderem se locomover e transpostar seus próprios dispositivos computacionais, expandindo assim a capacidade destes em usar serviços computacionais oferecidos independente de sua localização, se baseia na computação móvel (ARAUJO, 2003).

Uma das limitações da computação móvel é que o dispositivo não oferece flexibilidade de informação em relação ao contexto local, assim, ela não se adapta ao contexto atual (ARAUJO, 2003). A geração de interfaces para dispositivos móveis também gera um certo problema se criada somente pensando no dispositivo móvel e não levando em consideração o contexto.

Para sanar essas limitações da computação móvel, deve-se integrar a computação móvel com a pervasiva, fazendo com que a mobilidade e serviços do dispositivo móvel do usuário se ajuste ao ambiente sensível ao contexto suportado pela computação pervasiva.

Pesquisas na área de Interação Humano-Computador (IHC) para computação móvel tem se concentrado principalmente nos dispositivos computacionais, em como ajustar conteúdo em telas pequenas, como tornar os dispositivos mais inteligentes, como criar mecanismos mais rápidos de entrada, como estabelecer comunicação mais confiável, etc. (WOBBROCK, 2006).

Hoje em dia alcançou-se rapidamente o ponto onde já não pode-se dar ao luxo de considerar dispositivos isoladamente, e sim levar em conta fatores contextuais do ambiente onde o dispositivo se encontra.

Segundo WOBBROCK (2006), existem quatro tendências importantes na sociedade e tecnologia que possuem consequências diretas na interação humano-computador em dispositivos móveis. São eles:

1. O envelhecimento geral da população;
2. A quantidade crescente de uso de computadores pessoais fora das estações de trabalho;
3. O crescimento das capacidades de dispositivos menores; e
4. A convergência de recursos computacionais para dispositivos móveis.

Estas tendências, juntas, exigem que o futuro das pesquisas da área de IHC para computação móvel considere tanto o contexto quanto a capacidade dos dispositivos.

O envelhecimento geral da população faz com que a computação móvel melhore a questão de acessibilidade. Pessoas idosas tendem a possuir deficiências, incluindo perda de visão, coordenação e controle motor, dor nos dedos, pulsos ou outras articulações. Essas deficiências tornam o uso de dispositivos móveis problemático, visto que telas, botões e fontes pequenas geram dificuldades no manuseio do dispositivo.

A quantidade crescente de uso de computadores pessoais fora das estações de trabalho gera uma necessidade de maior atenção em relação a deficiências situacionais, onde um usuário pode estar em uma situação que torne o uso do dispositivo mais difícil. Essas situações podem ser, por exemplo: um usuário em movimento, onde pode ter a visão afetada no momento de ler algo na tela do dispositivo; um usuário utilizando luvas, dificultando o toque na tela de seu dispositivo sensível ao toque; ou um usuário na praia com bastante luminosidade, afetando o brilho da tela do seu dispositivo.

De acordo com WOBBROCK (2006), o crescimento das capacidades de dispositivos menores causa um problema para a computação móvel quando se trata de método de entrada, tornando frustrante para o usuário aprender, a cada novo dispositivo que utilizar, um novo método de entrada de dados. Para tirar proveito desse crescimento das capacidades dos dispositivos, é necessária a criação de métodos de entrada mais poderosos para os humanos utilizarem em qualquer dispositivo. Um exemplo citado por Wobbrock é a utilização de itens simples, como cartão de crédito, que no futuro pode ser capaz de aceitar uma senha, exibir um saldo ou limite de crédito no próprio cartão, ou mesmo ler impressões digitais.

Por fim, a convergência de recursos computacionais para dispositivos móveis traz um ganho enorme para certos países e culturas. Como Wobbrock cita em seu artigo, essa questão de convergência de recursos computacionais para dispositivos móveis pode ser notada na África. Com o crescimento do uso de dispositivos móveis, estudantes africanos podem ter oportunidades de utilizar a educação computadorizada pela primeira vez. Além da educação, fornecimento de informações médicas e de saúde, particularmente para as zonas rurais da África, é outro benefício que estes dispositivos móveis trazem, fazendo com que os usuários utilizem recursos em telefones móveis pequenos, sem a necessidade de um laptop volumoso.

As questões culturais, sociais, econômicas, educacionais, e médicas terão que ser entendidas antes que o software possa ser escrito, ou interfaces de usuário projetadas. Esta é uma tarefa grande, mas os benefícios gerados para certos países em crescimento e as disciplinas de computação podem ser enormes.

2.5 Considerações do Capítulo

Este capítulo apresentou características da Computação Pervasiva e também descreveu como a área de IHC é importante para a computação móvel, detalhando quatro tendências importantes para a sociedade e para a tecnologia, mostrando que essas tendências, caminhando lado a lado, exigem que futuras pesquisas na área de IHC para computação móvel devam cogitar tanto o uso do contexto, ou seja, utilizar a computação pervasiva, quanto a capacidade dos dispositivos. A seguir será detalhado o uso de perfis de adaptação de conteúdo, mostrando as tendências através do uso de perfis de usuários, contexto de ambiente, e perfis de dispositivos, tornando a abordagem genérica para qualquer dispositivo e contexto, e tratando as necessidades e preferências dos usuários de forma individual, através de seu perfil.

3 PERFIS PARA ADAPTAÇÃO DE CONTEÚDO E LINGUAGENS DESCRITIVAS DE INTERFACE DE USUÁRIO

A necessidade de perfis para adaptação de conteúdo surge pelo fato de existirem poucas formas eficientes para obter as informações sobre dispositivos, usuários e contexto em que ambos se encontram, além da complexidade de adaptar a interface do usuário de forma automática, devido à heterogeneidade dos dispositivos (diferentes processadores, memórias, tamanho de tela, etc). Desta forma, este trabalho apresenta uma infraestrutura baseada no *framework Composite Capability/Preference Profiles* (CC/PP) (KISS, 2007) e na especificação *User Agent Profiling Specification* (UAProf) (ALLIANCE, 2001) capaz de oferecer as informações necessárias para que possa ser realizada a adaptação do conteúdo necessário. Tais especificações têm a finalidade de possibilitar o acesso a informações estáticas e dinâmicas dos dispositivos e dos usuários. O CC/PP e o UAProf são recomendações W3C (W3C, 2011), os quais são descritos na seção 3.1.

Além da necessidade de captar informações sobre dispositivos, é de suma importância coletar informações sobre os usuários finais. A seção 3.2 detalha os dados referentes ao usuário final.

3.1 CC/PP e UAProf

O CC/PP tem o objetivo de expressar características de dispositivos e preferências de usuários. Baseado no *Resource Description Framework* (RDF, 2004), o CC/PP é descrito em documentos RDF ou *eXtensible Markup Language* (XML) (XML, 2011), os quais, na criação de novos vocabulários, permitem uma maior flexibilidade.

Para descrever as características dos dispositivos e dos usuários, o CC/PP especifica uma estrutura geral de um documento XML, o qual pode ser dividido em componentes (por exemplo *Hardware* e *Software*). Para descrever as características básicas dos dispositivos (dimensões de tela, sistema operacional, etc.), o CC/PP fornece um vocabulário padrão. Entretanto, o CC/PP não pode evitar que cada fabricante crie um vocabulário próprio, dificultando assim a padronização de sistemas para a obtenção das características dos dispositivos.

Com a necessidade de padronização de vocabulários CC/PP entre fabricantes de dispositivos móveis, a *Open Mobile Aliance* (ALLIANCE, 2001) criou a especificação UAProf. Para descrever as características dos dispositivos, a especificação UAProf possui um conjunto de vocabulários. Um documento UAProf é criado e armazenado em um repositório *Web* do fabricante

para cada dispositivo projetado.

O UAProf está concentrado na captura de classes de capacidades de dispositivos e informações de preferências. Estas classes possuem características de *hardware* e *software* do dispositivo, além de informações sobre a rede a qual o dispositivo se encontra, mas não estão restritas somente a elas (ALLIANCE, 2001).

O esquema UAProf, descrito em (ALLIANCE, 2001), é composto pelo seguintes componentes:

- *HardwarePlataform* - é uma coleção de propriedades que descrevem as características de *hardware*. Isto inclue o tipo de dispositivo, modelo, tamanho da tela, métodos de entrada e saída, etc;
- *SoftwarePlataform* - é uma coleção de atributos associados ao *software* do dispositivo. Estes atributos fornecem informações sobre o sistema operacional, video e áudio, etc;
- *BrowserUA* - conjunto de atributos para descrever o *browser* HTML do dispositivo;
- *NetworkCharacteristics* - informações sobre a infraestrutura e ambiente tais como a operadora, largura de banda, etc;
- *WapCharacteristics* - conjunto de informações relacionadas as capacidades WAP do dispositivo;
- *PushCharacteristics* - conjunto de atributos referentes às capacidades *Push* suportadas pelo dispositivo. Isto inclue detalhes de suporte a tipos MIME (Extensões Multi-função para Mensagens de Internet), tamanho máximo de uma mensagem, etc.

A utilização do UAProf neste trabalho vem da necessidade da captura de informações estáticas dos dispositivos.

3.1.1 XML (*eXtensible Markup Language*)

XML é uma recomendação da W3C para gerar linguagens de marcação, a qual é capaz de descrever diversos tipos de dados. O objetivo principal do XML é facilitar a troca de informações.

Existem várias vantagens do uso de XML, algumas delas, segundo XML (2011), são:

- Os dados em XML são armazenados em formato de texto simples (txt). Isso torna muito mais fácil a criação de dados que podem ser compartilhados por diferentes aplicações;

- A maioria dos desenvolvedores perdem muito tempo com troca de dados entre sistemas incompatíveis na internet. O XML reduz esse tempo, pois pode ser lido por diferentes sistemas;
- Dados armazenados em XML tornam fácil a expansão ou atualização para novos sistemas operacionais, novas aplicações, etc, sem perda de dados;
- XML suporta qualquer tipo de máquina de leitura, sejam computadores portáteis, máquinas de voz, *feeds* de notícias, etc. Isto faz com que pessoas com deficiências tenham mais chance de acessar a computação;
- XML pode ser utilizado para criar novos tipos de linguagens para a internet. Por exemplo: OWL, RDF, XHTML, WSDL, RSS, etc.

XML é muito semelhante à linguagem de apresentação HTML. Contudo, *tags* HTML servem apenas para definir parágrafos e estilos de formatação de conteúdo para apresentação do mesmo. Arquivos XML podem possuir quantas *tags* o usuário desejar, pois possui um esquema para criação de *tags* para dados estruturados.

3.1.2 RDF (*Resource Description Framework*)

RDF é um modelo padrão para intercâmbio de dados na *Web* recomendado pela W3C. O principal objetivo do RDF é criar um modelo simples de dados, com uma semântica formal, utilizando um vocabulário baseado em um Identificador Uniforme de Recursos (URI) e uma sintaxe baseada em XML. De acordo com (RDF, 2004), três componentes básicos compõem os arquivos RDF, tornando a linguagem altamente escalável: recurso, propriedade e indicação.

- Recurso - qualquer coisa que pode conter um URI, incluindo as páginas *Web*, assim como elementos de um documento XML;
- Propriedade - um recurso que tenha um determinado nome e possa ser utilizado como uma propriedade;
- Indicação - consiste na combinação de um recurso, de uma propriedade, e de um valor.

3.2 Perfil do Usuário

Existem diversas formas de criar e personalizar interfaces de usuários. Uma das mais populares formas é o *design* centrado no usuário (*User-Centered Design* - UCD) (NORMAN;

LEWIS; DRAPER, 1986), o qual mantém uma participação ativa com o usuário, afim de dar atenção aos requisitos necessários em cada estágio do processo de *design* da interface, e no fim, testar com usuários reais.

Para que uma interface se adapte às necessidades do usuário, é necessário levantar determinados aspectos para definir o perfil do usuário final. Baseado no estudo de SHNEIDERMAN (2000), um dos pioneiros na área de IHC, esses aspectos, separados em grupos, são:

- Deficiências: motora, visual, auditiva, cognitiva;
- Preferências: idioma, somente texto, somente imagens, cor, etc;
- Experiência com computador (em níveis): iniciante, mediano, experiente;
- Dados demográficos: idade, sexo e nacionalidade;
- Dados culturais.

Com um perfil de usuário pré-especificado, a participação ativa do usuário especificando requisitos relacionados à interface não é mais necessária, pois a interface final do produto em questão, agora deve se aplicar a esse perfil. As subseções seguintes descrevem cada grupo citado acima.

3.2.1 Deficiências

Há uma diversidade de definições para deficiências auditivas, visuais e motoras. Desta forma, a comparação entre os estudos se torna difícil. Para definir os níveis de deficiência auditiva, utilizou-se a classificação do *World Health Organization* (WHO) WHO (1991) que classifica o comprometimento de audição relacionado a média tonal na melhor orelha.

A medição é feita utilizando valores ISO audiométricos (média de 500, 1000, 2000, 4000 Hz) e variam em níveis de grau de comprometimento, onde cada grau de comprometimento possui um valor ISO em dB (decibéis) associado. O grau 0 (sem comprometimento) possui um valor ISO de 25 dB ou menos e assim a pessoa não tem problemas de audição ou, caso tenha, é muito leve. O grau 1 (comprometimento leve) vai de 26 a 40 dB e a pessoa é capaz de ouvir e repetir palavras ditas em voz normal em até um metro. O comprometimento moderado, de grau 2 fica entre 41 e 60 dB e a pessoa é capaz de ouvir e repetir as palavras ditas em voz alta em até um metro. Entre 61 e 80 dB fica o grau 3, de comprometimento severo, onde a pessoa é capaz de ouvir algumas palavras, quando gritar na melhor orelha. Por ultimo, o grau 4 de

comprometimento profundo, incluindo surdez e tem um valor ISO audiométrico de 81 dB ou mais e a pessoa se torna incapaz de ouvir e compreender, mesmo uma voz muito alta.

A definição dos níveis relacionados à visão, também são baseados na classificação do *World Health Organization* (WHO, 1997). WHO classifica o comprometimento visual relacionado com a visão no melhor olho (possível correção com o uso de óculos) como perda leve da visão ou visão quase normal, possuindo valores em um intervalo de 20/30 até 20/60. De 20/70 a 20/160 é considerada deficiência visual moderada ou baixa visão moderada. O terceiro nível de deficiência têm valores de 20/200 a 20/400 e é considerada uma deficiência visual grave ou baixa visão severa. A deficiência visual profunda ou baixa visão profunda está dentre os limites de 20/500 a 20/1000. E a deficiência visual com valor menor do que 20/1000 é considerada quase uma deficiência visual total ou cegueira quase total. Por fim, a cegueira total ou deficiência visual total é quando não há percepção de luz.

Segundo o guia dos direitos de pessoas com deficiência de SKAF; MATTOS; BARA (2007), a deficiência motora refere-se a alteração completa ou parcial de um ou mais segmentos do corpo humano, exceto as deformidades estéticas e as que não produzam dificuldades para o desempenho de funções, acarretando o comprometimento da função física, apresentando-se sob a forma de:

- Paraplegia: perda total das funções motoras dos membros inferiores;
- Paraparesia: perda parcial das funções motoras dos membros inferiores;
- Monoplegia: perda total das funções motoras de um só membro (inferior ou posterior);
- Monoparesia: perda parcial das funções motoras de um só membro (inferior ou posterior);
- Tetraplegia: perda total das funções motoras dos membros inferiores e superiores;
- Tetraparesia: perda parcial das funções motoras dos membros inferiores e superiores;
- Triplegia: perda total das funções motoras em três membros;
- Triparesia: perda parcial das funções motoras em três membros;
- Hemiplegia: perda total das funções motoras de um hemisfério do corpo (direito ou esquerdo);

- Hemiparesia: perda parcial das funções motoras de um hemisfério do corpo (direito ou esquerdo); e
- Amputação: perda total ou parcial de um determinado membro ou segmento de membro.

SKAF; MATTOS; BARA (2007) também descreve a deficiência cognitiva como sendo o funcionamento intelectual significativamente inferior a média, com manifestação antes dos dezoito anos de idade e limitações associadas a duas ou mais áreas de habilidades adaptativas, tais como: comunicação, cuidado pessoal, habilidades sociais, utilização dos recursos da comunidade, saúde e segurança, habilidades acadêmicas, lazer, e trabalho.

Pessoas idosas as quais possuem tendência a ter alguma deficiência, seja ela motora, visual, auditiva, ou cognitiva devem também conseguir utilizar dispositivos computacionais como qualquer outra pessoa. Estas pessoas especiais tendem a possuir dificuldades quando utilizam dispositivos computacionais. Estudos feitos por ZIEFLE; BAY (2005) e KURNIAWAN (2008) mostram que idosos encontram problemas de usabilidade em dispositivos não familiares, como por exemplo telefones móveis.

Habilidades do usuário não são determinadas somente pela sua saúde, mas também pelo contexto do ambiente atual (NEWELL, 1995) e (SEARS et al., 2003).

3.2.2 Preferências

As preferências do usuário final devem ser um dos fatores mais importantes para a geração de interfaces, visto que é através destas preferências que a interface final ficará de acordo com o que o usuário necessita, satisfazendo seus desejos (LIU; OSVALDER; KARLSSON, 2010). Dados como cor preferida de fundo e de texto, idioma preferido, se deseja somente texto em sua interface ou somente imagens, fonte preferida, entre outros, são de extrema importância para o propósito da personalização da interface do usuário.

A importância de conhecer as preferências dos usuários é para melhor adequar a interface, e sua usuabilidade, de maneira a formar um ambiente confortável e eficaz de se trabalhar. Desta forma o usuário alcança um maior nível de experiência com o dispositivo que está utilizando, visto que a interface está adaptada para suas preferências e necessidades.

3.2.3 Experiência com Computadores

Dados como experiência com computadores foram estudados em várias pesquisas e foram classificados em diferentes categorias. SHNEIDERMAN (1992) classificou em três classes

comuns: Usuários novatos, usuários ocasionais e usuários especialistas. Usuários novatos são aqueles que sabem o que devem fazer, porém não possuem conhecimento ou possuem pouco conhecimento sobre o sistema. O usuário ocasional é aquele usuário que sabe a tarefa que deve desempenhar no sistema, mas devido ao uso pouco frequente, acaba tendo dificuldade em lembrar como realizar seus objetivos no sistema. Por fim, o usuário especialista é aquele que tem o conhecimento profundo das tarefas que deve desempenhar e sabe as ações necessárias para cumprí-las.

3.2.4 Dados Demográficos

Dados demográficos como idade e sexo do usuário influenciam muito no momento da criação/personalização da interface de um sistema computacional. De acordo com os estudos de OWSLEY et al. (1991), BOTWINICK (1973) e WELFORD (1977), usuários idosos tiveram problemas com o processo cognitivo, tal como atenção, e também uma demora em relação às tarefas que normalmente são de rápido desempenho, o que demonstra que a idade é um fator importante para o propósito deste trabalho. BROOS (2005) realizou um estudo sobre aspectos relacionados ao sexo do ser-humano e concluiu que mulheres mostram mais ansiedade quando estão utilizando um sistema computacional do que homens, mostrando que os homens são mais seguros de si e as mulheres são mais hesitantes.

3.2.5 Dados Culturais

Diferenças culturais também interferem no desenvolvimento de interfaces de usuários. Os usuários em alguns países tendem a estar mais familiarizados com computadores do que em outros países. Segundo IBM (2001), as diferenças culturais e geográficas devem ser aplicadas, fazendo com que os *designers* de interfaces tomem a iniciativa de compreender a sua comunidade de usuários e agreguem essas diferenças, visando obter o melhor resultado.

Tendo em vista a familiaridade com computadores em diferentes países, a nacionalidade do usuário também é importante, pois através dela tiram-se algumas conclusões em relação a sua cultura e desta forma pode-se personalizar a interface de acordo com certos dados culturais.

Segundo REINECKE; GAJOS (2011), resultados de estudos comportamentais mostram que a formação cultural de uma pessoa pode ser correlacionada com certas preferências e habilidades. Os autores ainda apontam que a cultura influencia no comportamento das pessoas, refletindo parcialmente em mudanças neuroatômicas no cérebro, alterando as habilidades para perceber e interpretar informações.

Os pesquisadores GALDO; NIELSEN (1996) e BURGMANN; KITCHEN; WILLIAMS (2006) tem mostrado que projetos de interfaces de usuário diferem entre países. Um exemplo disso são *websites* projetados por membros de cultura oriental, que muitas vezes são mais complexos visualmente e mais coloridos do que *websites* de cultura ocidental (CHOI, 2005).

De acordo com LINDSEY; BROWN (2004), algumas culturas não percebem a diferença entre certas cores, tais como verde e azul. A razão para isso é o idioma que as pessoas falam. Um exemplo disso são os Russos, pois eles percebem uma maior gama de diferentes tons de azuis do que pessoas que falam outro idioma (BORODITSKY, 2009).

Além de percepção de cores, o idioma também difere quanto a orientação (BORODITSKY, 2009). Um exemplo bem interessante de como o idioma muda a forma de como um povo se orienta, é a maneira de orientação de uma comunidade aborígena da Austrália, chamada Kuuk Thaayorre. Essa comunidade se orienta através de pontos cardeais. O estudo de Boroditsky mostra que não importa para onde uma pessoa dessa comunidade esteja virada, seja para o sul, norte, leste, ou oeste, ela sempre se orientará do leste para o oeste. Um experimento que Boroditsky fez com algumas pessoas desta comunidade foi pedir para que elas organizassem cartas em ordem temporal. Foi testado duas vezes por cada pessoa, cada vez olhando para um ponto cardeal diferente. Quando os Kuuk Thaayorre estavam sentados de frente para o sul, eles organizavam as cartas da esquerda para a direita. Quando estavam encarando o norte, as cartas ficavam da direita para a esquerda. Quando virados para o leste, as cartas eram organizadas em direção ao seu corpo, e quando estavam virados para o oeste, as cartas eram colocadas em ordem a partir do seu corpo.

As pessoas que falam em inglês, ao realizar uma tarefa dessas, iriam organizar as cartas da esquerda para a direita. Os que falam a língua hebraica, iriam colocar as cartas da direita para a esquerda. Todo esse estudo feito por Boroditsky mostra que a forma que cada cultura organiza o tempo (orientação), depende da direção da escrita de sua linguagem.

Outros fatores culturais que influenciam nos projetos de interfaces são: a maneira que os pais e professores na escola educam, práticas e regras de organização, o nível de educação, e religião (KARAHANNA; EVARISTO; SRITE, 2005).

3.3 Linguagens Descritivas de Interface de Usuário

Uma linguagem descritiva de interface de usuário (LDIU) é uma linguagem de marcação que renderiza e descreve componentes de interfaces gráficas e comportamento dos mesmos.

Uma LDIU é uma linguagem formal usada na IHC para descrever uma interface de usuário particular e independente de qualquer implementação. Assim, basta descrever a interface de usuário com uma LDIU e utilizar uma linguagem lógica para criar alguma funcionalidade para a mesma.

Existem inúmeras Linguagens Descritivas de Interfaces de Usuário (LDIUs) disponíveis. Algumas das LDIUs mais conhecidas são XUL (XML *User Interface Language*) (MOZILLA, 2010), UIML (*User Interface Markup Language*) (UIML, 2010), XAML (*Extensible Application Markup Language*) (XAML, 2010), UsiXML (*User Interface eXtensible Markup Language*) (USIXML, 2010), entre outras. Essas LDIUs são linguagens que descrevem os componentes da interface, disposição na tela, e comportamento, ou seja, toda a arquitetura de uma interface. Com a utilização destas linguagens, as informações da interface são lidas e então é gerado um código executável para o dispositivo específico. Desta forma, não é necessário criar um projeto para cada dispositivo, pois as interfaces são apenas montadas em cada dispositivo. Isso faz com que aumente a produtividade, resultado da diminuição da reimplementação.

O interesse no uso de linguagens abstratas para descrever interfaces de usuário surgiu por causa dos problemas de usabilidade e adaptabilidade relacionados à IHC. As linguagens abstratas devem fornecer toda informação necessária para criar uma interface para qualquer tipo de usuário. Para geração de interfaces de usuários de forma automática, uma representação abstrata (linguagem abstrata) deve ser bem expressiva (TREWIN; ZIMMERMANN; VANDERHEIDEN, 2003).

Estas linguagens abstratas (LDIUs) são linguagens que especificam características de uma interface de usuário para sistemas interativos. As LDIUs são compostas por uma sintaxe, que define os termos e a gramática da linguagem, e semântica, que da o significado e as relações entre os termos (SOUCHON; VANDERDONCKT, 2003).

Os objetivos que uma LDIU deve possuir, segundo LIMBOURG et al. (2004), são:

- Usar uma descrição que permita a geração automática do código da Interface do Usuário;
- Ter um formato de uma comunicação;
- Ser uma linguagem de especificação;
- Ser portável entre plataformas, mantendo sua consistência;
- Capturar requisitos da Interface do Usuário em uma definição abstrata;

- Dar suporte a sensibilidade ao contexto;
- Dar suporte a extensibilidade e adaptabilidade às Interfaces de Usuário; e
- Melhorar a reusabilidade do *design* das Interfaces de Usuário.

Grande parte das LDIUs são linguagens baseadas em XML (BRAY et al., 2008). XML também é uma linguagem declarativa, desta forma, pode ser utilizada facilmente por não programadores (SOUCHON; VANDERDONCKT, 2003).

Para o suporte a adaptabilidade de interfaces, as linguagens baseadas em XML são ideais, graças a sua versatilidade de manutenção, extensão, e capacidade de refinamento que os documentos XML proporcionam (MOLINA, 2003). As linguagens baseadas em XML se encaixam bem em especificações de interfaces independentes de dispositivo, plataforma e contexto. Estas linguagens possuem níveis de abstração, fazendo com que algumas delas sejam menos expressivas que as outras, e desta forma, não servem para o desenvolvimento de algumas interfaces de usuário.

Uma das vantagens destas linguagens descritivas de interface de usuário abstratas é que ao desenvolver um sistema, o *designer* da interface não necessita ter conhecimento da lógica da programação, e sim, somente da descrição da interface. Assim, o desenvolvimento do sistema é feito de forma mais rápida e com maior qualidade. Um exemplo disso é como a Microsoft trabalha nesse quesito. Eles possuem uma linguagem descritiva de interface de usuário chamada XAML, a qual descreve os componentes da interface do usuário, bem como os atributos (cor, alinhamento, tamanho de fonte, etc.) dos componentes e a lógica da programação fica por parte de uma das linguagens do *Framework* .NET (C#, ASP.NET, VB, etc.) (MICROSOFT, 2009).

Para definirmos qual a linguagem de descrição de interfaces de usuário utilizar nesse trabalho, algumas questões devem ser analisadas. Essas questões, de acordo com LIMBOURG et al. (2004) são:

- Facilidade de programação: devido ao fato de que todas as linguagens são baseadas em XML, a programação dos documentos nas diferentes linguagens é um trabalho bastante simples;
- Portabilidade: capacidade que um sistema tem de ser executado em diferentes arquiteturas de sistemas operacionais, ou de dispositivos (*hardware*);

- Extensibilidade: capacidade de criar novos componentes necessários no desenvolvimento da interface;
- Elementos de interface: suporte à componentes *built-in* de interface como botões, janelas, menus, entre outros;
- Eventos: suporte ao comportamento de componentes;
- Integração com outras tecnologias: suporte à integração com ferramentas externas; e
- Documentação: facilidade de encontrar conceitos e documentos relacionados.

A tabela 3.1, compara as principais linguagens descritivas de interfaces de usuário para demonstrar, dentre as questões citadas acima, qual a linguagem escolhida para efetuarmos os testes nesta dissertação.

Tabela 3.1: Comparativo das Linguagens Descritivas de Interface em XML

Tabela de comparação das LDIUs					
	XUL	**UIML**	**XAML**	**UsiXML**	**Android**
Facilidade de Programação	ALTA	ALTA	ALTA	ALTA	ALTA
Portabilidade	ALTA	MÉDIA	BAIXA	ALTA	ALTA
Extensibilidade	SIM	NÃO	SIM	SIM	SIM
Elementos de Interface	SIM	SIM	SIM	SIM	SIM
Eventos	SIM	SIM	SIM	SIM	SIM
Integração com outras tecnologias	SIM	SIM	NÃO	SIM	SIM
Documentação	BAIXA	BAIXA	MÉDIA	MÉDIA	ALTA

A partir do estudo comparativo entre as linguagens de descrição de interfaces de usuário, a escolhida foi a linguagem que descreve a interface do Android, pois, dentre as estudadas, foi a que melhor atendeu aos requisitos.

A linguagem descritiva do Android não é comparada no estudo de LIMBOURG et al. (2004), o estudo realizado sobre esta linguagem apenas utilizou as questões relevantes levantadas por Limbourg.

Como pode notar-se, a tabela 3.1 mostra que tanto a linguagem que descreve a interface do Android, como a linguagem UsiXML, empatam nos quesitos levantados por Limbourg, exceto o quesito Documentação, o qual Android leva vantagem. Além da vantagem da documentação, ferramentas para edição, tanto de código, como componentes e montagem de interface

do UsiXML mostraram falhas, além de ser uma linguagem mais antiga. Desta forma, optou-se por utilizar a linguagem que descreve interfaces para Android. A seção 3.4 detalhará esta linguagem.

3.4 Android

Android é um sistema operacional móvel baseado em uma versão modificada do Linux. Ela foi originalmente desenvolvida por uma equipe de mesmo nome, Android, Inc. Em 2005, como parte de sua estratégia para entrar no espaço móvel, o Google comprou o Android e assumiu o seu trabalho de desenvolvimento (assim como sua equipe de desenvolvimento) (LEE, 2011).

A empresa Google queria o Android para torná-lo *open-source* e um *software* livre. Portanto, a maioria do código do Android foi liberado sob a licença Apache *open-source*, o que significa que qualquer um que queira usar o Android, pode baixar o código fonte completo do Android. Além disso, os fornecedores (fabricantes de *hardware* em geral) podem adicionar suas próprias extensões proprietárias para o Android e personalizá-lo, para diferenciar seus produtos dos outros. Este modelo de desenvolvimento simples torna o Android muito atraente, despertando o interesse de muitos fornecedores (LEE, 2011).

Empresas como a Motorola e Sony possuíam seus próprios sistemas operacionais. Quando o iPhone da Apple foi lançado, muitos desses fabricantes tiveram que lutar para encontrar novas formas de revitalizar seus produtos. Estes fabricantes viram no Android uma solução - eles continuarão a projetar seu próprio *hardware* e usarão o Android como sistema operacional (LEE, 2011).

Segundo LEE (2011), a principal vantagem da adoção do Android é que ele oferece uma abordagem unificada para desenvolvimento de aplicações. Os desenvolvedores só precisam desenvolver para Android, e sua aplicação deve ser capaz de rodar em diversos dispositivos diferentes, desde que os dispositivos suportem Android. No mundo dos *smartphones*, os aplicativos são a parte mais importante da cadeia de sucesso.

De acordo com MOBILE (2009), o qual trata das diretrizes de projetar aplicativos em Android (Android *Design Guidelines*), projetar um aplicativo em Android é o mesmo que projetar qualquer outro aplicativo móvel. Aplicativos Android seguem as mesmas regras de experiência de usuários que todos os aplicativos móveis devem seguir. Estas regras são baseadas em (1) conhecer seu público; (2) simplificar sua funcionalidade quando puder, mantendo organizado e limpo quando não puder; (3) manter o aplicativo intuitivo e amigável, sabendo que você tem

meros segundos para ganhar ou perder um usuário.

Dicas de como projetar um bom aplicativo em Android são detalhadas em (MOBILE, 2009). Estas dicas são sobre tamanho e resolução de ícones, botões, imagens, *layouts*, entre outros *widgets*.

3.4.1 Versões do Android

O sistema operacional Android tem passado por várias atualizações desde que foi criado. A figura 3.1 mostra as versões, seus codinomes, nível da API utilizada e o percentual de distribuição.

Platform	Codename	API Level	Distribution
Android 1.5	Cupcake	3	0.6%
Android 1.6	Donut	4	1.1%
Android 2.1	Eclair	7	8.5%
Android 2.2	Froyo	8	30.4%
Android 2.3 - Android 2.3.2	Gingerbread	9	0.6%
Android 2.3.3 - Android 2.3.7		10	54.9%
Android 3.0	Honeycomb	11	0.1%
Android 3.1		12	1.5%
Android 3.2		13	1.7%
Android 4.0 - Android 4.0.2	Ice Cream Sandwich	14	0.3%
Android 4.0.3		15	0.3%

Figura 3.1: Versões do Android. (ANDROID, 2011a)

A figura 3.1 é baseada no número de dispositivos Android que acessaram o Mercado Android (Android *Market*) durante 14 dias, finalizando em 03 de janeiro de 2012.

Os níveis de API indicam as atualizações na API do Android. Assim, um aplicativo desenvolvido na versão 2.2 do Android, por exemplo, pode ser executado em qualquer versão do Android que seja mais atual. Caso a versão seja inferior a que o aplicativo foi desenvolvido, talvez ela não funcione, depende dos recursos utilizados no projeto e também da API da versão inferior, a qual pode não possuir recursos que foram utilizados no projeto do aplicativo.

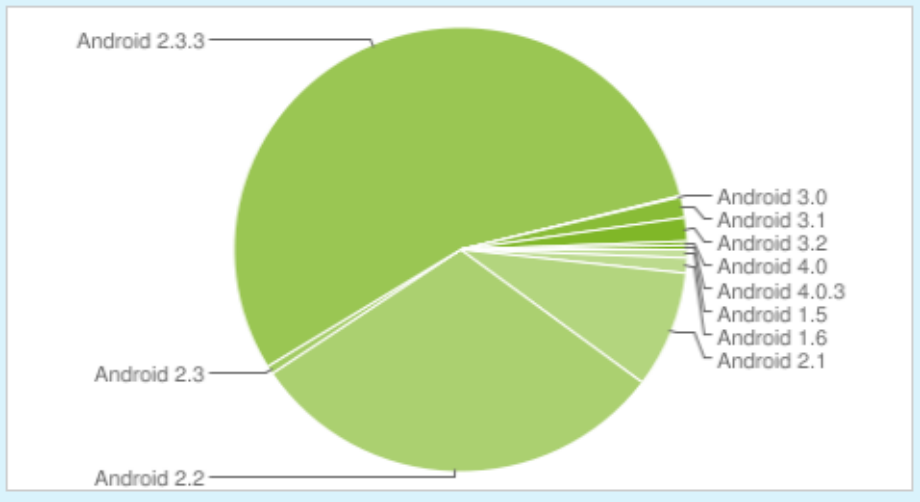

Figura 3.2: Distribuição atual das versões dos sistemas operacionais Android. (ANDROID, 2011a)

A figura 3.2 mostra a quantidade de distribuições em cada versão do Android. Como é notável, a versão 2.2 e 2.3.3 são as mais distribuídas dentre as disponíveis. Os dados da figura 3.2 também foram coletados da mesma forma e mesmo período que os dados da figura 3.1.

3.4.2 SDK do Android

O SDK (*Software Development Kit*) do Android fornece as ferramentas e APIs necessárias para iniciar o desenvolvimento de aplicações na plataforma Android usando a linguagem de programação Java (ANDROID, 2011b). Porém, uma das diferenças do SDK Android para o SDK Java, é que o do Android suporta menos funcionalidades do que o SDK Java completo. A principal diferença é compilar um programa em Java e compilar um programa em Android. A figura 3.3 mostra esta dissemelhança.

Em Java, o código fonte é escrito e compilado em Java *byte code* usando o compilador Java, logo após, este *byte code* é executado em uma máquina virtual Java (JVM - *Java Virtual Machine*).

Em Android a compilação é diferente. O código fonte é escrito em Java, e este código é ainda compilado em Java *byte code* usando o mesmo compilador Java. Porém, a partir deste ponto, o código é recompilado usando o compilador Dalvik, gerando assim um Dalvik *byte code*. Este Dalvik *byte code* é então executado em uma máquina virtual Dalvik (DVM - Dalvik *Virtual Machine*).

A compilação em Android parece mais complicada, porém, ao utilizar a IDE Eclipse, juntamente com as ferramentas que o SDK do Android disponibiliza, como o Ant (ferramenta para

automação de processos para construção de *softwares*), os passos adicionais aos da compilação Java são automatizados e o programador não nota essas diferenças.

A maior diferença entre Android e Java, é a biblioteca para geração de interface de usuário que elas utilizam. Android possui um conjunto de bibliotecas próximo a versão *Standard Edition* do Java, mas possui características e funcionalidades que são úteis somente para projetos Android.

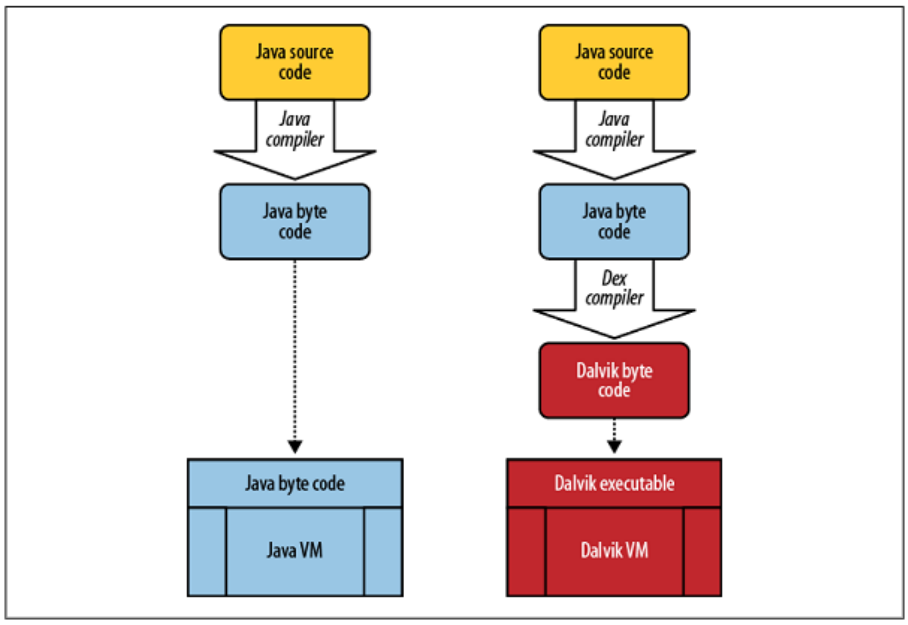

Figura 3.3: Compilação Android vs. Compilação Java. (GARGENTA, 2011)

3.4.3 Interface de Usuário do Android

Interfaces de usuário em Android podem ser geradas de duas maneiras: via um arquivo XML ou via código Java.

Em uma aplicação, a interface do usuário é construída usando objetos View e ViewGroup. Há muitos tipos de *Views* e *ViewGroups*, onde cada um é descendente da classe View.

Objetos View são unidades básicas da interface do usuário na plataforma Android. A classe View serve como base para subclasses chamadas *widgets*, os quais oferecem objetos de interface totalmente implementados, como campo de texto e botões. A classe ViewGroup serve como base para subclasses chamadas *layouts*, os quais oferecem diferentes tipos, como linear, tabular, e relativo (ANDROID, 2011b).

Um objeto View é uma estrutura de dados cujas propriedades armazenam os parâmetros do *layout* e o conteúdo para uma área retangular específica da tela. Um objeto View lida com

suas próprias medidas, *layouts*, desenhos, mudanças de foco, *scrolling* e eventos que possam acontecer na área retangular a qual ele reside (ANDROID, 2011b).

Na plataforma Android, você define uma atividade da interface do usuário usando uma hierarquia de View e ViewGroup, como mostrado na figura 3.4. Esta árvore hierárquica pode ser simples ou complexa, depende do que você necessita. A hierarquia pode ser formada por um conjunto pré-definido de *widgets* e *layouts*, ou com `Views` criadas pelo desenvolvedor.

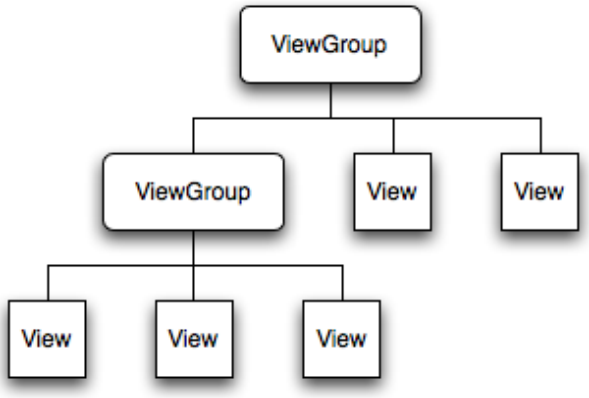

Figura 3.4: Objetos *View* e *ViewGroup* que formam a interface do usuário em Android. (ANDROID, 2011c)

A fim de renderizar a árvore hierárquica de exibição dos objetos, sua atividade precisa chamar o método setContentView() e passar a referência do nodo raíz, o qual contém o ViewGroup (*layout*) inicial. O nodo raíz da hierarquia recebe esta referência e usa ela para declarar, medir, e desenhar a árvore. Os nodos filhos podem requisitar um tamanho e localização, mas o objeto pai tem a decisão final de tamanho e localização que cada filho deve ter. O Android analisa os elementos de seu *layout* do topo da hierarquia até o filhos folha (*top-down*).

Por convenção, o arquivo XML inicial que descreve a interface do usuário é nomeado main.xml, e é chamado da seguinte forma: setContentView(R.layout.main). A seção 3.4.5 detalha os tipos de recursos de aplicativos Android utilizados pela classe R do Android.

3.4.4 *Layout* XML

A maneira mais comum de definir *layouts* em Android é através de arquivos XML. XML oferece uma estrutura legível para o *layout*, como HTML. Cada elemento no arquivo XML é um objeto View ou ViewGroup. Objetos View são nodos folhas na árvore, e objetos ViewGroup são ramos na árvore (note a figura 3.4).

O nome de um elemento XML é respectivo à classe Java que ele representa. Assim, um

elemento <TextView> cria um *widget* TextView na interface do usuário, e um elemento <LinearLayout> cria um ViewGroup LinearLayout. Por exemplo, para criar um *layout* vertical simples, com um campo de texto e um botão, em XML, é necessário descrevê-lo como na figura 3.5.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

Figura 3.5: Exemplo de um *layout* XML (ANDROID, 2011c)

Note que o elemento LinearLayout contém os elementos <TextView> e <Button>. É possível aninhar outros *layouts* (ViewGroup), assim, expande-se a hierarquia e cria-se um *layout* mais complexo. Existe uma variedade de formas de construir um *layout*. Pode-se aninhar e mesclar diferentes tipos de ViewGroup. Alguns objetos do tipo ViewGroup pré-definidos oferecidos pelo Android (chamados de *layout*) incluem LinearLayout, RelativeLayout, TableLayout, GridLayout, entre outros. Cada tipo de *layout* citado oferece um conjunto único de parâmetros que são usados para definir as posições dos nodos filhos (*widgets*) e a estrutura do *layout*.

Um *widget* é um objeto da classe View que serve como um componente de interface para interação com o usuário. O Android fornece um conjunto de *widgets*, como botões, *checkboxes*, campos de entrada de texto, calendários, relógios, entre outros, ajudando o desenvolvedor a desenvolver a interface do usuário rapidamente. O Android não limita o desenvolvedor a usar somente os *widgets* pré-estabelecidos no SDK. O desenvolvedor fica livre para criar e customizar seus próprios *widgets*.

3.4.5 Recursos de Aplicativos Android

Desenvolvedores devem sempre separar recursos tais como imagens e *strings* do código da aplicação; desta forma, eles tornam-se independentes. A separação dos recursos também habilita o desenvolvedor a fornecer recursos alternativos que suportam configurações de dispositivos específicas, tais como diferentes linguagens ou tamanhos de tela, o que torna cada vez

mais importante, à medida que mais dispositivos Android tornam-se disponíveis com diferentes configurações. A fim de proporcionar maior compatibilidade com diferentes configurações, o desenvolvedor precisa organizar os recursos de seu projeto na pasta res/, usando vários subdiretórios que agrupam recursos por tipo e configuração.

Para qualquer tipo de recurso, pode-se especificar recursos padrões ou múltiplos recursos alternativos. Os recursos padrões são aqueles que devem ser usados independentemente da configuração do dispositivo ou quando não existem recursos alternativos. Recursos alternativos são aqueles que são criados para utilizar com uma configuração específica. Por exemplo, enquanto o *layout* padrão da interface do usuário é armazenada no diretório res/layout/, o desenvolvedor pode especificar um *layout* diferente para ser usado quando a tela está na orientação paisagem (*landscape*), salvando esse *layout* alternativo no diretório res/layout-land/. O Android automaticamente aplica o recurso apropriado, combinando a configuração atual do dispositivo com o nome do diretório do recurso.

A figura 3.6 demonstra como uma coleção de recursos padrões de um aplicativo são aplicados para dois dispositivos diferentes quando não há recursos alternativos disponíveis.

Figura 3.6: Dois diferentes dispositivos, ambos utilizando recursos padrão. (ANDROID, 2011c)

A figura 3.7 mostra a mesma aplicação com um conjunto de recursos alternativos que se qualificam para uma das configurações do dispositivo, assim, os dois dispositivos utilizam recursos distintos.

Existem vários tipos de recursos que podem ser utilizados no diretório res/ (ANDROID, 2011c). São eles:

- Recursos de animação: colocados em res/anim/ para Animações *Tween* (entre dois *frames*) e res/drawable/ para animações de *frame*;

- Recursos de cores: define uma cor que muda, baseado no estado da *View*. São colocados no diretório res/color/;

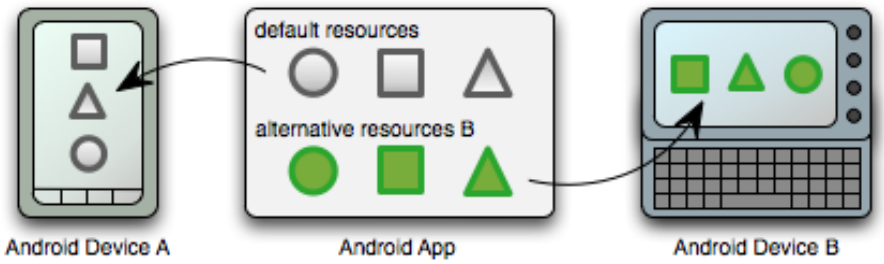

Figura 3.7: Dois diferentes dispositivos, um utilizando recursos alternativos. (ANDROID, 2011c)

- Recursos de imagens: define várias imagens com *bitmaps* ou XML. São colocados em res/drawable/;
- Recursos de *layout*: define o *layout* para a interface do aplicativo. São colocados em res/layout/;
- Recursos de menus: define o conteúdo dos menus do aplicativo. São colocados no diretório `res/menu/`;
- Recursos de *strings*: define *strings*, vetor de *strings*, e plurais (inclui formatação da *string* e estilo). São salvos em res/values/;
- Recursos de estilos: define a aparência e formato dos elementos da interface. São salvos em res/values/;
- Outros tipos de recursos: definem valores tais como *boolean*, inteiro, dimensões, cores, e outros vetores. São armazenados em res/values/.

Para acessar estes recursos, uma vez que eles já estão criados no projeto da aplicação, pode-se referenciar ao seu identificador (ID). Todos IDs dos recursos são definidos na classe R do projeto. Estes IDs são criados automaticamente pela ferramenta aapt que o SDK do Android fornece.

Quando a aplicação é compilada, a ferramenta aapt gera a classe R, a qual contém os IDs de todos recursos que estão no diretório res/.

O ID de um recurso é sempre composto de um tipo, tal como *string*, *drawable*, ou *layout*, e um nome, que é o nome do arquivo, sem a extensão do mesmo; ou o valor no atributo XML android:name, se o recurso é um simples valor, tal como uma *string*.

Existem duas maneiras de acessar um recurso. Pode ser via código, usando a classe R, como por exemploR.string.hello, onde *string* é um tipo de recurso e *hello* é o nome do recurso. Ou pode acessar via XML, usando uma sintaxe especial que também corresponde ao ID do recurso definido na classe R. O acesso via XML ficaria @string/hello, onde *string* é o tipo do recurso e *hello* é o nome do recurso. O arquivo XML referente ao recurso do tipo *string* e nome *hello* deve ficar semelhante a figura 3.8.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello!</string>
 </resources>
```

Figura 3.8: Exemplo de criação de recurso. (ANDROID, 2011c)

A partir deste arquivo XML da figura 3.8, pode-se usar o recurso de nome *hello* em um *layout* que configura o texto do componente <EditText> (vide figura 3.9) conforme o valor (*Hello*!) do recurso acionado por @string/hello. A utilização do recurso pode ser vista na figura 3.9.

```
<?xml version="1.0" encoding="utf-8"?>
<EditText xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:text="@string/hello" />
```

Figura 3.9: Exemplo de utilização de recurso. (ANDROID, 2011c)

Para acessar arquivos originais como um vídeo, áudio, ou uma fonte, é necessário salvá-los no diretório assets/, o qual não possui IDs, assim, não é possível acessar o recurso através da classe R e nem através de recursos XML. Em vez disso, é preciso utilizar a classe AssetManager. Para arquivos de áudio e vídeo é indicado utilizar o diretório res/raw/ e ler um *stream* de *bytes* usando o método openRawResource().

3.5 Considerações do Capítulo

Este capítulo detalhou como perfis, tanto de dispositivo como de usuário, são importantes para a personalização de interfaces de usuário. Também é detalhado neste capítulo quais os

dados relevantes do perfil do usuário para a adaptação da interface, além de apresentar o CC/PP e o vocabulário UAProf para descrição das capacidades de dispositivos. As linguagens descritivas de interface de usuário que são necessárias para modelar e definir o comportamento dos elementos da interface também foram detalhadas, a fim de mostrar o quanto é importante detalhar a interface do usuário separada da lógica de programação. Por fim, foi apresentada a linguagem descritiva de interface para Android, mostrando todas as características do mesmo e como a interface em XML é construída. A seguir serão discutidas questões relacionadas a ontologias, tais como formas de representação, componentes, vantagens de uso, ferramentas para criação de ontologias e consultas.

4 ONTOLOGIAS

A área de Ciência da Computação define, em geral, ontologia como uma especificação formal e explícita de uma conceituação compartilhada (GRUBER, 1993), (FENSEL, 2000), (GAVA; MENEZES, 2003), onde, segundo STUDER; BENJAMINS; FENSEL (1998), conceituação se refere ao modelo abstrato de algum fenômeno do mundo o qual identifica conceitos relevantes do próprio fenômeno; formal se refere ao fato da ontologia ser entendida pela máquina; e compartilhada dá a noção de que uma ontologia obtém o conhecimento apresentado não somente por um único indivíduo, mas por um grupo.

O uso de ontologias na computação já vem sendo estudado há vários anos. Inicialmente na área de Inteligência artificial e mais tarde em outras áreas da computação, as quais desejam manter conceitos relativos ao domínio da aplicação (GAVA; MENEZES, 2003). A importância de seu uso é devida à capacidade de representar hierarquias de classes de objetos (taxonomias) e seus relacionamentos (LIBRELOTTO et al., 2008a). Segundo GUARINO (1997), ontologia é uma caracterização axiomática do significado vocabulário lógico.

Apesar de existirem diferentes linguagens para expressar ontologias, elas possuem semelhanças em sua estrutura. Os elementos básicos de uma ontologia são três dessas semelhanças: as classes (conceitos), os indivíduos (instâncias das classes) e as propriedades, que podem ser atributos como “nome”, “idade”, “sexo”, ou relacionamentos como *possuiDeficiencia*, *possuiPreferencia* (DEAN; SCHREIJBER, 2004). Como exemplo de conceitos, pode-se utilizar: um usuário e uma deficiência, onde o usuário possui um relacionamento com uma deficiência, indicando que o conceito usuário se relaciona com o conceito deficiência através do relacionamento *possuiDeficiencia*.

Ontologias se diferem de outros modelos de dados pela sua preocupação com conceitos e seus relacionamentos, no qual a semântica destes relacionamentos é aplicada uniformemente (LIBRELOTTO et al., 2008b).

Esta seção descreve linguagens utilizadas para a construção de ontologias e o seu comportamento, bem como as vantagens de seu uso.

4.1 Representação de Ontologias

Várias linguagens para representação de ontologias existem, e algumas delas são (MCGUINNESS; HARMELEN, 2004): XML, a qual provê uma sintaxe para documentos estru-

turados, mas não possui restrição semântica no significado desses documentos; XML *Schema*, que é uma linguagem para restringir a estrutura de documentos XML, além de estender XML com tipos de dados; RDF, que é um modelo de dados para objetos e relações entre eles, e provê uma semântica simples para esse modelo de dados, e esses modelos de dados podem ser representados em XML; RDF *Schema*, o qual é um vocabulário para descrever propriedades e classes de recursos RDF, com uma semântica para generalização de hierarquias dessas propriedades e classes; e OWL, a qual adiciona um maior vocabulário para descrever propriedades e classes, por exemplo, relações entre classes (como disjunção), cardinalidade (como "exatamente um"), igualdade, tipos de propriedades ricas, características de propriedades e classes enumeradas.

Este trabalho utiliza e discute a OWL 1.0 (*Web Ontology Language*), uma recomendação W3C para *Web* Semântica (MCGUINNESS; HARMELEN, 2004). Uma versão mais atual da linguagem OWL existe, é a OWL 2.0, porém, para as necessidades deste trabalho a linguagem não alcançou um requisito, o qual foi não suportar a linguagem de consulta SQWRL, necessária para as consultas na ontologia deste trabalho (MOTIK, 2009). Esta seção irá discutir a linguagem em si e também citar as suas versões, apresentando a versão escolhida, mostrando os motivos da utilização da mesma.

A OWL é uma linguagem para definir e instanciar ontologias na *Web*. A linguagem OWL foi criada para aplicações que precisam processar o conteúdo da informação. Ela simplifica a interpretação por máquinas do conteúdo da *Web* do que XML, RDF e RDFS (RDF *Schema*) por fornecer vocabulário adicional com uma semântica formal. A linguagem OWL possui uma grande utilização e desenvolvimento, bem como ferramentas para criação, edição e processamento. A OWL foi desenvolvida para ser utilizada em aplicações nas quais é indispensável processar seu conteúdo e não apenas apresentá-lo aos humanos. Esses fatores levaram à escolha da OWL como linguagem para descrição de ontologias neste trabalho.

A linguagem OWL 1.0 possui três sub-linguagens:

- OWL-Lite: possui apenas algumas características da OWL e é voltada para usuários que necessitam de uma classificação hierárquica com simples restrições. Suporta apenas classificações hierárquicas e restrições simples, como cardinalidade, onde os valores permitidos são 0 ou 1;

- OWL-DL: suporta o máximo de expressividade com garantias de que todas as conclusões são computáveis e terminam em um tempo finito. A OWL-DL inclui todos os construtores da OWL, mas eles só podem ser usados sob certas restrições (por exemplo, enquanto uma

classe pode ser uma subclasse de muitas classes, uma classe não pode ser uma instância de outra classe);

- OWL-Full: é destinada a usuários que querem o máximo de expressividade e a liberdade sintática do RDF, sem garantias computacionais. Uma desvantagem é a de ter um maior custo de decisão, pois dá mais liberdade para o programador desenvolver o código, o que pode gerar maior complexidade ao ser processado pelo computador. A união de OWL com RDFS é suportada pela OWL-Full. Essa sub-linguagem não requer disjunção de classes, propriedades, indivíduos e valores de dados, isto é, uma ontologia criada em OWL-Full pode ter uma classe que além de ser classe, pode ser indivíduo (LIMA; CARVALHO, 2005).

Todos os construtores da OWL estão na OWL-Full, já a OWL-Lite é composta por um número bem menor de construtores do que a OWL-Full e contém algumas restrições referentes a descrições. A OWL-DL possui todos os construtores que a OWL-Full possui, porém com algumas restrições na utilização dos mesmos. Essas restrições em relação ao uso de alguns construtores é que fazem a OWL-DL ter computação em um número finito de tempo. Assim, quando se trata de processamento de inferências para raciocínio, a OWL-DL é a sub-linguagem mais apropriada, a qual será utilizada neste trabalho.

Houve um crescimento considerável de usuários que passaram a usar a linguagem OWL para o desenvolvimento de ontologias, alguns problemas e limitações passaram a ser evidenciados. Desta forma, em 2007, um grupo de pesquisa para resolver estas questões e trabalhar em uma nova versão da linguagem foi criado. Essa linguagem foi definida em 2008 como OWL 2. Alguns problemas de limitações da versão 1 da linguagem OWL foram resolvidos, além da sintaxe ter sido um pouco modificada com o objetivo de atingir novas funcionalidades.

A OWL 2 possui 3 sub-linguagens que oferecem vantagens importantes em cenários de aplicações específicas. Cada sub-linguagem é definida como um subconjunto de elementos que podem ser usados em uma ontologia. A sub-linguagem OWL 2 EL captura o poder expressivo usado em ontologias de grande escala, mas é uma sub-linguagem simples e com bom desempenho no tempo de inferência sobre ela. Por sua vez, a OWL 2 QL captura a expressividade utilizada em ontologias simples como *thesaurus* e a maior parte do poder expressivo de esquemas ER/UML. Esta sub-linguagem é mais adequada para aqueles que necessitam uma fácil comunicação com bancos de dados relacionais e onde o raciocínio sobre grandes conjuntos de dados é o mais importante. Por último, a OWL 2 RL foi projetada para acomodar aplicações

OWL 2 que necessitam usar toda a expressividade da linguagem com mais eficiência e em aplicações RDF(S) que precisam adicionar a expressividade da OWL 2. As três sub-linguagens possuem restrições sintáticas sobre determinados construtores e axiomas. As versões QL e RL são adequadas para aplicações onde ontologias relativamente leves são usadas com grandes conjuntos de dados. A escolha de qual utilizar vai depender dos tipos de dados a serem processados. Quando for necessário acessar dados diretamente através de consultas SQL, a sub-linguagem OWL 2 QL é a mais indicada, enquanto a OWL 2 RL é voltada para quem precisa operar sobre dados na forma de triplas RDF.

4.2 Componentes de Ontologias

Os elementos mais comuns em uma ontologia são classes, propriedades, instâncias de classes, e relações entre as instâncias (SMITH; WELTY; MCGUINNESS, 2004). Esta seção apresenta os elementos básicos de uma ontologia, segundo GRUBER (1993) e NOY; MCGUINNESS (2001):

- Classes: conceito básico de uma ontologia. Definem um determinado objeto no mundo. Ex: Pessoa, Automóvel, Animal, etc;
- Indivíduos: objetos ou instâncias de uma classe. Ex: Indivíduo elefante que pertence a classe Animal;
- Atributos: propriedades que classes e indivíduos podem possuir, como cor, quantidade de rodas, potência do motorde um automóvel;
- Relacionamentos: descrevem como as classes e indivíduos se relacionam. Ex: Pessoa possui Automovel;
- Axiomas: são regras lógicas pertinentes ao domínio em questão que evidenciam um fato. Um exemplo de axioma é afirmar que toda pessoa possui uma mãe. Conhecimentos que não estão explícitos na taxonomia da ontologia podem ser inferidos por axiomas.

4.3 Vantagens do Uso de Ontologias

Após conceituar ontologias, podemos apresentar uma lista com as principais vantagens da utilização de ontologias segundo GUIMARÃES (2002):

- Ontologias fornecem um vocabulário para representação do conhecimento. Esse vocabulário tem por trás uma conceitualização que o sustenta, evitando assim interpretações ambíguas;

- Ontologias permitem o compartilhamento de conhecimento. Sendo assim, caso exista uma ontologia que modele adequadamente certo domínio de conhecimento, essa pode ser compartilhada e usada por pessoas que desenvolvam aplicações dentro desse domínio. Para exemplificar, considere que exista uma ontologia para o domínio de livrarias. Uma vez que essa ontologia está disponível, várias livrarias podem construir seus catálogos usando o vocabulário fornecido por essa ontologia sem a necessidade de refazer uma análise do domínio de livraria;

- Fornece uma descrição exata do conhecimento. Diferentemente da linguagem natural em que as palavras podem ter semântica totalmente diferente conforme o seu contexto, a ontologia por ser escrita em linguagem formal, não deixa espaço para o *gap* semântico existente na linguagem natural. Por exemplo, quando uma pessoa fala para outra a palavra "Globo" ela pode estar querendo falar a respeito de um corpo esférico, como também de um canal de televisão brasileiro. A interpretação da palavra pode ser atribuída a um conceito ou outro conforme o estado mental do indivíduo;

- Porém, se há uma conceitualização comum entre essas duas pessoas a possibilidade de mal entendido diminui muito. Por exemplo, se essas pessoas concordam em uma ontologia sobre o domínio de formas geométricas, possivelmente não haverá mal entendido;

- É possível fazer o mapeamento da linguagem da ontologia sem que com isso seja alterada a sua conceitualização, ou seja, uma mesma conceitualização pode ser expressa em várias línguas;

- Pode ser possível estender o uso de uma ontologia genérica de forma a que ela se adeque a um domínio específico. Por exemplo, se alguém precisa de uma ontologia sobre bicicletas para construir uma aplicação e só encontra uma ontologia sobre o domínio genérico de veículos, pode utilizar essa ontologia estendendo-a para o domínio específico da aplicação, que no caso são de bicicletas.

Segundo USCHOLD; GRUNINGER (1996), outra vantagem importante é a necessidade de confiabilidade junto aos conceitos de um vocabulário que descreve um domínio, visto que a

formalização do conhecimento representado possibilita a automação da verificação de consistência, tornando ambientes mais confiáveis.

4.4 Ferramentas para Processamento de Ontologias

Descrever uma ontologia bem fundamentada não é uma tarefa simples, porém, se bem descrita, ela se torna de grande utilidade para sistemas que a utilizam. Entretanto, além da necessidade de ter uma ontologia bem descrita, também é necessário processar esta ontologia, visando inferir resultados a partir de conhecimentos expostos na mesma.

Existem algumas ferramentas para criação e manipulação de ontologias. A mais utilizada até o momento é o projeto Protégé. Esta ferramenta será utilizada neste trabalho por ter um grande número de usuários e por ter uma boa documentação, além de disponibilizar uma API Java para manipulação de ontologias. A ferramenta Protégé suporta editar e visualizar classes, propriedades e regras SWRL (*Semantic Web RuleLanguage*) (HORROCKS et al., 2004), executar motores de inferência, editar indivíduos OWL, carregar e salvar ontologias OWL e RDF, e definir características de classes como expressões OWL.

A parte gráfica, além de suportar o que já foi citado, habilita o usuário a configurar a ferramenta como desejar, adicionando tabs para vários *plug-ins* (regras SWRL, visualizações, etc.), bem como oferece funções *built-in* para criação de expressões e restrições da ontologia.

Como citado anteriormente, o projeto Protégé disponibiliza uma API Java para manipular ontologias. Através desta API é possível utilizar motores de inferência, desta forma é possível processar regras e consultar dados na ontologia. Existem alguns motores de inferência disponíveis no Protégé, e neste trabalho será utilizado o Jess (JESS, 2011), o qual é um dos motores de inferência que possibilita o processamento de regras e consultas na ontologia.

A seções 4.5.1 e 4.5.2 detalharão as regras SWRL e consultas SQWRL respectivamente.

4.5 Inferências em Ontologias

A utilização de inferências sobre ontologias permite a resolução de problemas baseados em conhecimentos antecipados, ou seja, através de um conhecimento pré-estabelecido na ontologia, é capaz de deduzir resultados. Ao utilizar inferências, o poder de expressividade das ontologias aumentam.

Uma inferência recebe como entrada uma parte do conhecimento do domínio em questão e gera uma saída, a qual é uma transformação desse conhecimento (JUNIOR, 2003).

4.5.1 Regras SWRL (*Semantic Web Rule Language*)

Existem ontologias que não conseguem expressar determinados fatos por si só. Essas ontologias dependem de motores de inferência para poder deduzir resultados que não estão descritos na ontologia.

Em uma ontologia OWL, é possível descrever, por exemplo, que Sócrates *possuiIrmão* Raí, de forma fixa, onde Sócrates e Raí pertencem a classe Pessoa. Para inferir algo genérico sobre a relação *possuiIrmão* para qualquer indivíduo da classe Pessoa, é necessário a utilização de regras SWRL. Estas regras são capazes de inferir conhecimentos sobre indivíduos na ontologia. Vejamos um exemplo de regra utilizando a relação *possuiIrmão* citada acima, onde qualquer indivíduo da classe pessoa pode possuir um irmão, basta apenas possuir a mesma mãe. Para inferir o exemplo acima, pode-se utilizar a seguinte regra:

```
Pessoa(?p) ^ possuiMae(?p, ?m) ^ Pessoa(?p2) ^ possuiMae(?p2, ?m) ^ differentFrom(?p, ?p2) -> possuiIrmao(?p, ?p2)
```

A regra SWRL acima infere que o indivíduo “p” da classe Pessoa que possui uma mãe “m” também da classe Pessoa, e o indivíduo “p2” da classe Pessoa que possui uma mãe “m” também da classe Pessoa, ou seja, mesma mãe de “p”, são irmãos. Desta forma, será adicionada uma relação *possuiIrmao* entre os indivíduos “p” e “p2” na ontologia. A clausula “differentFrom(?p, ?p2)” serve para não ocorrer ambiguidades, ou seja, para que o indivíduo “p” não seja o mesmo que “p2”. Além de adicionar esta clausula OWL na regra SWRL, também é necessário adicionar a clausula *OWL:AllDifferents* nos indivíduos da classe Pessoa, assim, fica claro que os indivíduos da classe Pessoa não são os mesmos.

Caso esta regra fosse utilizada em uma ontologia que descrevesse indivíduos que tem a mesma mãe, os mesmos seriam irmãos e teriam a propriedade *possuiIrmao* referenciada, relacionando os indivíduos na ontologia.

Além das clausulas mostradas no exemplo acima, existem funções *built-in* incorporadas a SWRL. Algumas delas são:

- swrlb:equal(arg1, arg2), satisfeito quando arg1 e arg2 são iguais;
- swrlb:notEqual(arg1, arg2), satisfeito quando arg1 e arg2 são diferentes;
- swrlb:greaterThan(arg1, arg2), satisfeito quando arg1 é maior que arg2;

- swrlb:lessThan(arg1, arg2), satisfeito quando arg1 é menor que arg2.

Portanto, existem várias funções embutidas no SWRL para comparações, cálculos matemáticos, valores booleanos, manipulação de strings, data e tempo, etc. Estas funções tem o objetivo de facilitar a criação e manuseio de regras SWRL. Para acessar uma lista completa das funções *built-in* disponíveis, visite http://www.w3.org/Submission/SWRL/#8.

4.5.2 SQWRL (*Semantic Query-Enhanced Web Rule Language*)

A linguagem SQWRL é baseada na SWRL e tem o propósito de possibilitar consultas em ontologias OWL (SQWRL, 2011). Ela fornece operações estilo SQL para recuperar conhecimento de ontologias OWL.

De forma distinta a linguagem SWRL, onde os resultados das inferências são aplicados a ontologia em questão, SQWRL somente retorna resultados que satisfazem a consulta elaborada, e, através dessa consulta, é possível tratar os resultados utilizando uma linguagem de programação. Neste trabalho usou-se a linguagem de programação Java. Para explicar melhor como a SQWRL funciona, utilizamos a regra definida anteriormente na seção 4.5.1, onde inferimos que dois indivíduos que possuem mesma mãe são irmãos. A consulta em SQWRL para o exemplo anterior deve ser:

```
Pessoa(?p) ^ possuiMae(?p, ?m) ^ Pessoa(?p2) ^ possuiMae(?p2, ?m) -> sqwrl:selectDistinct(?p)
```

Desta forma, a consulta retornará todos os indivíduos que possuem a mesma mãe “m”, ou seja, todos os irmãos. A clausula sqwrl:selectDistinct(?p) busca os resultados da consulta sem duplicações.

Outra forma de executar a mesma consulta é a ontologia possuir a regra SWRL, que indica que dois indivíduos que possuem a mesma mãe são irmãos. Logo, a consulta poderia ser feita diretamente à propriedade inferida (*possuiIrmao*), da seguinte maneira:

```
possuiIrmao(?p, ?p2) -> sqwrl:selectDistinct(?p)
```

As consultas SQWRL também podem conter outros operadores, os quais pertencem à biblioteca de *built-ins* da SWRL. Alguns dos operadores possíveis são: sqwrl:orderBy, sqwrl:min, sqwrl:max, sqwrl:avg, sqwrl:groupBy, sqwrl:union, etc. A lista completa pode ser encontrada na ontologia SQWRL em http://swrl.stanford.edu/ontologies/built-ins/3.4/sqwrl.owl. Além dos

operadores *built-in* citados, também pode ser utilizado operadores SWRL nas consultas SQWRL. Um exemplo de uso de operadores SWRL em consultas poderia ser: Retorne as pessoas que possuem irmãos(ãs) e são do sexo masculino. A consulta SQWRL seria:

```
Pessoa(?p) ^ possuiMae(?p, ?m) ^ Pessoa(?p2) ^ possuiMae(?p2,
?m) ^ sexo(?p, ?s) ^ sexo(?p2, ?s) ^ swrlb:equal(?s, "masculino")
-> sqwrl:selectDistinct(?p)
```

Onde swrlb:equal(?s, “masculino”) pertence aos operadores SWRL e sua função é testar se ?s é igual a “masculino”.

4.6 Considerações do Capítulo

Este capítulo apresentou questões sobre a utilização de ontologias como forma de representação do conhecimento de um domínio de aplicação, componentes, vantagens de uso, e linguagens para realização de inferências e consultas. Como a linguagem OWL possui um poder de expressividade maior que as outras linguagens e é uma recomendação W3C, pode-se concluir que ela é a linguagem adequada para estes fins. As linguagens mais indicadas para manipulação de ontologias são SWRL e SQWRL, uma vez que foram desenvolvidas para serem aplicadas específicamente em ontologias OWL, podendo assim trabalhar com toda expressividade existente na mesma. No capítulo a seguir é apresentada a ontologia desenvolvida para este trabalho, a qual representa o conhecimento existente em um perfil de usuário, a fim de detalhar o perfil para adaptar a interface conforme suas necessidades e preferências.

5 UMA ONTOLOGIA PARA REPRESENTAÇÃO DO CONHECIMENTO DO DOMÍNIO DE PERFIL DO USUÁRIO

Este capítulo apresenta a ontologia desenvolvida para descrever perfis de usuários. A ontologia que descreve o perfil do usuário trata de dados que são relevantes à personalização da interface do usuário, como cor de um botão, tamanho da fonte, etc. As informações referentes ao usuário estão armazenadas na ontologia e foram descritas na seção 3.2.

A ontologia foi desenvolvida na ferramenta Protégé versão 3.4.4 (KNUBLAUCH et al., 2004) utilizando a linguagem OWL-DL pelo seu poder de expressividade e garantia de que suas computações terminam em tempo finito. A escolha pela versão 3.4.4 do Protégé se deu pelo fato do grande número de usuários e excelente documentação. As versões mais novas da ferramenta (4 em diante) ainda não suportam consultas SQWRL, portanto não satisfazem as condições deste trabalho. A não utilização da linguagem OWL 2 vêm da necessidade de utilizar consultas SQWRL, o que não é suportado pela versão atual da linguagem.

5.1 Classes OWL

A partir do estudo relacionado às informações pertinentes de perfis de usuários para personalização de interfaces, foi necessária a criação de uma hierarquia de classes, baseada no estudo levantado no capítulo 3, seção 3.2, para que elas sejam bem estruturadas e consigam se relacionar umas com as outras.

Existem algumas classes abstratas criadas na ontologia com o intuito de organizar a hierarquia de uma forma melhor, fazendo com que a relação entre as classes seja mais clara.

As superclasses da ontologia são *Cultura, Deficiencia, Experiencia, MembroCorpoHumano, Preferencia* e *Usuario*. Dentre essas classes, *Deficiencia, Experiencia* e *MembroCorpoHumano* são classes abstratas, pois não possuem nenhuma instância relacionada a elas. Elas servem apenas para organizar a hierarquia da ontologia. A superclasse *Deficiencia* possui mais quatro subclasses abstratas, que são *DeficienciaAuditiva, DeficienciaCognitiva, DeficienciaMotora* e *DeficienciaVisual*. Cada uma dessas quatro classes, exceto *DeficienciaCognitiva*, possui algumas subclasses. A classe *Experiencia* possui três subclasses nomeadas *ExpEspecialista, ExpNovato* e *ExpOcasional*. Por sua vez, a classe *MembroCorpoHumano* possui as subclasses *Cabeca, Mao* e *Perna*. A figura 5.1 mostra a hierarquia de classes da ontologia. As descrições de cada classe podem ser vistas no capítulo 4, onde estão bem detalhadas.

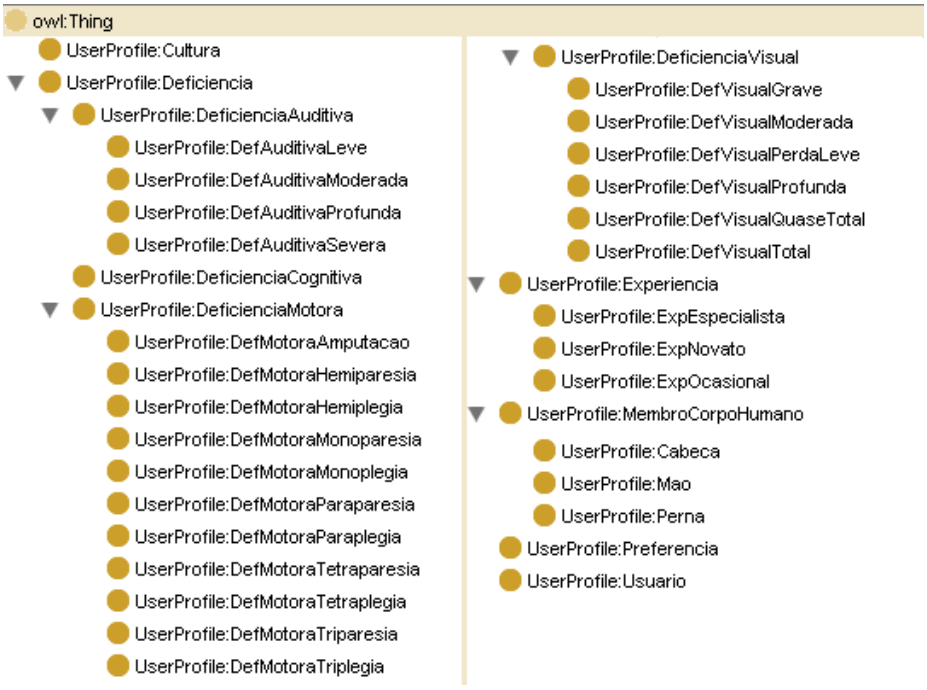

Figura 5.1: Hierarquia das classes da ontologia.

Existem dois tipos de propriedades em ontologias, as quais são relacionadas às classes, propriedades de dado e propriedades de objeto. As propriedades de dado são as características que os indíviduos de cada classe possuem e as propriedades de objeto são os relacionamentos existentes entre os indivíduos de cada classe.

5.2 Propriedades de Dado (*Datatype Properties*)

As propriedades de dado de uma classe são características que elas possuem para descrevê-las, diferenciando uma das outras. A função das propriedades é identificar as classes e também definir ações que possam ser exercidas pelas mesmas (LIBRELOTTO et al., 2010). As propriedades existentes na ontologia de perfil do usuário, bem como seus tipos e valores, estão descritas na tabela 5.1.

A tabela 5.1 mostra todas as propriedades de dados existentes na ontologia Perfil do Usuário com seus tipos de dados e valores que podem ser atribuídos a essas propriedades. Um exemplo é a propriedade *sexo* da classe *Usuario*, a qual é do tipo *boolean* e pode ter valores como "M"de masculino ou "F"de feminino.

Tabela 5.1: Tabela de propriedades de dado da Ontologia Perfil do Usuário

Tabela de propriedades de dado da Ontologia Perfil do Usuário			
Classe	**Propriedade**	**Tipo de dado**	**Valores**
Usuario	idade	inteiro	-
	nacionalidade	String	brasileiro, alemão, inglês, etc.
	sexo	String	M (Masculino) ou F (Feminino)
Preferencia	corPreferidaBackground	String	amarelo, branco, preto, azul, etc.
	corPreferidaTexto	String	amarelo, branco, preto, azul, etc.
	estiloTexto	String	bold, italic, normal, ou boldlitalic
	idioma	String	inglês, português, alemão, espanhol, etc.
	fontePreferida	String	calibri.tff, arial.ttf, etc.
	somenteImagem	Boolean	True ou False
	somenteTexto	Boolean	True ou False
Cabeca	numeroOlhos	Inteiro	0, 1, ou 2
	numeroOuvidos	Inteiro	0, 1, ou 2
Mao	numeroDedosMaoDireita	Inteiro	0, 1, 2, 3, 4, ou 5
	numeroDedosMaoEsquerda	Inteiro	0, 1, 2, 3, 4, ou 5
	temPolegarMaoDireita	Boolean	True ou False
	temPolegarMaoEsquerda	Boolean	True ou False
Perna	temPernaDireita	Boolean	True ou False
Perna	temPernaEsquerda	Boolean	True ou False
DefVisualGrave	grauDefVisualGrave	Float	20/200 a 20/400
DefVisualModerada	grauDefVisualModerada	Float	20/70 a 20/160
DefVisualPerdaLeve	grauDefVisualPerdaLeve	Float	20/30 a 20/60
DefVisualProfunda	grauDefVisualProfunda	Float	20/500 a 20/1000
DefVisualQuaseTotal	grauDefVisualQuaseTotal	Float	Menor que 20/1000
DefVisualTotal	grauDefVisualTotal	Inteiro	0
DefAuditivaLeve	nivelDefAuditivaLeve	Inteiro	1
DefAuditivaModerada	nivelDefAuditivaModerada	Inteiro	2
DefAuditivaSevera	nivelDefAuditivaSevera	Inteiro	3
DefAuditivaProfunda	nivelDefAuditivaProfunda	Inteiro	4

5.3 Propriedades de Objetos (*Object Properties*)

As propriedades de objetos em uma ontologia tem o papel de relacionar o indivíduo de uma classe com outra, fazendo com que as relações dêem um significado para a ontologia. As relações existentes na ontologia perfil do usuário estão descritas na tabela 5.2.

Além da tabela, para ficar mais visível os relacionamentos entre as classes da ontologia, a figura 5.2 mostra um grafo das classes e seus relacionamentos criado através do *plug-in* Jambalaya disponível para o Protégé.

O grafo da figura 5.2 mostra como as classes se relacionam entre si. Um exemplo das relações mostradas na figura é a relação entre a classe *Usuario possuiDeficiencia* com a classe

Tabela 5.2: Tabela de relacionamentos entre classes da Ontologia Perfil do Usuário

Tabela de relacionamentos entre classes da Ontologia Perfil do Usuário		
Classe	**Relacionamento**	**Classe**
Usuario	possuiCultura	Cultura
Usuario	possuiDeficiencia	Deficiencia
Usuario	possuiExperiencia	Experiencia
Usuario	possuiMembroCorpoHumano	MembroCorpoHumano
Usuario	possuiPreferencia	Preferencia
MembroCorpoHumano	possuiDeficiencia	Deficiencia

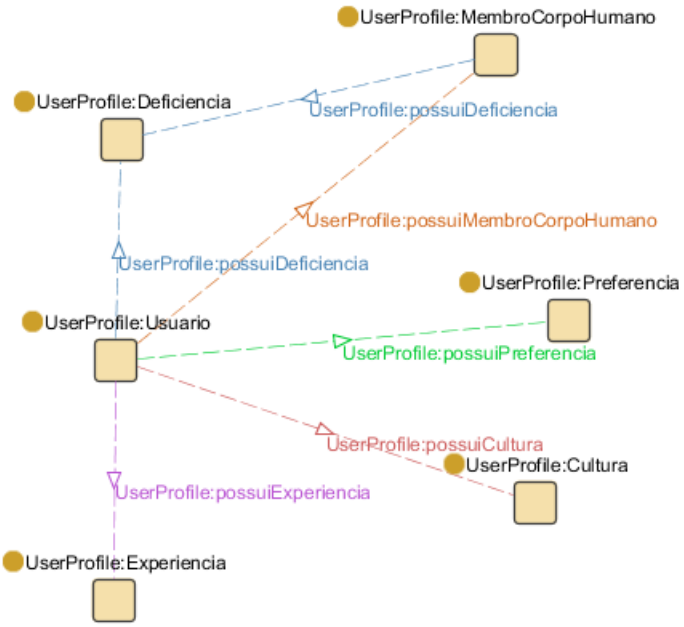

Figura 5.2: Grafo das classes da ontologia.

Deficiencia e a classe *MembroCorpoHumano possuiDeficiencia* com a classe *Deficiencia*. Na imagem do grafo fica mais clara essa relação, onde pode-se identificar qual o membro do corpo humano que possui determinada deficiência.

5.4 Instâncias (*Individuals*)

Instâncias em uma ontologia representam objetos do domínio em questão. Na ontologia, as instâncias possuem atributos (propriedades de dado) e comportamento (propriedades de objetos).

Cada usuário que está descrito na ontologia de perfil do usuário é uma instância, e toda instância do mesmo tipo deve ser colocada na mesma classe. Por exemplo, uma instância poderia ser uma pessoa de nome João, a qual possui uma idade, nacionalidade, sexo, talvez possua al-

guma(s) deficiência(s), preferência(s), etc, e está na classe *Usuario*, assim como poderia existir uma instância da classe *Deficiencia*, a qual deve possuir seus atributos e comportamentos. Um exemplo de instâncias e seus relacionamentos pode ser vista no grafo da figura 5.3.

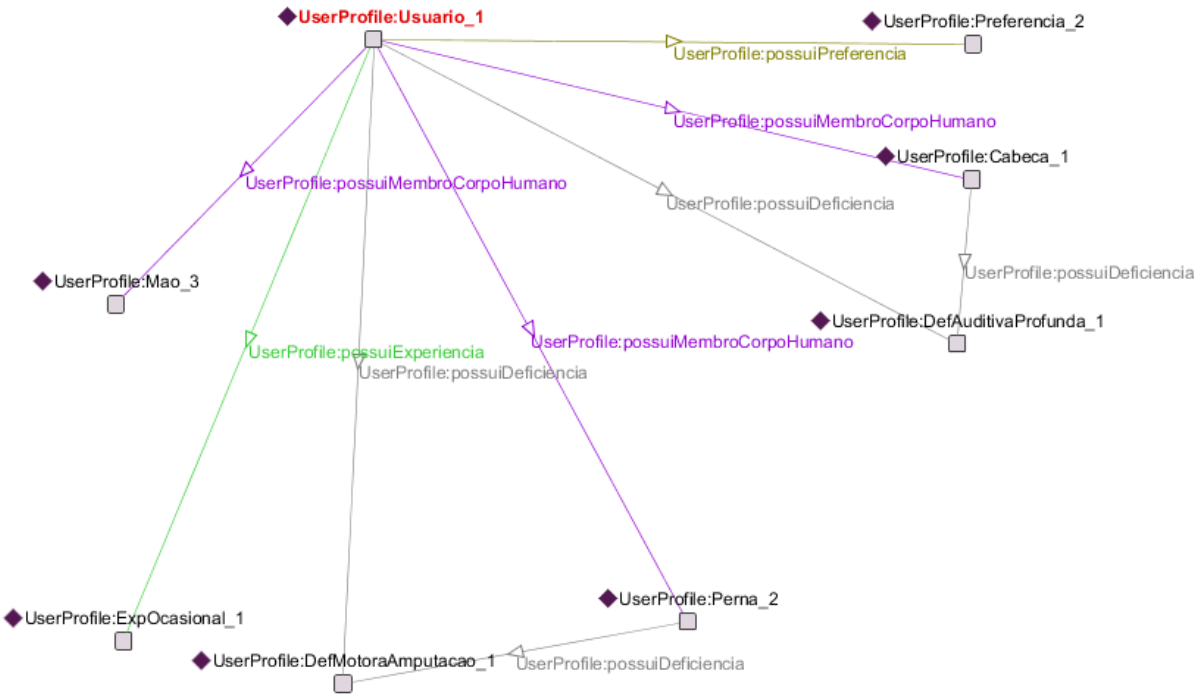

Figura 5.3: Exemplo de grafo de instâncias e seus relacionamentos.

Como pode ser visto na figura 5.3, o indivíduo Usuario_1, da classe *Usuario* possui várias relações. A tabela 5.3 a seguir descreve as mesmas.

Tabela 5.3: Tabela de relacionamentos entre instâncias da Ontologia Perfil do Usuário

Tabela de relacionamentos entre instâncias da Ontologia Perfil do Usuário		
Instância	**Relacionamento**	**Instância**
Usuario_1	possuiPreferencia	Preferencia_2
Usuario_1	possuiMembroCorpoHumano	Cabeca_1
Usuario_1	possuiDeficiencia	DefAuditivaProfunda_1
Usuario_1	possuiMembroCorpoHumano	Perna_2
Usuario_1	possuiDeficiencia	DefMotoraAmputacao_1
Usuario_1	possuiExperiencia	ExpOcasional_1
Usuario_1	possuiMembroCorpoHumano	Mao_3
Perna_2	possuiDeficiencia	DefMotoraAmputacao_1
Cabeca_1	possuiDeficiencia	DefAuditivaProfunda_1

Algumas instâncias foram criadas na ontologia de forma a executar alguns testes, mas a forma como a ontologia é atualizada e populada será vista na Seção 6.5 de forma mais detalhada.

5.5 Consultas SQWRL

Consultas em ontologias são feitas para poder obter dados relevantes ao domínio da ontologia. Por exemplo, para obter a cor preferida de um determinado usuário, usa-se consultas.

As consultas na ontologia deste trabalho foram feitas através da linguagem SQWRL a partir de uma API Java disponibilizada pelos desenvolvedores da ferramenta Protégé. A classe `SQWRLPerfilUsuario.java` apresentada na figura 6.1 da abordagem proposta é a responsável por executar essas consultas na ontologia de perfil do usuário.

Nesta classe `SQWRLPerfilUsuario.java` estão implementados métodos para capturar informações pertinentes à determinado usuário que os sensores do ambiente identificaram. Métodos como `getCulture()`, `getUserInfo()`, `getUserExperience()`, `getPreferences()`, `getImpairments()`, e `getHumanBodyMember()`. A figura 5.4 mostra o código do método `getUserExperience()`.

```
public SQWRLResultValue getUserExperience(String pUser) {
   try{

       OWLModel owlModel = ProtegeOWL.createJenaOWLModelFromURI(path);

       SWRLFactory swrlfactory = new SWRLFactory(owlModel);

       swrlfactory.createImp("consultaUserExperience", "UserProfile:Experiencia(?e) ^" +
       " UserProfile:possuiExperiencia("+pUser+", ?e)" + " -> sqwrl:select(?e)");

       SQWRLQueryEngine queryEngine = SQWRLQueryEngineFactory.create(owlModel);

        SQWRLResult resultadoExperiencia = queryEngine.runSQWRLQuery("consultaUserExperience");
        return resultadoExperiencia.getValue(0);
   }
   catch (Exception e) {return null;}
}
```

Figura 5.4: Código Java do método `getUserExperience()`.

Na Figura 5.4 pode-se notar que o método `getUserExperience()` primeiramente cria um modelo OWL com o *path* que é o caminho da ontologia perfil do usuário. Logo após é criada uma fábrica de SWRL, para poder criar as consultas e executá-las. Após criar a fábrica de SWRL sobre o modelo OWL, é necessário criar a consulta que se deseja, chamando o método `createImp()` e passando como primeiro parâmetro o nome da consulta que deseja e como segundo parâmetro, a consulta. Por fim, só basta executar a consulta chamando o método `runSQWRLQuery()` e atribuir o resultado a variável do tipo `SQWRLResult` de nome `resultadoExperiencia`. A consulta da figura 5.4 busca a experiência de um determinado usuário (`pUser`) passado como parâmetro para o método `getUserExperience()`.

5.6 Inferências

Este trabalho utiliza inferências sobre a ontologia através da linguagem SWRL. As inferências são necessárias para deduzirem novos fatos a partir de fatos pré-estabelecidos na ontologia.

Um exemplo no qual é utilizado inferência sobre a ontologia deste trabalho é quando um usuário possui um membro do corpo humano amputado, porém, na ontologia, os fatos pré-estabelecidos são que o usuário possui menos de cinco dedos em uma mão. Assim, logo após a execução da regra, pode-se inferir que determinado usuário, por não possuir os cinco dedos em uma das mãos, possui um membro amputado, e ainda pode-se descobrir qual é o membro que está amputado, pelo simples fato da relação entre a classe MembroCorpoHumano e Deficiencia. A regra que demonstra esse caso está na figura 5.5 a seguir:

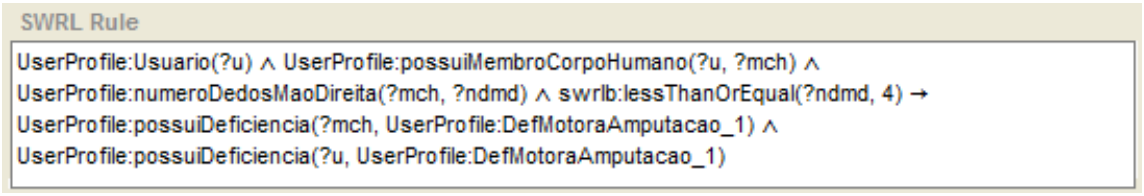

Figura 5.5: Exemplo de regra SWRL na ferramenta Protégé

A regra demonstrada na figura 5.5 atribui ao usuário e membro do corpo humano que satisfazer a condição de possuir quatro dedos ou menos (`swrlb:lessthanOrEqual(?ndmd,4)`), a instância de nome DefMotoraAmputacao_1, da classe DefMotoraAmputacao, na relação *possuiDeficiencia*. Desta forma, a relação entre as classes Usuario e Deficiencia, bem como MembroCorpoHumano e Deficiencia agora possuem a deficiência DefMotoraAmputacao_1.

5.7 Considerações do Capítulo

Este capítulo detalhou a ontologia criada para descrever o domínio de perfil do usuário, descrevendo em detalhes as classes, indivíduos, relacionamentos e atributos relacionados ao perfil do usuário. Também foram descritas algumas consultas e inferências realizadas sobre a ontologia através das linguagens SQWRL e SWRL, a fim de inferir resultados e executar consultas sobre o domínio da ontologia para torná-la fiel ao domínio que ela representa.

A ontologia faz parte da metodologia proposta pelo trabalho e é de fundamental importância para a personalização da interface de usuário, visto que os dados descritos na ontologia descrevem dados relacionados às preferências e necessidades do usuário e são dados importantes para o fluxo da arquitetura proposta, a qual será vista no capítulo seguinte.

6 MODELAGEM DE UMA ARQUITETURA PARA PERSONALIZAÇÃO DE INTERFACES DE USUÁRIO PARA DISPOSITIVOS MÓVEIS EM AMBIENTES PERVASIVOS

Este capítulo aborda a modelagem de uma arquitetura para personalizar interfaces de usuário para dispositivos de computação móvel em ambientes pervasivos. A ontologia do capítulo 5 servirá para o detalhamento desta arquitetura.

6.1 Arquitetura PIDIM (Personalização de Interfaces para DIspositivos Móveis)

O objetivo principal da arquitetura é adaptar interfaces de usuário para dispositivos móveis baseando-se no perfil do usuário, através do uso de ontologias, capacidades do dispositivo, utilizando um vocabulário UAProf, e o contexto do ambiente, o qual é captado por sensores. Assim, este capítulo tem por meta descrever a arquitetura para personalização de interfaces de usuário.

Na arquitetura proposta existem dois domínios, os quais comunicam-se entre eles. O primeiro domínio se refere ao aplicativo, lado cliente da arquitetura, o qual contém a parte lógica da aplicação móvel, e também a linguagem descritiva de interface de usuário. O segundo domínio, servidor, possui a ontologia que descreve o perfil de usuário, apresentada no capítulo 5.

Além da ontologia, o lado servidor da arquitetura também retém três módulos que buscam dados relativos ao perfil do usuário (SQWRLPerfilUsuario), sensores do ambiente (Sensores – Seção 6.2), e repositório de perfis UAProf dos dispositivos (*UAPROFDispositivo* – Seção 6.3). Ainda no lado servidor, existe um outro módulo que possui o papel de executar regras com todos os dados oriundos dos módulos SQWRLPerfilUsuario, Sensor e UAPROFDispositivo, a fim de personalizar a interface final do usuário (Seção 6.4).

Também existe um módulo para atualizar a ontologia (*AtualizaOntologia* – Seção 6.6) no lado servidor da arquitetura, o qual recebe o usuário captado pelos sensores e o resultado de testes de deficiências e preferências do usuário para atualizar na ontologia perfil do usuário. Uma cache para armazenar os documentos sobre os perfis de dispositivos também está no servidor.

A figura 6.1 apresenta a arquitetura para alcançar este objetivo. A partir disto, o fluxo da arquitetura tem o seu funcionamento dividido em 14 passos:

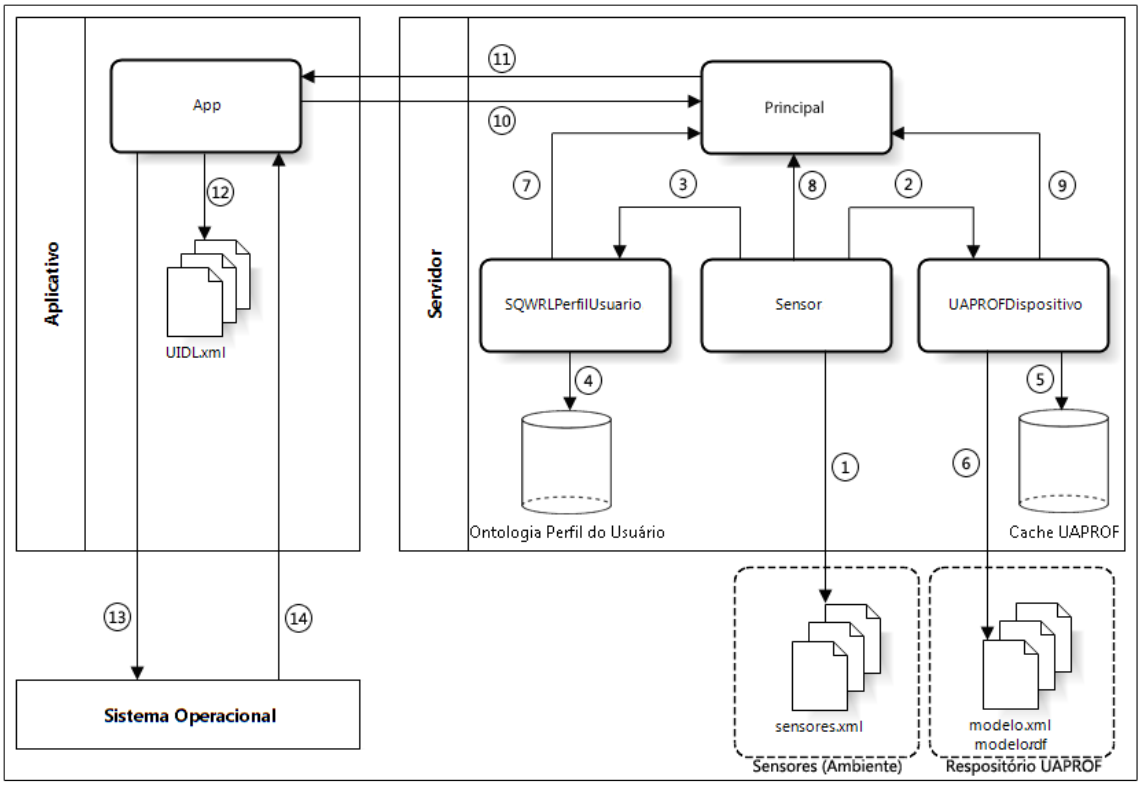

Figura 6.1: Arquitetura PIDIM

1. o módulo Sensor busca os documentos XML criados por um serviço que se comunica com os sensores do ambiente pervasivo. Esses documentos XML refletem as mudanças que ocorrem no ambiente. Desta forma, o módulo Sensor retém os dados de contexto, como dispositivos (modelo), localização de usuários e dispositivos, usuários presentes no ambiente, temperatura, luminosidade, ruído, etc;

2. o módulo Sensor passa os dados dos dispositivos (modelos) para o módulo UAPROFDispositivo;

3. o módulo Sensor passa o usuário que está no ambiente (a ser encontrado na ontologia) para o módulo SQWRLPerfilUsuario;

4. após processados os dados dos sensores, este passo faz com que o módulo SQWRLPerfilUsuario busque dados do usuário na ontologia perfil do usuário com a utilização da OWL API;

5. em seguida, o módulo UAPROFDispositivo tenta buscar o dispositivo (modelo) na cache UAProf.

6. caso o modelo não exista na cache, o módulo UAPROFDispositivo utiliza a classe HttpURLConnection do pacote java.net para buscar o documento .xml ou .rdf do modelo do dispositivo no repositório UAProf;

7. o módulo SQWRLPerfilUsuario passa os dados referentes ao usuário para o módulo Principal;

8. o mesmo faz o módulo UAPROFDispositivo, enviando os dados do dispositivo;

9. o módulo Sensor repassa o restante dos dados (luminosidade, ruído, temperatura, localização do usuário, etc.); Neste ponto, o módulo Principal executa regras cruzando todos os dados captados pelos sensores, perfil de usuário, e perfil de dispositivo, conforme apresentado na Seção 6.4;

10. o aplicativo móvel envia arquivos .xml, ou id e atributo referentes à interface do usuário para o módulo Principal modificar. Esse envio é realizado via http com a classe Java HttpURLConnection;

11. após receber os dados do lado cliente (aplicativo), para personalização da interface, o módulo Principal modifica os arquivos referentes à interface do usuário recebidos (ou valores de atributos xml) e devolve para o aplicativo;

12. o aplicativo atualiza os arquivos relativos à interface do usuário (UIDL.xml);

13. e requisita para o sistema operacional renderizar a interface gráfica do usuário;

14. por fim, o sistema operacional renderiza a interface final do usuário.

As seções seguintes descrevem cada componente da arquitetura proposta, a fim de esclarecer o papel de cada um no contexto deste trabalho.

6.2 Sensores

Em ambientes pervasivos, mudanças no contexto do ambiente ocorrem frequentemente. Essas mudanças devem ser monitoradas, a fim de fazer com que o sistema funcione corretamente.

Os sensores representam qualquer tipo de sensor que possa ser incluído no ambiente, como etiquetas RFID (RFID, 2011), sensores de luminosidade, umidade, temperatura, ruídos, localização de pessoas, localização de dispositivos, etc. Estes sensores devem ser capazes de captar

todos estes tipos de mudanças, para que então possam transformar as mudanças detectadas em informações pertinentes ao sistema.

A comunicação dos sensores de ambiente com o servidor é feita através da troca de arquivos XML, os quais são criados a partir da detecção de mudanças no ambiente.

No momento que o sensor do ambiente detectar alguma mudança no ambiente pervasivo, ele publica essa mudança de contexto e um serviço faz a conexão com o módulo Sensor, do servidor, a fim de fazer com que o módulo Sensor, do servidor, consiga acessar o documento XML referente às mudanças de contexto do ambiente pervasivo.

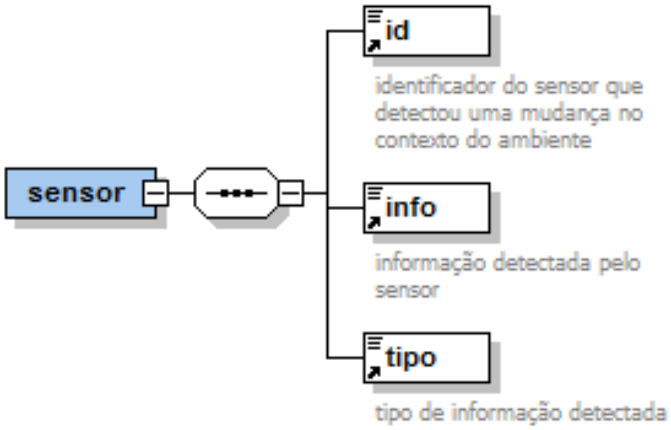

Figura 6.2: Esquema do arquivo XML que notifica uma alteração de contexto no ambiente.

A estrutura do documento XML que é enviado ao servidor é mostrado na figura 6.2. O esquema XML mostrado na figura 6.2 é formado por uma *tag* chamada id, a qual se refere ao identificador do sensor. Esta *tag* é obrigatória, visto que em um ambiente pervasivo, vários tipos de sensores podem estar presentes. Desta forma, um identificador único é necessário. A *tag* info, descreve a informação captada pelo sensor. Em um sensor que detecta a temperatura do ambiente, na *tag* info, seria armazenado o valor da temperatura, por exemplo, 28°C. A última *tag* tipo, refere-se ao tipo de informação que o sensor detecta, por exemplo, temperatura. Um exemplo de um arquivo XML que armazena informações sobre temperatura de um ambiente é mostrado na figura 6.3.

6.3 Repositório UAProf

O repositório UAProf armazena dados relacionados às características dos dispositivos. Este repositório é mantido por cada fabricante de dispositivos, como por exemplo a Nokia, HTC, Samsung, LG, Motorola, entre outros. Cada fabricante possui um repositório que contém os

```
<?xml version="1.0" encoding="UTF-8"?>
<sensor xmlns:xsi="sensor.xsd">
    <id>SR0217</id>
        <!-- código identificador do sensor -->
    <info>28ºC</info>
        <!-- atributo valor -->
    <tipo>Temperatura</tipo>
        <!-- Tipo de informação que o sensor detecta -->
</sensor>
```

Figura 6.3: Exemplo de informações detectadas por um sensor de temperatura.

arquivos XML ou RDF de cada modelo fabricado. Um exemplo de uma parte de um documento UAProf do modelo do dispositivo Samsung Galaxy S (GT-I9000) está demonstrado na figura 6.4.

```
- <prf:component>
   - <rdf:Description rdf:ID="HardwarePlatform">
        <rdf:type rdf:resource="http://www.openmobilealliance.org/tech/profiles/UAPROF/ccppschema-20021212#HardwarePlatform"/>
      - <prf:BluetoothProfile>
         - <rdf:Bag>
              <rdf:li>General Access</rdf:li>
              <rdf:li>Handsfree</rdf:li>
              <rdf:li>Headset</rdf:li>
              <rdf:li>Service Discovery</rdf:li>
              <rdf:li>File Transfer</rdf:li>
              <rdf:li>Object Push</rdf:li>
              <rdf:li>Dial-Up Networking</rdf:li>
              <rdf:li>Phone Book Access</rdf:li>
              <rdf:li>Basic Printing</rdf:li>
              <rdf:li>Sim Access</rdf:li>
              <rdf:li>Serial Port</rdf:li>
              <rdf:li>Audio/Video Control Transport</rdf:li>
              <rdf:li>Audio/Video Remote Control</rdf:li>
              <rdf:li>Generic Audio/Video Distribution</rdf:li>
              <rdf:li>Audio/Video Distribution Transport</rdf:li>
              <rdf:li>Advanced Audio Distribution</rdf:li>
           </rdf:Bag>
        </prf:BluetoothProfile>
```

Figura 6.4: Exemplo de um documento XML referente ao perfil UAProf de um dispositivo.

Na figura 6.4, pode-se notar a descrição de um componente UAProf (*HardwarePlataform*), o qual, contém a descrição *BluetoothProfile* do modelo em questão, que é do tipo de componente `Bag` da especificação RDF. Dentro do componente `Bag` está descrito cada item que o Galaxy S da Samsung suporta em relação a seu serviço de *bluetooth*. Por exemplo, este modelo suporta *File Transfer*, que seria transferir um arquivo via *bluetooth*, entre outros.

A comunicação entre o servidor e o repositório de documentos UAProf é feita através da troca de documentos XML/RDF. Para esta troca ocorrer, é necessário utilizar métodos de requisição HTTP encontrados em classes Java. Para isso, utiliza-se a classe `HttpURLConnection` do pacote `java.net`. Esta classe possui métodos para requisições HTTP. Como deseja-se enviar/receber um arquivo XML/RDF, é preciso utilizar um dos métodos de requisição que a classe `HttpURLConnection` oferece. O método referente é o `setRequestMethod()`, o qual recebe um parâmetro do tipo *String* que se refere ao tipo de requisição que se deseja fazer. Neste

caso, como é um arquivo que está sendo requisitado, é utilizado o parâmetro `POST`. Primeiramente é necessário abrir uma conexão com a URL que possui o arquivo que se deseja receber, através do método `openConnection()`. Logo em seguida é preciso abrir um canal para a entrada dos dados que são recebidos da URL, através do método `getInputStream()`. Após isso, os dados recebidos são guardados e manuseados conforme necessário.

Existem outras formas de requisitar um documento de uma URL, como por exemplo através de *servlets*. Uma das formas mais comuns é utilizando a classe `HttpURLConnection`, a qual é utilizada neste trabalho.

A abordagem proposta neste trabalho possui uma cache para armazenar documentos UAProf. Esta cache serve para manter documentos já acessados anteriormente do repositório em um repositório local. Desta forma, quando um documento UAProf é acessado pela primeira vez, ele é também armazenado na cache. Assim, a próxima vez que o servidor quiser um documento UAProf, ele acessará a cache antes de tentar buscar o documento no repositório. Isso torna a busca mais rápida, caso o documento necessário esteja na cache.

6.4 Módulo Principal

O módulo Principal possui o maior fluxo de dados de toda abordagem. Ele controla dados que chegam dos sensores, do repositório UAProf, da aplicação, e dados que chegam da ontologia perfil do usuário, e processa todos esses dados em conjunto, fazendo uma união de todos para que o resultado final seja a interface do usuário modelada conforme o seu perfil (preferências, necessidades, etc.), perfil de dispositivo e contexto do ambiente.

O módulo Principal recebe todos os dados referentes ao usuário, dispositivo e sensores, e executa regras conforme a figura 6.5.

De acordo com a figura 6.5, o método `executaRegras()` deve executar alguma ação sobre os atributos e valores do XML referente a interface do usuário. Esse XML é enviado pelo aplicativo móvel para ser personalizado segundo as regras pré-estabelecidas no módulo Principal. A regra da figura 6.5 testa se o dispositivo é capaz de aceitar cores e se a tela do dispositivo possui uma resolução de 480x320 pixels, além de testar se a luminosidade é igual a “alto”, bem como se a localização do usuário é igual “emMovimento” e se ele possui uma deficiência visual grave, a qual se encontra dentro dos valores de 0.05 a 0.1. Caso este teste seja verdadeiro, o módulo Principal deve modificar os atributos XML referentes à cores que o usuário possui como preferência na ontologia. Deve também redimensionar os *layouts* para que

Servidor.java

```
1  public class Servidor {
2      public void executaRegras(int idAtributo, UAPROFDispositivo modeloDispositivo,
3                                Sensor sensor, SQWRLPerfilUsuario pu) {
4          //regra1
5          if ((modeloDispositivo.colorCapable == true) &&
6             (modeloDispositivo.screenSize == "480x320") &&
7             (sensor.Luminosidade.info == "alto") &&
8             (sensor.LocalizacaoPessoa.info == "emMovimento") &&
9             ((pu.Deficiencia.DefVisualGrave.nivelDefVisualGrave >= 0.05) &&
10             (pu.Deficiencia.DefVisualGrave.nivelDefVisualGrave <= 0.1))){
11
12             //executa alguma ação sobre os atributos/valores do XML
13             //referente à interface do usuário
14
15         }
16         //regra2
17         ...
18         ...
19     }
20 }
```

Figura 6.5: Exemplo de regra que o módulo Principal executa.

fiquem corretos na resolução de 480x320, além de ajustar o brilho da tela do dispositivo para que o usuário consiga ver a interface de forma legível. Também deve aumentar a fonte e ajustar os elementos da interface, pois o usuário encontra-se em movimento, desta forma, o ajuste se torna necessário para alcançar uma legibilidade maior. As cores de texto e fundo não devem possuir tons parecidos, visto que o usuário possui uma deficiência, a qual se refere à visão, e está no nível grave.

6.5 Módulo de atualização da Ontologia Perfil do Usuário

Esta seção apresenta a maneira como a ontologia perfil do usuário é atualizada conforme mudanças no perfil do usuário. A forma como estas mudanças são capturadas é baseada em testes realizados periodicamente. Estes testes servem para testar habilidades e capturar as preferências do usuário. Os testes são sempre realizados antes do usuário utilizar o aplicativo que será adaptado ao seu perfil. Os tipos de testes realizados são em relação a deficiências, perguntas sobre experiência com dispositivos computacionais, questões sobre preferências de cores de *background* e texto, fontes, idioma e se o usuário prefere somente texto ou somente imagem.

Com a atualização da ontologia Perfil do Usuário ocorrendo periodicamente, a adaptação da interface para determinado usuário é alcançada de forma satisfatória, visto que os dados referentes ao seu perfil estão descritos corretamente na ontologia.

Para exemplificar uma forma de descobrir se o usuário possui deficiência visual (daltonismo), um teste foi desenvolvido para Android. O teste consiste em o usuário comparar duas cores até encontrar o mesmo tom. O usuário deve deslizar a barra (*seekBar*) até encontrar a cor do lado esquerdo igual a do lado direito. Caso encontre, deve clicar no botão “Igual”, caso

contrário, deve clicar em “Diferente”.

Este teste é chamado de Anomaloscopia RGB (*Red-Green-Blue*), e foi desenvolvido no século XX. Desde então é o teste mais preciso para daltonismo vermelho-verde. O teste desenvolvido pode ser visto na figura 6.6.

Figura 6.6: Tela do aplicativo desenvolvido para testar daltonismo.

Ao fim do teste, o módulo AtualizaOntologia atualiza a ontologia perfil do usuário referente ao usuário que executou o teste com o uso da OWL API, modificando seus dados conforme resultados obtidos nos testes. As preferências do usuário também são capturadas periodicamente. Assim, a ontologia perfil do usuário fica atualizada e desta forma a interface final do usuário se adapta cada vez melhor ao seu perfil e suas necessidades.

6.6 Processo de Atualização da Ontologia

Antes de executar qualquer tarefa, a arquitetura PIDIM realiza testes periódicos com o intuito de atualizar a ontologia Perfil do Usuário, caso o perfil do usuário tenha sofrido alguma alteração. A figura 6.7 ilustra como é o fluxo dos dados oriundos dos testes até a atualização da ontologia.

Seguindo a ordem alfabética da figura 6.7, o fluxo (a) tem o objetivo de buscar o documento XML gerado pelos sensores referente ao usuário que entrou no ambiente pervasivo, a fim de a partir disso atualizar a ontologia Perfil do Usuário referente ao usuário que executou os testes. Em seguida, no passo (b), o aplicativo (App) executa os testes necessários para atualizar a ontologia Perfil do Usuário e cria um arquivo XML com os resultados dos testes. Estes tes-

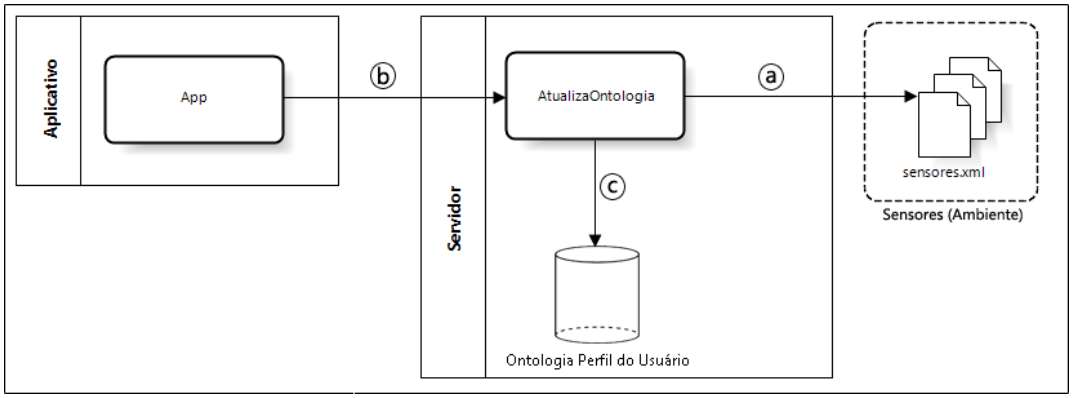

Figura 6.7: Parte da Arquitetura PIDIM que executa a atualização da ontologia Perfil do Usuário

tes são referentes à deficiências, preferências, perguntas sobre dados demográficos, cultura e experiência com computadores.

Após os testes serem realizados, o aplicativo móvel envia os resultados para o módulo AtualizaOntologia, o qual, demonstrado em (c), tem o objetivo de atualizar a ontologia Perfil do Usuário conforme os resultados obtidos nos testes executados no dispositivo móvel. Estas atualizações são feitas com o auxílio da OWL API, utilizando a linguagem SWRL.

6.7 Aplicativo móvel

Nesta parte da abordagem proposta entram os desenvolvedores de aplicativos para dispositivos móveis. Os desenvolvedores implementam a lógica de sua aplicação e criam uma interface padrão para seu aplicativo. A partir deste ponto, onde a aplicação está pronta (tanto lógica quanto interface), entra o papel do módulo Principal, o qual foi descrito na seção 6.4.

O aplicativo deve possuir sua lógica desenvolvida em qualquer linguagem (Java, C#, etc.) que possua a lógica separada do projeto de interface, como por exemplo o Android, que possui a lógica descrita na linguagem Java e a interface descrita em XML, ou C#, que descreve a lógica da aplicação em C# e a linguagem XAML descreve a interface do usuário.

Além da lógica da aplicação e *design* da interface, o aplicativo é responsável por enviar os arquivos referentes à interface do usuário para o módulo Principal do servidor, para que ele a personalize. Assim que a tarefa do módulo Principal terminar, o aplicativo deve receber novamente os arquivos referentes a interface do usuário e então atualizar os documentos XML referentes a interface (representado como UIDL.xml na abordagem). Outro papel importante que o aplicativo deve desempenhar é o pedido de renderização ao Sistema Operacional.

Para que isso se concretize, é necessário que o desenvolvedor do aplicativo móvel crie alguns

métodos na parte lógica da aplicação relativos ao envio de documentos XML via http para o servidor. O uso da classe `HttpURLConnection` do pacote `java.net` é recomendado.

Um exemplo de documento que descreve um botão em XML a ser modificado, e que deve ser enviado ao servidor é mostrado na figura 6.8.

```
<Button
    android:id="@+id/backward"
    android:layout_width="50dp"
    android:layout_height="30dp"
    foo:customFont="verdana.ttf"
    android:text="@string/back"
    android:textColor="@color/text_bg"
    android:textSize="22sp"
    android:textStyle="bold"
    android:background="@color/preto" />
```

Figura 6.8: Exemplo de arquivo XML descrevendo um elemento <button>.

A partir deste exemplo mostrado na figura anterior, os atributos que podem ser modificados devem ser enviados para o módulo Principal. Neste caso, todos os atributos, com exceção do `android:text` podem ser passados para o módulo Principal modificar. Quando o aplicativo envia o identificador (*id*) do elemento a ser personalizado, o módulo Principal sabe qual o elemento a modificar e personaliza os atributos necessários. É recomendado que os desenvolvedores do aplicativo criem uma API para enviar os atributos ou arquivos XML referentes a interface de usuário.

6.8 Considerações do Capítulo

Este capítulo apresentou de forma detalhada como a arquitetura PIDIM funciona, mostrando como cada módulo se comunica e descrevendo cada módulo da arquitetura.

A arquitetura PIDIM, juntamente com a ontologia descrita no Capítulo 5 possibilitam o objetivo deste trabalho, o qual é personalizar a interface do usuário em dispositivos móveis conforme necessidades e preferências do usuário. Com as aplicações desenvolvidas utilizando a metodologia proposta neste trabalho, acredita-se que este objetivo de personalização de interfaces será alcançado, fazendo com que o usuário utilize as aplicações com maior facilidade e com menos erros, resultando em uma interação homem-computador sem estresse e maiores dificuldades.

A fim de validar a arquitetura proposta neste trabalho, a seguir será apresentado um estudo de caso para personalização de interfaces de usuários quando o usuário se encontra em movimento utilizando um dispositivo móvel.

7 CASO DE ESTUDO: INTERFACES PARA USUÁRIOS EM MOVIMENTO (*WALKING USER INTERFACE*)

Para validar a abordagem proposta, foi utilizado um caso de estudo onde o usuário utiliza um *player* de música em duas situações: (1) quando está parado; e (2) quando está em movimento. O processo de validação é baseado na figura da arquitetura 6.1, onde existe uma numeração que orienta o controle do fluxo dos dados dentro da arquitetura.

O cenário deste caso de estudo é baseado no trabalho de KANE; WOBBROCK; SMITH (2008). Um usuário utilizando um dispositivo móvel encontra-se em um ambiente pervasivo povoado de sensores de luminosidade, ruído, temperatura, localização dos usuários, localização do dispositivo, etc. No momento que o usuário entra em movimento, muitos fatores influenciam na perda de desempenho, perda de atenção e habilidade motora do mesmo. Para diminuir essa perda de desempenho e deficiências motoras situacionais, a arquitetura proposta neste trabalho visa adaptar a interface do aplicativo utilizado pelo usuário.

Testes com diferentes perfis de usuários foram realizados. Primeiramente, um aplicativo para tocar músicas (*music player*) foi desenvolvido para Android na versão 2.2, com o intuito de realizar os testes com usuários reais em um dispositivo real. O aplicativo desenvolvido possui uma interface padrão, a qual deve ser adaptada conforme o perfil do usuário que está utilizando o aplicativo no momento, o contexto do ambiente, e as capacidades do dispositivo.

Além do aplicativo desenvolvido, um percurso foi estabelecido (vide figura 7.1), com o intuito de receber um *feedback* dos usuários do teste logo após o término do percurso. O percurso possuía pessoas, bancos e árvores como obstáculos.

Figura 7.1: Percurso percorrido pelos usuários. (GOOGLE, 2011)

Cinco usuários foram testados, sendo três do sexo masculino e dois do sexo feminino. Dentre eles haviam usuários com daltonismo, membro amputado, idosos e jovens. A tabela 7.1 mostra o perfil de cada usuário que participou dos testes.

Tabela 7.1: Tabela de usuários que realizaram os testes

Tabela de usuários que realizaram os testes						
Usuário	**Idade**	**Sexo**	**Cor do Texto**	**Cor *Background***	**Fonte**	**Experiência**
Usuario_1	21	F	Verde	Branco	Arial.ttf	Ocasional
Usuario_2	58	F	Preto	Branco	Calibri.ttf	Ocasional
Usuario_3	14	M	Preto	Vermelho	Impact.ttf	Ocasional
Usuario_4	63	M	Azul Fraco	Preto	Arial.ttf	Novato
Usuario_5	25	M	Branco	Vermelho	Verdana.ttf	Especialista

Além dos dados descritos na tabela 7.1, todos os usuários possuem nacionalidade brasileira e entendem o idioma português. O Usuario_2 possui daltonismo e o Usuario_5 possui a mão esquerda amputada. Outro dado importante é a orientação de cada usuário, visto que em um teste no qual o usuário está em movimento, certas habilidades motoras são perdidas. O Usuario_1, Usuario_2 e Usuario_4 são destros, enquanto os usuários Usuario_3, e Usuario_5 são canhotos.

Posteriormente ao desenvolvimento do aplicativo, foi simulado quando o usuário está em movimento ou parado, além de dados sobre as capacidades dos dispositivos, como se suporta cores, sistema operacional, e versão do sistema operacional. O dispositivo utilizado para os testes foi um Samsung Galaxy 5 (modelo I5500), o qual é possível apresentar cores, seu sistema operacional é Android e a versão do Android no dispositivo é a versão 2.3. A figura 7.2 mostra a interface padrão do aplicativo desenvolvido para Android na versão 2.2.

Figura 7.2: *Layout* padrão do aplicativo *music player*

Baseado na numeração da figura da arquitetura 6.1, será mostrado aqui como o fluxo de comunicação entre os módulos funciona para o cenário deste caso de estudo. Os sensores do ambiente pervasivo captam mudanças de contexto que possam ser importantes para a personalização da interface do usuário. A cada mudança captada pelos sensores, é criado um documento XML descrevendo a mesma. No início do fluxo, em (a), o módulo AtualizaOntologia busca o documento XML referente ao usuário, nos sensores. Logo em seguida, em (b), o Aplicativo (App) repassa o resultado de alguns testes referentes à preferências e deficiências do usuário, e por fim, em (c), a ontologia perfil do usuário é atualizada. Após a atualização da ontologia, em (1), são buscados os documentos XML criados pelos sensores, pelo módulo Sensor. Neste ponto o módulo Sensor possui dados do contexto, localização de usuários ou dispositivos, luminosidade, modelo do dispositivo, etc.

```
▼<sensor xmlns:xsi="sensor.xsd">
   <id>SR0011</id>
   <!-- código identificador do sensor -->
   <info>Samsung I5500</info>
   <!-- atributo valor -->
   <tipo>Dispositivo</tipo>
   <!-- Tipo de informação que o sensor detecta -->
 </sensor>
```

Figura 7.3: Dados capturados pelo sensor de dispositivos.

Após a captura dos dados relevantes ao sistema, em (2), o módulo Sensor passa o modelo do dispositivo Samsung I5500 para o módulo UAPROFDispositivo. Os dados do Usuario_5 também são passados para o módulo SQWRLPerfilUsuario, no passo (3). No passo (4), SQWRLPerfilUsuario busca na ontologia perfil do usuário através da OWL API os dados do Usuario_5. A figura 7.4 mostra como é feita a busca dos dados do Usuario_5 (pUser) na ontologia de perfil do usuário. Como exemplo, o código apresentado busca os dados demográficos do Usuario_5.

A fim de capturar as capacidades do dispositivo Samsung I5500, em (5), o módulo UAPROFDispositivo tenta buscar o modelo do dispositivo na cache UAProf. Caso o modelo não exista na cache, deve buscar no repositório UAProf. Isso acontece no passo (6), onde o módulo UAPROFDispositivo utiliza a classe java HttpURLConnection do pacote java.net para buscar o documento XML ou RDF referente ao modelo Samsung I5500. Depois disso, em (7), o módulo SQWRLPerfilUsuario tem o papel de passar os dados do Usuario_5 para o módulo Principal. Os dados referentes ao Usuario_5 estão descritos na tabela 7.2. Note que a instância Mao_2 *possuiDeficiencia* DefMotoraAmputacao_1. Desta forma sabe-se que o Usuario_5 possui a

SQWRLPerfilUsuario.java

```
public SQWRLResultValue[] getUserInfo(String pUser) {
    SQWRLResultValue vetUserInfo[] = new SQWRLResultValue[3];
    try{

        OWLModel owlModel = ProtegeOWL.createJenaOWLModelFromURI(path);

        SWRLFactory swrlfactory = new SWRLFactory(owlModel);

        swrlfactory.createImp("consultaUserInfoIdade", "UserProfile:idade("+pUser+", ?i) -> sqwrl:select(?i)");
        swrlfactory.createImp("consultaUserInfoNacionalidade", "UserProfile:nacionalidade("+pUser+", ?n) -> sqwrl:select(?n)");
        swrlfactory.createImp("consultaUserInfoSexo", "UserProfile:sexo("+pUser+", ?s) -> sqwrl:select(?s)");

        SQWRLQueryEngine queryEngine = SQWRLQueryEngineFactory.create(owlModel);
        queryEngine.runSQWRLQueries();

        SQWRLResult resultadoUserInfoIdade = queryEngine.getSQWRLResult("consultaUserInfoIdade");
        vetUserInfo[0] = resultadoUserInfoIdade.getValue(0);
        SQWRLResult resultadoUserInfoNacionalidade = queryEngine.getSQWRLResult("consultaUserInfoNacionalidade");
        vetUserInfo[1] = resultadoUserInfoNacionalidade.getValue(0);
        SQWRLResult resultadoUserInfoSexo = queryEngine.getSQWRLResult("consultaUserInfoSexo");
        vetUserInfo[2] = resultadoUserInfoSexo.getValue(0);

        return vetUserInfo;
    }
    catch (Exception e) {return null;}
}
```

Figura 7.4: Código Java da busca dos dados demográficos do Usuario_5 na ontologia perfil do usuário.

Mao_2 amputada. A instância Mao_2 se refere à mão esquerda.

Tabela 7.2: Tabela com dados do Usuario_5 buscados na ontologia

Tabela com dados do Usuario_5 buscados na ontologia	
corPreferidaTexto	branco
corPreferidaBackground	vermelho
fontePreferida	Verdana.ttf
possuiExperiencia	ExpEspecialista_1
idade	25
nacionalidade	brasileiro
sexo	M
possuiDeficiencia	DefMotoraAmputacao_1
MembroCorpoHumano	Cabeca_3, Mao_2, e Perna_3

No passo (8), o módulo UAPROFDispositivo deve passar os dados do dispositivo (Samsung I5500) para o módulo Principal. Os dados do dispositivo Samsung I5500 são:

- <prf:ColorCapable>Yes</prf:ColorCapable>
- <prf:OSName>Android</prf:OSName>
- <prf:OSVersion>2.3</prf:OSVersion>

Logo após, em (9), o módulo Sensor passa o restante dos dados captados pelos sensores para o módulo Principal. Neste momento o módulo Principal deve executar regras com a junção dos dados dos sensores, dispositivo, e perfil do usuário conforme Seção 6.4. Depois do módulo

Principal executar as regras necessárias, no passo (10), o aplicativo móvel envia para o módulo Principal os atributos e identificadores (id), ou arquivos XML referentes a interface do usuário para a personalização da mesma. O envio é realizado através da classe HttpURLConnection.

No caso do *music player*, os arquivos XML referentes são: *list_bg.xml* que possui um componente <selector>, o qual possui componentes <item> para definir as cores de fundo (*background*) dos componentes da interface. Exemplo:<item android:state_selected="false" android:state_pressed="false"android:drawable="@color/preto"/>, onde state_selected="false" significa que o componente na interface não está selecionado, bem como não está pressionado (android:state_pressed="false"). Desta forma, a cor a ser aplicada é baseada em android:drawable="@color/preto".

O *text_bg.xml* possui os mesmos componentes que o arquivo list_bg.xml, porém, este arquivo serve para modificar a cor do texto. O componente <item> deve ser especificado conforme <item android:color="@color/dourado"/>, modificando o atributo android:color e não android:drawable, o qual é para *background*.

O arquivo *list.xml* serve para configurar a lista de músicas do *music player*. Este arquivo possui um <LinearLayout> e dentro do layout existe um componente <TextView>. Os atributos que podem ser modificados nestes dois componentes do arquivo list.xml, não restritos à, são android:layout_width="fill_parent", android:layout_height="50dp", android:textSize="15sp", android:textStyle="normal"e foo:customFont="calibri.ttf", onde "fill_parent", "50dp", e "15sp" podem ser trocados por qualquer dimensão que necessite, "normal"pode ser substituído por "bold", "italic", ou "bold|italic", e "calibri.ttf"pode ser modificado para qualquer fonte que o sistema aceite. E por fim, o arquivo *main.xml*, é o arquivo principal da interface de usuário. Nele existem três *layouts*. O primeiro, <LinearLayout> é composto de outro <LinearLayout> e um <TableRow> *layout*.

O segundo *layout* é composto por um componente <ListView>, o qual possui um <TextView> do arquivo list.xml. Os atributos que podem ser modificados neste <ListView> são android:layout_width, android:layout_height, android:divider, e android:dividerHeight. O atributo android:divider recebe uma cor como valor, pois é ele que divide os componentes <TextView> pertencentes a <ListView>. Por fim, o último *layout*, TableRow, possui quatro componentes, sendo três botões (<button>) e um <TextView>. Nestes quatro componentes, é possível modificar os atributos android:layout_width, android:layout_height, android:text, android:textColor, android:textSize, android:textStyle, e foo:customFont.

A partir disso, em (11), os arquivos referentes a interface do usuário são modificados e devolvidos para o aplicativo móvel por http. Depois da modificação, em (12), o aplicativo móvel recebe os dados do módulo Principal e atualiza os arquivos relativos a interface do usuário (list_bg.xml, text_bg.xml, main.xml, list.xml). Após atualizar os arquivos da interface do usuário, o aplicativo móvel requisita a renderização da interface através do método invalidate() da classe android.view.View do android. Isso ocorre no passo (13). Por fim, em (14), o sistema operacional renderiza a interface do usuário final.

Após o último passo da abordagem proposta concluído, o usuário verá sua interface adaptada. As figuras 7.5 e 7.6 mostram como a interface final ficou adaptada para o Usuario_5.

Figura 7.5: Interface final adaptada para as necessidades e preferências do Usuario_5, onde (a) corresponde à interface quando o usuário está parado, e (b) quando o usuário está em movimento.

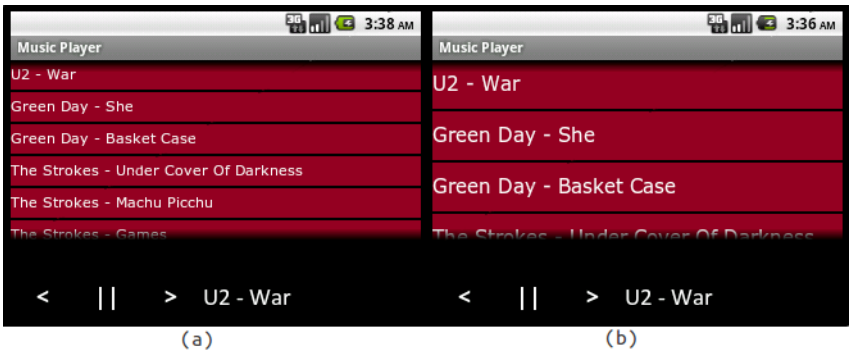

Figura 7.6: Interface final adaptada para as necessidades e preferências do Usuario_5, onde (a) corresponde à interface quando o usuário está parado, e (b) quando o usuário está em movimento. (Orientação: *landscape*)

7.1 Resultados da Avaliação do Caso de Estudo

Todos os usuários testados ficaram satisfeitos com a adaptação. O Usuario_5 se mostrou muito satisfeito com a adaptação para o seu perfil, visto que ele possui a mão esquerda amputada recentemente e ele é canhoto. Desta forma, o usuário teve que testar o aplicativo *music player* com a mão direita, a qual não é tão habilidosa quanto à esquerda. O Usuario_4 também foi um dos que mais elogiou a adaptação na interface. O mesmo não possuía nenhuma deficiência, porém, por ser idoso, algumas deficiências situacionais podem aparecer, ou seja, deficiências causadas pela situação momentânea, a qual era utilizar um dispositivo móvel em movimento, em um percurso com obstáculos.

Testes em ambiente fechado também foram realizados. Foram utilizadas dez pessoas para o teste em ambiente fechado. Elas tiveram que utilizar o protótipo em uma sala, sem obstáculos, ou ruídos para tirar a concentração na hora do teste. Três dos usuários testados executaram os teste nos dois ambientes (público e fechado). As dificuldades relatadas pelos três usuários foram os obstáculos e desvios de atenção com outras pessoas e barulhos ao seu redor no ambiente público. Porém, os usuários também relataram que a adaptação para o perfil de cada um estava bom e supria suas dificuldades. No ambiente fechado, o *feedback* de cada usuário foi somente em relação à adaptação quando caminhavam, pois não haviam obstáculos para desviar a atenção, como no ambiente público. A diferença entre testes em um ambiente fechado e um ambiente público é grande, visto que no último, muitos obstáculos, ruídos, luminosidade, estavam presentes. O grande problema de testes em ambientes abertos (públicos) é que além dos obstáculos, barulhos também existem, estes fatores tiram a atenção do usuário no momento de executar as tarefas do teste. Porém, tanto no ambiente público como fechado, os testes foram realizados com 100% de sucesso. Nenhum dos usuários teve problemas em utilizar a interface adaptada para o seu perfil.

Segundo (SANTOS, 2008), existem diversos autores que procuraram ao longo do tempo, encontrar o conjunto ideal de regras e métricas para medir o grau de usabilidade de sistemas. Muitas regras são semelhantes e por isso, a unificação dos critérios avaliados pelos autores pôde ser concluída. Com a união das métricas, chegou-se a um conjunto de quatro critérios importantes: facilidade no aprendizado, facilidade de relembrar, eficiência, e eficácia e satisfação. Esses quatro critérios foram utilizados para avaliar o aplicativo *music player* para que os usuários passassem o *feedback* de avaliação da IU adaptada conforme seu perfil. O gráfico da figura 7.7 mostra o grau de satisfação dos usuários conforme os critérios avaliados.

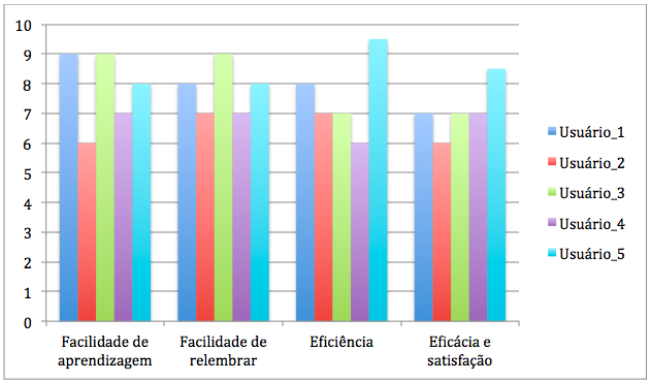

Figura 7.7: Grau de satisfação dos usuários baseado em critérios de usabilidade

7.2 Considerações do Capítulo

A partir da análise do caso de estudo apresentado, é possível afirmar que a metodologia proposta neste trabalho pode ser aplicada para qualquer usuário, dispositivo e ambiente, a fim de personalizar a interface do usuário. É necessário evidenciar que, para que a arquitetura PI-DIM funcione corretamente, a infra-estrutura do ambiente em questão necessita de dispositivos computacionais que o caracterizem como um ambiente pervasivo.

O estudo de caso foi retirado da literatura e adaptado para esta arquitetura a fim de provar que a abordagem proposta neste trabalho funciona em qualquer situação, ou seja, para qualquer usuário, dispositivo e ambiente com infra-estrutura que caracterize um ambiente pervasivo.

8 TRABALHOS RELACIONADOS

O escopo deste capítulo é apresentar alguns trabalhos relacionados a esta dissertação de mestrado com foco nos assuntos de personalização de interfaces de usuário para dispositivos móveis utilizando conceitos de perfil de usuário juntamente com perfil do dispositivo em ambientes pervasivos.

Na literatura, muitos trabalhos se referem a personalização de interfaces e sistemas computacionais para pessoas com deficiências ou pessoas idosas somente, deixando um pouco de lado as preferências e necessidades de qualquer usuário. Esta dissertação trata de qualquer tipo de usuário.

O trabalho de ABASCAL et al. (2011) aborda o sistema EGOKI, desenvolvido para gerar interfaces de usuário adaptadas para dispositivos móveis para fornecer acesso à serviços em ambientes pervasivos para pessoas com deficiências. Em ambientes de computação pervasiva, um *middleware* fornece uma descrição abstrata da interface do usuário depois da fase de descoberta de serviços. Através desta descrição, o EGOKI cria uma instância de interface de usuário adaptada às características do usuário e as características técnicas do seu dispositivo móvel. Esta adaptação é baseada no usuário e modelos de dispositivo, gerenciado por meio de ontologias. A interface do usuário é criada através do EGOKI utilizando a linguagem de descrição de interfaces UIML.

O trabalho de Abascal difere da proposta apresentada nesta dissertação em poucos aspectos. Abascal utiliza ontologias para descrever usuários, dispositivos, e o contexto do ambiente. Porém, os autores não fazem uso de inferências sobre a ontologia, as quais permitem resolver problemas baseados em conhecimentos antecipados, além de aumentar o poder de expressividade da ontologia.

Outra questão que difere da abordagem desta dissertação com a de Abascal é que os autores criaram uma ontologia para descrever as capacidades do dispositivo do usuário, enquanto a abordagem desta dissertação utiliza um vocabulário UAProf recomendado pela W3C, padronizado e completo quanto às características dos dispositivos. Desta forma, utilizando um vocabulário recomendado pela W3C e utilizando inferências sobre os dados descritos na ontologia perfil do usuário, a personalização da interface final do usuário atingirá um nível de satisfação maior ao usuário, visto que através das inferências realizadas no presente trabalho é possível deduzir certas questões importantes para a personalização da interface final e assim adaptar a

interface de acordo com as capacidades do dispositivo móvel e necessidades e preferências do usuário.

Já o trabalho de PARTARAKIS et al. (2009) detalha uma arquitetura que tem o intuito de oferecer novos potenciais para suporte à adaptação de conteúdo de interfaces de usuário para a *Web*. O trabalho de Partarakis é baseado em um *framework* de criação de interfaces de usuário para a *Web* chamado EAGER. O EAGER permite um desenvolvedor produzir portais *Web* que possuem capacidade de adaptar desde formas de interação com o usuário até elementos de interface para cada usuário, de acordo com informações do perfil do mesmo.

A arquitetura do trabalho desenvolvido por Partarakis inicia com uma interface de usuário abstrata, modelada com EAGER, a qual é adaptada conforme um repositório de projetos que consiste de elementos de interface de usuário. Este repositório se comunica com um módulo chamado *Reasoner and Rule Engine* (Raciocinador e motor de regras), o qual tem o papel de escolher os elementos para construir a interface do usuário concreta baseado na decisão do motor de regras.

Além destes módulos, a arquitetura do trabalho de Partarakis possui uma base de conhecimento, a qual é modelada através de ontologias, para armazenar o perfil do usuário, capacidades dos dispositivos, e interações do usuário. Quando um perfil de usuário é criado, o raciocinador e o motor de regras são invocados, gerando resultados que são armazenados na base de conhecimentos e utilizadas por um núcleo de adaptação (*Adaptation Core*), o qual é usado para inferir determinadas ações para concluir a interface final.

Uma das diferenças do trabalho de Partarakis para este é que o autor desenvolveu sua arquitetura para interfaces *Web*, possuindo certas vantagens sobre a forma de abordagem desta dissertação, como certas desvantagens. Estas vantagens e desvantagens são em relação à aplicações nativas, híbridas, e *Web*. O trabalho de Partarakis não tem como base a computação pervasiva, mas sim, qualquer dispositivo que suporte interfaces *Web*, incluindo *smartphones*, PDAs, e outros dispositivos móveis. Sem a computação pervasiva, aplicativos, sejam eles *Web*, híbridos, ou nativos, perdem certa flexibilidade de informação em relação ao contexto atual. Para solucionar este problema, a união da computação móvel com a pervasiva deve ser alcançada (ARAUJO, 2003).

Existem alguns trabalhos relacionados à partes da abordagem desta tese de mestrado. Estes trabalhos serviram como base para o fundamento deste trabalho. Um deles é o trabalho de GOLEMATI et al. (2007), o qual apresenta métodos e aplicações relacionadas à perfis de usuários

modelados por ontologias. Outros trabalhos relacionados a computação pervasiva, descrição de capacidades de dispositivos, e geração automática de interfaces de usuário foram alicerce para a criação da abordagem proposta neste trabalho.

9 CONCLUSÃO

Com os diferentes tipos de plataformas e dispositivos eletrônicos, é importante que se trabalhe para obter-se uma hegemonia nas interfaces, de modo que a interface se adapte ao usuário e não o contrário. Desta forma, diminui-se a curva de aprendizado entre diferentes dispositivos e por consequência reduz-se as possibilidades de erros e estresse.

A realização desta pesquisa foi motivada principalmente devido a falta de aplicações computacionais que tratam a interface do usuário conforme as necessidades e preferências do usuário. Mesmo com a popularização de dispositivos computacionais móveis, ainda não existem aplicações pervasivas capazes de interagir automaticamente com o usuário, levando em consideração o contexto em que se encontra.

Este trabalho apresentou detalhadamente uma abordagem que propõe uma arquitetura que possa ser utilizada como base para o desenvolvimento de sistemas pervasivos a fim de personalizar as interfaces de usuários de acordo com seu perfil através da utilização de conceitos de computação pervasiva e ontologias aplicadas ao domínio de perfil do usuário. Para o projeto, foi desenvolvida uma ontologia a qual representa o conhecimento existente sobre perfis de usuários.

Com o objetivo de descrever o fluxo de funcionamento da arquitetura e também para validação da mesma, foi apresentado um caso de estudo encontrado na literatura e adaptado para a arquitetura deste trabalho.

Utilizando esta arquitetura como base, acredita-se que sistemas pervasivos com interfaces de usuários adaptadas para as necessidades e preferências dos usuários tornam-se mais fáceis de manusear e dão ao usuário um prazer maior ao realizar suas tarefas, bem como diminuem o estresse do usuário ao utilizar um dispositivo móvel.

O principal objetivo deste trabalho foi adaptar interfaces de usuário para dispositivos móveis baseadas no perfil de usuário, com auxílio de ontologias, perfil de dispositivo, com uso de um vocabulário UAProf, e contexto do ambiente pervasivo, captado por sensores.

A partir dos testes realizados pelo caso de estudo, foi alcançado um *feedback* dos usuários quanto a adaptação da IU segundo os critérios do gráfico da figura 7.7. Quanto ao critério de facilidade de aprendizado, onde o sistema deve apresentar facilidade de uso, os resultados foram bons, obtendo uma média de 7,8 pontos dentre os 5 usuários. Quando medido o critério de facilidade de relembrar tarefas e ações sobre o sistema, os resultados foram os mesmos, com

média entre todos os usuários de 7,8 pontos.

No ponto de avaliação sobre eficiência, onde o sistema deve desempenhar seu papel de forma correta e habilitar o usuário a atingir um alto grau de produtividade e desempenho em suas tarefas, a média obtida foi de 7,5 pontos, o que mostra que o sistema desempenha bem suas tarefas.

No momento da avaliação de eficácia e satisfação, onde por eficácia entende-se o sistema realizar a tarefa da melhor forma possível e por satisfação entende-se a percepção do usuário com o sistema e a forma com que a IU é adaptada, a média dos testes dos usuários foi de 7,1 pontos, o que demonstra que a adaptação realizada atingiu de forma satisfatória o objetivo. A média geral das avaliações, que foi de 7,5 pontos, nos remete que o resultado da adaptação da IU baseada na arquitetura PIDIM atingiu seu objetivo satisfatoriamente, com média superior a 7 de 10 pontos no total.

A fim de embasar alguns trabalhos futuros sobre adaptação de interfaces de usuários para dispositivos móveis, algumas alterações na abordagem deste trabalho poderão ocorrer em trabalhos subsequentes. Estas alterações estão citadas a seguir:

- Expandir a abordagem tratando de interfaces de usuário dinâmicas;
- Especificar maiores detalhes do domínio da ontologia Perfil do Usuário, tentando absorver mais dados do usuário quanto puder, em questões de cultura, deficiências cognitivas, tamanho das mãos do usuário, objetos utilizados pelo usuário, etc, melhorando assim o método de entrada de dados, como por exemplo, como captar a interação em uma tela sensível ao toque se o usuário está utilizando luvas?
- Sobre dados dinâmicos de contexto do ambiente, uma possível alteração da abordagem proposta seria utilizar ontologias para inferir resultados mais precisos;
- A linguagem SWRL utilizada para inferências nessa abordagem permite a criação de novos operadores. Assim, a construção de uma biblioteca que colabore na construção de regras e consultas para a computação sensível ao contexto, aplicada à adaptação de interfaces de usuários para ambientes pervasivos pode ser elaborada;
- Um estudo para tentar adaptar a abordagem proposta neste trabalho à computação nas nuvens (*cloud computing*) pode ser realizado, visando utilizar serviços da *cloud computing*, a qual facilitaria a comunicação entre cliente e servidor;

- Verificar a possibilidade de criar um repositório UAProf confiável. Desta forma, a busca à vários repositórios distintos (dos fabricantes) não será necessária. Isso tornaria a busca de documentos referentes aos perfis de dispositivos mais rápida e mais confiável;
- Criar uma API para facilitar o trabalho do desenvolvedor do aplicativo móvel a enviar os atributos e arquivos XML que deseja personalizar.

REFERÊNCIAS

ABASCAL, J.; AIZPURUA, A.; CEARRETA, I.; GAMECHO, B.; GARAY, N.; MIÑÓN, R. A modular approach to user interface adaptation for people with disabilities in ubiquitous environments. **Internal Technical Report No. EHUKATIK0111**, [S.l.], 2011.

AHAMED, S. I.; ZULKERNINE, M.; HAQUE, M. M. Security, Privacy, and Trust for Pervasive Computing Applications. **Mobile multimedia communications: concepts, applications, and challenges**, [S.l.], p.327–342, 2008.

ALLIANCE, O. M. **Especificação UAProf**. 2001. Disponível em, `http://www.openmobilealliance.org/tech/affiliates/wap/wap-248-uaprof-20011020-a.pdf`.

ANDROID. **Android Developers Versions**. 2011a. Disponível em, `http://developer.android.com/resources/dashboard/platform-versions.html`.

ANDROID. **Android Developers SDK**. 2011b. Disponível em, `http://developer.android.com/sdk/index.html`.

ANDROID. **Android Developers Interface**. 2011c. Disponível em, `http://developer.android.com/guide/topics/ui/index.html`.

ARAUJO, R. B. D. Computação Ubíqua: princípios, tecnologias e desafios. **21 Simpósio Brasileiro de Redes de Computadores**, [S.l.], p.45–115, 2003.

BARBOSA, D. N. F. Um modelo de educação Ubíqua orientado à consciência do contexto do aprendiz. **Tese apresentada como requisito para obtenção do grau de doutor em Ciência da Computação (UFRGS). Porto Alegre, RS.**, [S.l.], 2007.

BORODITSKY, L. How does your language shape the way we think? **In Whats Next? Dispatches on the future of Science**, [S.l.], 2009.

BOTWINICK, J. Aging and behavior: a comprehensive integration of research findings. In: . [S.l.]: Springer, 1973.

BRAY, T.; PAOLI, J.; SPERBERG-MCQUEEN, C. M.; MALER, E.; YERGEAU, F. **Extensible Markup Language (XML) 1.0 (Fifth Edition)**. [S.l.]: World Wide Web Consortium, 2008.

BROOS, A. Gender and information and communication technologies (ICT) anxiety: male self-assurance and female hesitation. **Cyber Psychology and Behaviour**, [S.l.], p.21–31, 2005.

BURGMANN, I.; KITCHEN, P.; WILLIAMS, R. Does Culture Matter on the Web? **Marketing Intelligence & Planning**, [S.l.], p.62–73, 2006.

CHEN, H. An Intelligent Broker for Context-Aware Systems. In: IN ADJUNCT PROCEEDINGS OF UBICOMP, 2003. **Anais...** [S.l.: s.n.], 2003. p.183–184.

CHEN, H.; FININ, T.; JOSHI, A. An ontology for context-aware pervasive computing environments. **The Knowledge Engineering Review**, [S.l.], p.197–207, 2003.

CHOI, B. A qualitative cross-national study of cultural influences on mobile data service design. In: IN PROCEEDINGS OF THE SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, 2005. **Anais...** ACM Press, 2005. p.661–670.

DEAN, M.; SCHREIJBER, G. **OWL Web Ontology Language Reference**. [S.l.: s.n.], 2004. Recommendation.

DEY, A. K. Understanding and Using Context. **Personal Ubiquitous Comput.**, London, UK, v.5, p.4–7, January 2001.

FENSEL, D. **Relating Ontology Languages and Web Standards**. 2000.

FORTE, M. **Especificação de perfis e regras, baseada em ontologias, para adaptação de conteúdo na Internet.** 2006. Dissertação de mestrado — Universidade Federal de São Carlos, São Carlos, SP, Brasil.

FREITAS, L. O. **Uma metodologia para assistir pacientes em ambientes homecare pervasivos.** 2011. Dissertação de Mestrado — Universidade Federal de Santa Maria, UFSM, Santa Maria, RS, Brasil.

GALDO, E. M. del; NIELSEN, J. International users interface. In: ACM PRESS, 1996, New York, NY, USA. **Anais...** John Wiley and Sons: Inc., 1996.

GARGENTA, M. Learning Android - Building Applications for the Android Market. In: . [S.l.]: O'Reilly, 2011.

GASSEN, J. B.; COPPETI, M.; FREITAS, L. O.; MARTINS, E.; LIBRELOTTO, G. R. Construção de uma ontologia para um hospital. **VII Simpósio de Informática da Região Centro do RS**, [S.l.], 2008.

GAVA, T. B. S.; MENEZES, C. S. D. Uma ontologia de domínio para a aprendizagem cooperativa. **XIV Simpósio Brasileiro de Informática na Edução NCE IMUFRJ**, [S.l.], p.336–345, 2003.

GOLEMATI, M.; KATIFORI, A.; VASSILAKIS, C.; LEPOURAS, G.; HALATSIS, C. Creating an Ontology for the User Profile: method and applications. In: IN PROCEEDINGS OF THE FIRST INTERNATIONAL CONFERENCE ON RESEARCH CHALLENGES IN INFORMATION SCIENCE (RCIS), 2007. **Anais...** [S.l.: s.n.], 2007.

GOOGLE. **Google Maps**. 2011. Disponível em, `maps.google.com`.

GRUBER, T. R. Toward principles for the design of ontologies used for knowledge sharing. In: International Journal of Human-Computer Studies, 1993. **Anais...** Kluwer Academic Publishers, 1993. p.907–928.

GUARINO, N. Understanding, Building and Using Ontologies: a commentary to using explicit ontologies in kbs development. **International Journal of Human and Computer Studies**, [S.l.], p.293–310, 1997.

GUIMARÃES, F. J. Z. **Utilização de ontologias no domínio B2C.** 2002. Dissertação de Mestrado — Pontifícia Universidade Católica do Rio de Janeiro.

HORROCKS, I.; PATEL-SCHNEIDER, P. F.; BOLEY, H.; TABET, S.; GROSOF, B.; DEAN, M. **SWRL**: a semantic web rule language combining owl and ruleml. 2004.

IBM. **Easy To Use**: user expectations. 2001. Disponível em:, http://www.ibm.com/easy/.

JESS. **Jess, the Rule Engine for the Java Platform**. 2011.

JUNIOR, O. G. F. **Um Modelo de Sistema de Gestão do Conhecimento para Grupos de Pesquisa e Desenvolvimento**. 2003. Tese de Doutorado — Universidade Federal de Santa Catarina, Florianópolis, SC.

KANE, S. K.; WOBBROCK, J. O.; SMITH, I. E. Getting off the treadmill: evaluating walking user interfaces for mobile devices in public spaces. In: HUMAN COMPUTER INTERACTION WITH MOBILE DEVICES AND SERVICES, 10., 2008, New York, NY, USA. **Proceedings...** ACM, 2008. p.109–118.

KARAHANNA, E.; EVARISTO, J. R.; SRITE, M. Levels of Culture and Individual Behavior: an integrative perspective. **Journal of Global Information Management**, [S.l.], p.1–20, 2005.

KISS, C. **Composite Capability/Preference Profiles**: structure and vocabularies. 2007. Disponível em, `http://www.w3.org/TR/2007/WD-CCPP-struct-vocab2-20070430/`.

KNUBLAUCH, H.; FERGERSON, R. W.; NOY, N. F.; MUSEN, M. A. **The Protégé OWL plugin**: an open development environment for semantic web applications. [S.l.]: Springer, 2004. 229–243p.

KURNIAWAN, S. Older people and mobile phones: a multi-method investigation. **International Journal of Human-Computer Studies**, [S.l.], v.66, n.12, p.889–901, 2008.

LEE, W. Beginning Android Application Development. In: BEGINNING ANDROID APPLICATION DEVELOPMENT, 2011. **Anais...** John Wiley and Sons, 2011. (Wrox beginning guides).

LIBRELOTTO, G. R.; AUGUSTIN, I.; GASSEN, J.; KURTZ, G.; FREITAS, L.; MARTINI, R.; AZEVEDO, R. P. de. OntoHealth: an ontology applied to pervasive hospital environments. **IJAPUC**, [S.l.], p.33–45, 2010.

LIBRELOTTO, G. R.; FREITAS, L. O.; FIORIN, A.; MOZZAQUATRO, B. A.; PASETTO, L.; MARTINI, R. G.; AZEVEDO, R. P. de; PEREIRA, R. T. OntoHealth: a system to process ontologies applied to health pervasive environment. In: FOURTH INTERNATIONAL CONFERENCE ON UBI-MEDIA COMPUTING, 2011., 2011, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2011. p.59–64. (U-MEDIA '11).

LIBRELOTTO, G. R.; FREITAS, L. O.; GASSE, J. B.; COPETTI, M.; TURCHETTI, R. C.; SILVA, F. L. da; AUGUSTIN, I. Uma ferramenta para o processamento da representação do domínio de atividades médicas. **Hífen**, [S.l.], p.25–32, 2008a.

LIBRELOTTO, G. R.; GASSEN, J. B.; FREITAS, L. O.; SILVEIRA, M. C.; SILVA, F. L.; AUGUSTIN, I.; HENRIQUES, P. Uma ontologia aplicada a um ambiente pervasivo hospitalar. **In: 8 Conferência da Associação Portuguesa de Sistemas de Informação**, Setúbal, Portugal, p.34–34, 2008b.

LIMA, J. C.; CARVALHO, C. L. **Ontologias - OWL (Web Ontology Language)**. 2005.

LIMBOURG, Q.; VANDERDONCKT, J.; MICHOTTE, B.; BOUILLON, L.; FLORINS, M.; TREVISAN, D. UsiXML: a user interface description language for context-sensitive user interfaces. In: IN PROCEEDINGS OF THE ACM AVI'2004 WORKSHOP DEVELOPING USER INTERFACES WITH XML: ADVANCES ON USER INTERFACE DESCRIPTION LANGUAGES, 2004. **Anais...** Press, 2004. p.55–62.

LINDSEY, D. T.; BROWN, A. M. Sunlight and "blue": the prevalence of poor lexical color discrimination within the "grue"region. **Psychological Science**, [S.l.], p.291–294, 2004.

LIU, Y.; OSVALDER, A.-L.; KARLSSON, M. Considering the importance of user profiles in interface design. In: **User Interfaces**. [S.l.: s.n.], 2010. p.23.

MCGUINNESS, D. L.; HARMELEN, F. V. **OWL Web Ontology Language Overview**. 2004.

MICROSOFT. **Home of Microsoft .NET**. 2009. Disponível em, `http://www.microsoft.com/net`.

MOBILE, M. **Android Design Guidelines**. 2011. Disponível em, `http://www.mutualmobile.com/wp-content/uploads/2011/03/MM_Android_Design_Guidelines.pdf`.

MOLINA, P. **User interface specification**: from requirements to automatic generation. 2003. Tese de Doutorado — Universidad Politécnica de Valencia, Espanha.

MOTIK, B. **OWL 2 Web Ontology Language Profiles**. 2009. Disponível em, `http://www.w3.org/TR/owl2-profiles/`.

MOZILLA. **Home of the Mozilla XML User Interface Language**. 2010. Disponível em, `https://developer.mozilla.org/En/XUL`.

NEWELL, A. F. Extra-ordinary human-computer interaction. In: **Extra-ordinary human-computer interaction**. New York, NY, USA: Cambridge University Press, 1995. p.3–18.

NORMAN, D. A.; LEWIS, C.; DRAPER, S. W. Introduction. In: NORMAN, D. A.; DRAPER, S. W. (Ed.). **User Centered System Design**: new perspectives on human-computer interaction. Hillsdale, NJ: Erlbaum, 1986. p.1–6.

NOY, N. F.; MCGUINNESS, D. L. **Ontology Development 101**: a guide to creating your first ontology. [S.l.: s.n.], 2001.

OWSLEY, C.; BRUNI, J.; ROENKER, D.; SLOANE, M.; BALL, K. Visual perceptual/cognitive correlates of vehicle accidents in older drivers. In: . [S.l.: s.n.], 1991.

PARTARAKIS, N.; DOULGERAKI, C.; LEONIDIS, A.; ANTONA, M.; STEPHANIDIS, C. User Interface Adaptation of Web-Based Services on the Semantic Web. In: INTERNATIONAL ON CONFERENCEUNIVERSAL ACCESS IN HUMAN-COMPUTER INTERACTION. PART II: INTELLIGENT AND UBIQUITOUS INTERACTION ENVIRONMENTS, 5., 2009, Berlin, Heidelberg. **Proceedings...** Springer-Verlag, 2009. p.711–719.

RDF. **Resource Description Framework**. 2004. Disponível em, `http://www.w3.org/RDF/`.

REINECKE, K.; GAJOS, K. Z. One size fits many Westerners: how cultural abilities challenge ui design. In: WORKSHOP ON DYNAMIC ACCESSIBILITY: DETECTING AND ACCOMMODATING DIFFERENCES IN ABILITY AND SITUATION AT CHI'11, 2011. **Proceedings...** [S.l.: s.n.], 2011.

RFID. **Radio Frequency Identification**. 2011. Disponível em, `http://www.rfid.org`.

SAHA, D.; MUKHERJEE, A. Pervasive Computing: a paradigm for the 21st century. **IEEE Computer**, [S.l.], p.25–31, 2003.

SANTOS, R. C. dos. Systems Usability Evaluation Metrics Review. **Global Business And Technology Association Conference (GBATA)**, [S.l.], 2008.

SEARS, A.; LIN, M.; JACKO, J.; XIAO, Y. When Computers Fade Pervasive Computing and Situationally-Induced Impairments and Disabilities. **Proc HCI Intl**, [S.l.], v.2, p.1298–1302, 2003.

SHNEIDERMAN, B. Designing the user interface: strategies for effective human-computer interaction. In: **In 2nd ed.** Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1992.

SHNEIDERMAN, B. Universal usability. **Commun. ACM**, New York, NY, USA, v.43, p.84–91, May 2000.

SKAF, P.; MATTOS, B. P. E.; BARA, N. M. G. **Guia dos direitos das pessoas com deficiência**. 2007. Disponível em:, http://www.parkinson.org.br/imagens/guia/guia_direito.pdf.

SMITH, M. K.; WELTY, C.; MCGUINNESS, D. M. **OWL Web Ontology Language Guide**. 2004.

SOUCHON, N.; VANDERDONCKT, J. A Review of XML-compliant User Interface Description Languages. In: DSV-IS, 2003. **Anais...** [S.l.: s.n.], 2003. p.377–391.

SQWRL. **SQWRL Wiki**. 2011. Disponível em, `http://protege.cim3.net/cgi-bin/wiki.pl?SQWRL`.

STUDER, R.; BENJAMINS, V. R.; FENSEL, D. Knowledge Engineering: principles and methods. **Data Knowl. Eng.**, [S.l.], p.161–197, 1998.

TREWIN, S.; ZIMMERMANN, G.; VANDERHEIDEN, G. Abstract user interface representations: how well do they support universal access? In: UNIVERSAL USABILITY, 2003., 2003, New York, NY, USA. **Proceedings...** ACM, 2003. p.77–84.

UIML. **Home of the User Interface Markup Language**. 2010. Disponível em, `http://www.uiml.org/`.

USCHOLD, M.; GRUNINGER, M. Ontologies: principles, methods and applications. **Knowledge Engineering Review**, [S.l.], p.93–136, 1996.

USIXML. **Home of the User Interface eXtensible Markup Language**. 2010. Disponível em, `http://www.usixml.org`.

W3C. **W3C**. 2011. Disponível em, `http://www.w3c.org`.

WEISER, M. The computer for the 21st century. **Scientific American**, [S.l.], September 1991.

WEISER, M.; GOLD, R.; BROWN, J. S. The Origins of Ubiquitous Computing Research at PARC in the Late 1980s. **IBM Systems Journal**, [S.l.], p.693–696, 1999.

WELFORD, A. T. Motor Performance. In: **Handbook of the psychology of aging. Van Nostrand Reinhold Co., New York.** [S.l.: s.n.], 1977.

WHO. **Report of the Informal Working Group On Prevention Of Deafness And Hearing Impairment Programme Planning**. 1991. Disponível em, http://www.who.int/pbd/deafness/en/index.html.

WHO. **Care of the Patient with Visual Impairment (Low Vision Rehabilitation)**. 1997. Disponível em, www.aoa.org/documents/CPG-14.pdf.

WOBBROCK, J. The future of mobile device research in HCI. **CHI 2006 Workshop on "What is the Next Generation of Human-Computer Interaction?"**, Montreal, Quebec, Canada, p.131–134, 2006.

XAML. **Home of the Extensible Application Markup Language**. 2010. Disponível em, `http://msdn.microsoft.com/en-us/library/ms752059.aspx`.

XML. **eXtensible Markup Language**. 2011. Disponível em, `http://www.w3.org/XML/`.

YAMIN, A. C. **Arquitetura para um Ambiente de Grade Computacional Direcionado às Aplicações Distribuídas, Móveis e Conscientes do Contexto da Computação Pervasiva**. 2004. 195p. Tese de Doutorado — II/UFRGS, Porto Alegre, RS, Brasil.

ZIEFLE, M.; BAY, S. How older adults meet complexity: aging effects on the usability of different mobile phones. **Behaviour Information Technology**, [S.l.], v.24, n.5, p.375–389, 2005.