SERVIÇO DE POS-GRADUAÇÃO DO ICMC-USP

Data de Depósito: 22.06.2005

Assinatura:

Daniilo Boccardo
()

mCards: Ferramenta de apoio a construção de
aplicativos para assistentes pessoais digitais

Daniilo Rogério Boccardo

Orientadora: *Profa. Dra. Maria da Graça Campos Pimentel*

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências – Área: Ciências de Computação e Matemática Computacional.

USP - São Carlos
Junho/2005

Aluno: Danilo Rogério Boccardo

A Comissão Julgadora:

Profa. Dra. Maria da Graça Campos Pimentel

Prof. Dr. Dilvan de Abreu Moreira

Profa. Dra. Maria Cecília Calani Baranauskas

Universidade de São Paulo

MCARDS: FERRAMENTA DE APOIO A
CONSTRUÇÃO DE APLICATIVOS PARA
ASSISTENTES PESSOAIS DIGITAIS

Resumo

Tecnologias recentes, como a de assistentes pessoais digitais (PDAs), trazem uma série de questões no que tange novos recursos, limitações, possibilidades de utilização e, fatores técnicos relacionados o que freqüentemente demanda amplo tempo de aprendizado. Além de lidar com essas questões, pessoas ligadas à produção de sistemas de software para PDAs precisam também conviver com a diversidade de possibilidades de desenvolvimento e, principalmente com a velocidade com que a tecnologia se modifica. Em tal conjuntura, pesquisas que auxiliem no processo de desenvolvimento, equacionando tempo gasto e resultados efetivos constituem contribuições importantes. Neste trabalho, propõe-se a ferramenta *mCards*, com mecanismos visuais de interação, que permite a criação de aplicações, baseadas em *cards*, para assistentes pessoais digitais.

Abstract

Recent technologies such as PDAs (personal digital assistants) bring new issues with respect to available resources, limitations, possible use, to name a few. These issues, combined with the several software development platforms available, result in many demands relative to learning time and skills, for instance. In this scenario, there is an opportunity for research that support the software development for such platforms. We present *mCards*, a tool that exploits visual interaction to support the development of *card*-based applications for PDAs.

ÍNDICE ANALÍTICO

Introdução	1
1.1 Motivação	1
1.2 Problema	1
1.3 Objetivo e Metodologia	2
1.4 Estrutura do texto	2
Computação Ubíqua	3
2.1 Considerações iniciais	3
2.2 Computação Ubíqua	3
2.2.1 Interfaces Naturais	4
2.2.2 Computação Ciente de Contexto	5
2.2.3 Captura e acesso de informações	5
2.3 Considerações finais	6
User Interface Software	7
3.1 Considerações iniciais	7
3.2 Motivação	7
3.3 Classificações	8
3.3.1 Sistema de janelas (<i>Windowing System</i>)	8
3.3.2 Toolkit	9
3.3.3 Ferramentas de alto nível (<i>Higher-level Tools</i>)	10
3.4 Avaliação de ferramentas	15
3.5 Considerações finais	17
Trabalhos Relacionados	19
4.1 Considerações iniciais	19
4.2 Especificações de Interface abstratas	19
4.2.1 XForms	19
4.2.2 XUL	21
4.2.3 Xaml	22
4.3 Ferramentas de construção de User Interface	23
4.3.1 Hypercard	23
4.3.2 ARTStudio	24
4.3.3 Teresa	26
4.3.4 Editor UIML	27
4.3.5 AUTI	29
4.4 Outros Trabalhos	31
4.5 Considerações Finais	32
MCards	34
5.1 Considerações Iniciais	34
5.2 Motivação	34
5.3 Cenário de uso	35
5.4 Arquitetura	37
5.4.1 Editor Visual	39
5.4.2 Geração de código	42
5.4.3 Controle de Mensagem	43
5.5 Resultados Obtidos	44
5.6 Considerações Finais	45

Conclusão..... 46

6.1 Considerações Iniciais..... 46

6.2 Resultados..... 46

6.3 Trabalhos futuros 47

6.4 Considerações finais..... 48

Bibliografia..... 49

GLOSSÁRIO

Interface

No contexto desse trabalho, empregado como sinônimo da parte do software que representa a interface com o usuário.

Desenvolvente

Usuário com conhecimento de programação

Drag and Drop

Recursos de arrastar e soltar, usado quando se utiliza o mouse como mecanismo de interação.

User friendly

Atributo relativo a quão familiar, atrativo, fácil de utilizar um sistema pode ser.

Toolkits

Facilidade provida para abstrair camada de software mais complexa

Widgets

Componentes visuais de interface com o usuário

Framework

Software com objetivo de reaproveitamento de processos

Proxy

Trata-se de um servidor especial, que roda em uma máquina que pode agir também como se fosse um *Firewall*, escondendo os computadores da rede interna.

JavaBeans

Componentes de Software em Java, projetados para serem unidades reutilizáveis, que uma vez criados podem ser reusados sem modificação de código.

Desktop

Paradigma de hardware, baseado no computador de mesa com monitor, gabinete, teclado e mouse

SIGLAS

PDA Personal Assistant Digital
GPS Global Positioning System
WYSIWYG What Is You See Is What You Get
API Application Program Interface
DTE Diagram Transition Edge
BNF Backus-Naur Form
CAD Computer Aided Design
UIMS User Interface Management System
XML eXtensible Markup Language
XUL XML User Interface Language
XAML eXtensible Application Markup Language
CSS Cascading Style Sheets
DOM Document Object Model
WML Wireless Markup Language
SVG Scalable Vector Graphics
XSLT eXtended Style Sheet Transformations
IDE Integrated Development Environment
JSP Java Server Pages
ASP Active Server Pages
URL Uniform Resource Locators
SAX Simple API for XML

LISTA DE FIGURAS

FIGURA 1 – CAMADAS DO SOFTWARE DE INTERFACE (MYERS, 1995)	8
FIGURA 2 – ESTRUTURA DO XFORMS – DIVIDIDO EM CAMADAS. HTTP://WWW.W3.ORG/MARKUP/FORMS/ ...	20
FIGURA 3 - ARQUITETURA DO LONGHORN. ADAPTADO DE HTTP://WWW.FXCONSULT.COM.BR/XANDO/PDC2003/DEFAULT.ASP	23
FIGURA 4 – PERCURSO REALIZADO POR ARTSTUDIO PARA GERAR OS RESULTADOS DE UM MODELO DE ALTO NÍVEL ATÉ A INTERFACE FINAL (THEVENIN, 2002)	25
FIGURA 5 – EDITOR VISUAL PROPORCIONADO PELA FERRAMENTA EDITOR UIML (MAYORA-IBARRA ET AL, 2003).....	28
FIGURA 6 – ESQUEMA DE FUNCIONAMENTO DA FERRAMENTA EDITOR UIML (MAYORA-IBARRA ET AL, 2003).....	28
FIGURA 7 - ARQUITETURA PROPOSTA POR AUIT (GRUNDY & ZOU, 2002).....	30
FIGURA 8 - AMBIENTE GUI PARA A CRIAÇÃO DE ARTEFATOS AUIT (GRUNDY & ZOU, 2002)	30
FIGURA 9 - DIAGRAMA DE CASOS DE USO DE <i>MCARDS</i>	36
FIGURA 10 - APLICATIVO TÍPICO GERADO POR <i>MCARDS</i>	38
FIGURA 11 - ARQUITETURA E FLUXO DE INFORMAÇÕES DOS MÓDULOS DE <i>MCARDS</i>	39
FIGURA 12 - EDITOR VISUAL DE <i>MCARDS</i>	40
FIGURA 13 - ESTRUTURA DO MÓDULO DE GERAÇÃO DE CÓDIGO DE <i>MCARDS</i>	42
FIGURA 14 - FUNCIONAMENTO DE JAXB (ADAPTADO DE HTTP://WWW.SUN.COM/SOFTWARE/XML/DEVELOPERS/JAXB/INDEX.XML)	43

AGRADECIMENTOS

Agradeço a todos os amigos que participaram direta e indiretamente desse trabalho, aos irmãos de sangue, aos companheiros de pesquisa, aos companheiros de moradia, aos companheiros de refeição, aos companheiros de viagem, enfim, a todos que jornadaaram comigo nessa etapa da minha vida.

Agradeço a minha primeira orientadora Renata Pontin de Mattos Fortes por ter me selecionado para o programa de mestrado e auxiliado nos primeiros momentos.

Agradeço a minha orientadora Maria da Graça Campos Pimentel, pela paciência, compreensão, assistência, orientação e carinho em todas as fases desse trabalho.

Agradeço a minha namorada, por ser a pessoa maravilhosa que é, carinhosa, lutadora, inteligente, linda externamente e internamente, que sempre esteve comigo mesmo estando distante, aceitando e compreendendo.

Agradeço aos meus pais por todo apoio dado e pelo amor incondicional.

Agradeço as cobranças, dificuldades e obstáculos impostos que permitiram o meu crescimento e minha evolução pessoal.

E, principalmente, agradeço ao pai celestial, pela oportunidade concedida.

Introdução

1.1 Motivação

Com o crescente aumento do número de dispositivos portáteis, tanto na área acadêmica como no mercado corporativo, há demanda por pesquisas que explorem as possibilidades desse recente recurso computacional. Acompanhando o crescimento dessa tecnologia, surgem plataformas de desenvolvimento específicas baseadas em linguagens de programação adaptadas e outras totalmente especializadas em que o programador tem de conhecer detalhes específicos da tecnologia envolvida. Apesar da eficácia desse modelo, desenvolvedores¹ tendem a utilizar ambientes conhecidos de desenvolvimento rápido, ou ferramentas que facilitem o processo.

O desenvolvimento para esse tipo de dispositivo enseja uma gama de novas explorações em campos diversos como Interação Usuário-Computador, Hiperídia, Engenharia Web, entre outros, com possibilidades e limitações diferentes das encontradas no paradigma de estação de trabalho tradicional. Aliado a esse campo de pesquisa recente, existe um enorme legado de aplicações Web que anelam por novas formas de interação, principalmente aquelas em que o usuário não estabelece um ponto fixo, nem um momento para interagir, podendo estar em qualquer lugar a qualquer tempo. Mesclando essas duas tendências, emerge um campo de inquirição interessante, a interação entre dispositivos do tipo PDA (Personal Digital Assistant) e sistemas Web.

1.2 Problema

Devido à contemporaneidade da área de dispositivos portáteis, existem poucas ferramentas que auxiliam o processo de desenvolvimento para tais dispositivos, o que dificulta e, principalmente, acarreta um maior tempo de desenvolvimento. Agrava-se o problema quando se tem um mercado de equipamentos quase polarizado, entre a Palm² (PalmOS) e a Microsoft³ (WinCe, PocketPC, etc.) duas plataformas com ambientes de desenvolvimento diferentes, contando cada uma com linguagens de desenvolvimento específicas. Uma das soluções é a utilização de máquinas virtuais, proporcionando uma maior homogeneização no desenvolvimento, no entanto essa abordagem ainda leva a um alto custo de aprendizado e, por consequência, tempo de desenvolvimento.

¹ Usuários com conhecimento de programação e responsáveis pelo desenvolvimento de sistemas de software

² Palm <http://www.palm.com>

³ Microsoft <http://www.microsoft.com>

1.3 Objetivo e Metodologia

O objetivo precípua do trabalho proposto foi estabelecer mecanismos para a construção de aplicações para dispositivos do tipo PDA que fosse independente de plataforma e fizesse uso de manipulação direta de elementos de interação. Nesse sentido, foram investigadas formas de tornar o processo mais intuitivo e automatizado a fim de reduzir o tempo de desenvolvimento bem como a necessidade de conhecimentos específicos.

Diante dessas necessidades, a pesquisa averiguou o conjunto de ferramentas existentes que auxiliam no processo de desenvolvimento, especificamente as que apóiam a construção de interfaces, analisou-as, e, então, propôs a ferramenta *mCards*, um construtor de aplicativos para dispositivos do tipo PDA, baseado em mecanismos de manipulação visual, com geração de código automática. Para atingir tal objetivo, a ferramenta oferece ao desenvolvedor um ambiente com uma série de componentes visuais que podem ser utilizados por meio de manipulação direta.

1.4 Estrutura do texto

O restante desta dissertação está estruturado da seguinte maneira. No capítulo 2 faz-se uma sucinta descrição sobre computação ubíqua, descrevendo seus principais conceitos e áreas de atuação. O capítulo 3 explora os conceitos relacionados a software para desenvolvimento de interfaces com o usuário, classificando e contextualizando ferramentas. Após essa contextualização, no capítulo 4, são descritas diversas ferramentas e tecnologias relacionadas diretamente a este trabalho, e mais especificamente com *mCards*. Este é detalhado no capítulo 5, sendo apresentadas suas razões, cenários de uso, arquitetura e resultados obtidos. Por fim, no último capítulo são sumarizadas as conclusões alcançadas pelo trabalho.

Computação Ubíqua

2.1 Considerações iniciais

Este capítulo faz uma breve descrição da área de computação ubíqua, discutindo as questões essenciais da área bem como suas sub-áreas de interfaces naturais, computação ciente de contexto e captura e acesso de informações. A partir desta contextualização, ao final do capítulo é delineada a relação entre o presente trabalho e a área de computação ubíqua.

2.2 Computação Ubíqua

Mark Weiser, em seu artigo “*Computer of the 21st century*” (Weiser, 1991), vislumbrou um novo paradigma para a computação, designando o termo Computação Ubíqua à sua visão. Segundo ele, a computação restrita a mainframes e *desktops* deveria deslocar-se para o nosso dia-a-dia, auxiliando nas atividades cotidianas de forma transparente. Para isso, os ambientes seriam instrumentados com dispositivos computacionais interconectados de diversos tamanhos, cada um atendendo uma determinada necessidade. Como resultado, o uso desses dispositivos sairia do foco de atenção do usuário, a fim de que a interação ocorresse de forma transparente.

Partindo do pressuposto que o mundo físico possui objetos de diversos tamanhos, e a computação ubíqua demanda a instrumentalização do ambiente físico, três escalas de dispositivos foram identificados: polegadas, pés e jardas. A primeira escala corresponde a pequenos dispositivos, que cabem na palma da mão, tal como PDAs (*Personal Digital Assistants*) e *Tabs*⁴. Um ambiente teria centenas de dispositivos dessa escala, cada um atendendo uma determinada finalidade, permitindo a relação de vários dispositivos para cada pessoa. A segunda escala são os dispositivos de tamanho médio, do tamanho de uma pasta, tal como *tablets*⁵, *Pads*⁶ e *notebooks*, em que a relação seria de um dispositivo para cada pessoa. Estes dispositivos seriam como cadernos pessoais, podendo ser anotados e carregados de um lugar para outro. A última escala seria de dispositivos de maior porte, como *WhiteBoards*⁷ e *LiveBoards*⁸, cuja finalidade seria atender diversos usuários, sendo a relação um dispositivo para várias pessoas.

Instrumentar ambientes com artefatos computacionais é, nessa visão, somente o primeiro passo. O segundo consiste em descobrir uma nova maneira de utilizar os computadores, uma forma que leve em conta o ambiente natural humano e que permita aos computadores

⁴ Dispositivos computacionais semelhante e antecessores dos PDAs

⁵ Dispositivos computacionais portáteis com uma tela sensível ao toque, de tamanho similar a uma prancheta.

⁶ Dispositivo parecido com um *notebook*, desenvolvido no Xerox Parc.

⁷ Lousa eletrônica sem projeção, necessita de um projetor.

⁸ Lousa eletrônica com projeção.

ficarem ocultos, embutindo-os nos objetos do dia-a-dia. O objetivo é colocar o computador em *segundo plano* na interação, deixando as pessoas investirem em uma relação humana, sem a intrusão de computadores (Weiser, 1993). Weiser compara tal fenômeno ao desaparecimento dos motores, que encolheram até passarem a fazer parte de pequenos objetos do dia-a-dia, sendo, de fato, invisíveis aos olhos. Esta etapa é realmente complexa, pois cria uma nova forma de interação entre pessoas e computadores, cujos conceitos e aplicações estão por ser estabelecidos, envolvendo áreas tecnológicas e sociais.

Essa nova forma de pensar os computadores exige novas aplicações. De acordo com Abowd & Mynatt (2000), aplicações de computação ubíqua podem ser divididas em três temas: interfaces naturais, computação ciente de contexto, e captura e acesso a informações. As seções seguintes abordam cada um desses temas.

2.2.1 Interfaces Naturais

Um dos objetivos da computação ubíqua é tornar a interação do usuário com ambientes instrumentalizados a mais natural possível. Dessa forma, a questão de interfaces naturais tem um papel central no desenvolvimento de aplicações ubíquas. Atualmente as interfaces para computação se baseiam no paradigma convencional (teclado, mouse, display), no qual o usuário explicitamente interage com uma caixa. Isto não representa a idéia de transparência e interação implícita definida pela computação ubíqua. Assim, é preciso repensar o modo de interação humano-máquina, buscando construir interfaces que se adaptem às características do ambiente humano, e não o contrário (Weiser, 1994).

Existem diversas tentativas de engendrar interfaces mais naturais ao ambiente das pessoas. A inclusão de dispositivos computacionais em objetos de uso diário (lousas, canetas, murais, objetos etc.), tal como a reprodução de ações naturais ao ser humano (fala, escrita, gestos) representa algumas delas. Harrison et al (1998), por exemplo, faz uso de sensores em dispositivos computacionais do tipo PDA, possibilitando que a manipulação física desses dispositivos seja interpretada apropriadamente pelas aplicações que o controlam. Ishii & Ullmer (1997) foram além e vislumbraram, como forma de manipular artefatos eletrônicos, técnicas de interação baseadas em objetos do dia-a-dia, denominadas interfaces tangíveis.

Diversas pesquisas, desde então, investigam o campo das TUIs (Tangible User Interfaces), em diversos contextos, desde paradigmas para especificação e construção (Calvillo et al., 2003) até formas de utilizar interfaces tangíveis como mecanismo de aprendizado para crianças (Africano et al., 2004).

2.2.2 Computação Ciente de Contexto⁹

Uma área explorada pela computação ubíqua baseia-se na coleta de informações do ambiente, de forma que elas possam ser utilizadas por aplicações. O objetivo é tornar as aplicações cientes das informações existentes no ambiente, para que apresentem um comportamento dinâmico em sua execução, possibilitando, portanto, uma interação humano-máquina mais adaptadas as necessidades das pessoas (Dey & Abowd, 2000).

Informações de contexto poderiam ser explicitamente solicitadas pelas aplicações, no entanto, esta tarefa se torna, muitas vezes, cansativa ao usuário. Segundo Dey & Abowd (2000), aplicações cientes de contexto devem obter informações de contexto de modo automatizado, disponibilizando-as em um ambiente computacional em tempo de execução, e deixando a tarefa de decidir a relevância ou não da informação aos responsáveis pelo desenvolvimento.

As primeiras informações de contexto exploradas computacionalmente baseavam-se na localização espacial. Trabalhos pioneiros, como *Active Badge* (Want et al. 1992) e *ParcTab* (Want et al. 1995), tentavam prover facilidades e serviços significativos mediante a localização dos usuários em um escritório. Outros, tal como *Cyberguide* (Abowd et al., 1997), empregam o sistema GPS (Global Positioning System) em conjunto com PDAs, objetivando a captação de informações espaciais e a melhoria dos serviços turísticos.

A informação espacial tem sido bastante explorada, dado que os recursos para sua captação estejam bem difundidos. Contudo, além da localização, existem outras informações de contexto baseadas na identificação de pessoas, objetos, bem como no aspecto temporal, entre outros. Abowd & Mynat (2000) utilizam o conceito das cinco dimensões - os “cinco W’s” (Where, Who, What, When, Why), para captar a noção de como obter informações de contexto.

2.2.3 Captura e acesso de informações

Uma aplicação recorrente em computação ubíqua é a captura de experiências do cotidiano, tornando os registros disponíveis para o acesso posterior. A motivação é dar mais liberdade às pessoas para aproveitarem a experiência de fato sem se preocuparem em estar guardando os eventos de forma explícita (Abowd, 1999). Realmente, dispositivos computacionais oferecem a possibilidade de armazenamento de muitos dos eventos gerados na experiência, uma característica improvável nas pessoas, devido à experiência ser caracterizada por múltiplos fluxos de informações (áudio, vídeo, imagens, textos).

Essa liberdade propicia às pessoas um melhor aproveitamento das experiências, assim como a possibilidade de, posteriormente, consultá-las para ratificação ou retificação de possíveis

⁹ O termo contexto é empregado para designar informações de contexto

modelos mentais errados. Diversos trabalhos, em diferentes ambientes, têm sido propostos abordando essas idéias. Alguns exemplos são *Tivoli* (Pedersen et al., 1993), *Dolphin* (Streitz et al., 1994) e *TeamSpace* (Ritchter et al., 2001) em ambientes de reunião; *ClassRoom* (Abowd et al., 1998), *eClass* (Abowd et al., 2000) e *iClass* (Cattelan et al., 2003) em ambientes de ensino.

2.3 Considerações finais

Pesquisa e desenvolvimento em computação têm buscado formas de facilitar e melhorar a vida humana. A computação ubíqua, ante a isso, procura expandir esses objetivos, tornando os recursos computacionais onipresentes e integrados à vida humana. De fato, muitas aplicações ubíquas buscam trazer ao ambiente das pessoas mecanismos que permitam um melhor aproveitamento de suas experiências, utilizando, para isso, informações contextuais e interfaces naturais, de forma a evitar que tais mecanismos obstruam a experiência. Para alcançar tal objetivo é necessária uma infra-estrutura capaz de prover as condições para a consecução dos objetivos da computação ubíqua.

A questão não se resume a criar aplicações ubíquas, mas também a prover os meios de se criar tais aplicações. Assim, existe uma necessidade de ferramentas que ajudem no processo de criação de tais aplicações. Salienta-se ainda que essas ferramentas devem refletir a idéia de transparência da computação ubíqua, não estritamente o sentido literal, mas a idéia que o uso dessas ferramentas não obstrua o objetivo maior proposto, construir aplicações ubíquas. Assim, ferramentas ubíquas, fáceis de utilizar, com modelos intuitivos de interação, são necessárias na área de computação ubíqua.

No que concerne este trabalho, foi desenvolvida a ferramenta *mCards* que ajuda na criação de aplicações para dispositivos portáteis do tipo PDA, mais especificamente aplicações baseadas em formulários, estabelecendo, outrossim, mecanismos de comunicação sem fio, um dos recursos proeminentes em aplicações ubíquas. A partir de um modelo de interação baseado na técnica *Drag and Drop* de componentes visuais, *mCards* facilitar a criação de aplicações para PDAs, dispositivo incluído em uma das escalas discutidas por (Weiser, 1991).

Ao possibilitar a criação de aplicações para PDAs, bem como a comunicação entre o repositório móvel e um localizado na Web, por meio de comunicação sem fio, o presente trabalho insere-se na área de aplicações de apoio a aplicações ubíquas. No entanto, o trabalho tenciona a tarefa de construção de aplicações tornando-a mais natural ao usuário.

User Interface Software

3.1 Considerações iniciais

Este capítulo sumariza o estado da arte em termos de ferramentas de apoio a construção de interfaces com o usuário, almejando contextualizar o assunto e como explicitar uma terminologia para que, nos próximos capítulos, o trabalho possa ser situado na área em questão. Por tais ferramentas entende-se, no contexto deste trabalho, um conjunto de sistemas e ferramentas que ajudam no desenho e/ou implementação de interfaces com o usuário de um software. No restante do trabalho apenas o termo interface será utilizado.

Entende-se por interface, em um sistema interativo, a parte de uma aplicação responsável por controlar o diálogo entre o usuário e o sistema, sendo a outra parte denominada aplicação. Formalmente pode-se defini-la como parte do software de um sistema interativo responsável por traduzir ações do usuário em ativações das funcionalidades do sistema (aplicação), permitindo que os resultados possam ser observados e coordenar essa interação (Hartson & Hix, 1989). A divisão do software entre interface e aplicação proporciona o desacoplamento no processo de desenvolvimento dessas partes, ensejando uma maior especialização e conseqüentemente melhores resultados.

Nesse contexto, existem inúmeros esforços, concretizados nas ferramentas de apoio a construção de interfaces, que buscam apoiar o desenvolvimento de interfaces, algumas dedicadas à concepção e consecução do projeto, outras imbuídas da implementação, havendo, também, ferramentas que apóiam todo o ciclo de vida de desenvolvimento.

O capítulo está estruturado da seguinte maneira. Na seção 3.2 comentam-se as motivações para uso de ferramentas no processo de desenvolvimento de interfaces. Posteriormente, em 3.3, descreve-se uma classificação do software usado para se criar interfaces, desde sistemas de baixo nível até as ferramentas de apoio ao desenvolvimento. Na seção 3.4, descreve-se um conjunto de questões propostas por Myers (1995) e Myers et al (2000) para estimar ferramentas que apóiam a construção de interfaces, confrontando-as com a ferramenta implementada. Por fim, nas considerações finais, realiza-se uma síntese do capítulo, com alguns comentários que aludem ao trabalho realizado.

3.2 Motivação

Para usuários, a interface é o sistema (Hix & Hartson, 1993). Tal idéia reflete a relevância da interface dentro de um software. Questões como facilidade de aprendizado, usabilidade, *user friendly* de um sistema interativo são determinadas pela interface, ou seja, tem um grande peso

na produtividade do usuário. De fato, muitas vezes, um sistema com mais recursos e maior eficiência é preterido em razão de sua interface.

De fato, estima-se que 48 % do código de um sistema, e 50% do tempo de implementação relacionam-se com a parte da interface (Myers & Rosson, 1992). Nesse ponto o uso de ferramentas torna-se vital, pois permitem reduzir a quantidade de código que programadores precisam produzir. Outro ponto é a possibilidade de desenvolvimento de interfaces mais rapidamente. Isso habilita a prototipação rápida, uma característica fundamental no processo de desenvolvimento de interfaces de alta qualidade.

Segundo Nielsen (1993), o desenvolvimento de interfaces é um processo iterativo, requerendo uma contínua avaliação que realmente o próprio projeto. Para tal fim, a possibilidade de rápida prototipação enseja mais interações, permitindo que a interface seja refinada a cada ciclo de interação e avaliação. Com base nesses argumentos, o desenvolvimento de ferramentas que ajudem no projeto de interfaces é de grande interesse no campo de interfaces.

3.3 Classificações

Myers (1995) divide o software de interface em três camadas: sistema de janelas (*Windowing System*), *toolkits* e ferramentas de alto nível. As duas primeiras camadas fornecem facilidades, de baixo nível, para o manuseio de telas e *widgets* respectivamente, enquanto a terceira abstrai as duas primeiras, permitindo uma manipulação de alto nível. A Figura 1 ilustra a idéia de camadas, em que uma camada superior utiliza os serviços da camada abaixo.

Figura 1 – Camadas do software de interface (Myers, 1995)

3.3.1 Sistema de janelas (*Windowing System*)

Consiste em um pacote de software que possibilita ao usuário controlar diferentes contextos, mediante a separação física do espaço virtual, tela disponível, em diferentes regiões, geralmente retangulares, chamadas janelas. Essa camada gerencia recursos típicos de um

ambiente de janelas como eventos, criação e remoção de janelas, dispositivos de entrada (mouse, teclado) e saída (vídeo).

O ambiente Smalltalk (Kay, 1993) foi um dos primeiros sistemas de janelas, permitindo o uso de janelas sobrepostas, podendo essas serem selecionadas e transladadas com o uso do mouse. A primeira versão desse ambiente foi testada no Alto, um protótipo de computador desenvolvido, no Xerox PARC¹⁰ (Palo Alto Research Center), com conceitos inovadores para a época como *desktop*, *windows*, e *WYSIWYG*. Posteriormente, em 1984, surgiu o Apple Macintosh¹¹, e, um ano após, o Windows¹² que foi uma tentativa de suprir os computadores mais difundidos da época, o IBM-PC, com recursos semelhantes àqueles que os usuários dos computadores supracitados possuíam.

Os recursos fornecidos pelo sistema de janelas são de baixo nível, exigindo um esforço substancial do desenvolvedor, principalmente no que se refere ao tempo requerido de desenvolvimento. Assim, com o objetivo de facilitar o manuseio de sistemas de janelas, foram criados os toolkits, camada que abstrai os recursos oferecidos pelos sistemas de janelas, facilitando com isso sua manipulação.

3.3.2 Toolkit

No contexto de interfaces com o usuário, *toolkits* consistem em uma biblioteca de *widgets* que podem ser utilizados por desenvolvedores de aplicações para a criação de interfaces, gerenciado por uma arquitetura que controla as operações da interface provenientes dos *widgets*. Um *widget* consiste em um componente visual caracterizado como um objeto de interação com apresentação e comportamento bem definidos. Tipicamente, abrangem *menus*, *buttons*, *scroll bars*, *textfields*, etc.

Esses componentes visuais potencializam a reutilização, em que objetos bem definidos e testados podem ser agrupados para formar uma interface com o usuário. O objetivo é prover a programadores uma base, em termos de implementação, necessária para a criação de interfaces. De fato, o uso de *widgets* mais a abstração da camada de sistema de janelas, torna o processo de engendrar interfaces mais produtiva a medida que detalhes de baixo nível deixam de ser responsabilidade do desenvolvedor.

Ademais o uso de *toolkits* permite maior uniformização na produção de interfaces de usuário, pois o uso de um mesmo *toolkit* produzirá resultados semelhantes devido ao uso de componentes visuais semelhantes. Essa propriedade é importante, quando se objetiva consistência, contudo, limitam-se os estilos de interação a aqueles disponíveis pelo *toolkit*,

¹⁰ Xerox Parc <http://www.parc.xerox.com/>

¹¹ Apple Macintosh: <http://www.apple.com/hardware/>

¹² Windows: <http://www.microsoft.com/windows/default.msp>

sendo a criação de novos componentes um processo complexo, com muitas interdependências.

3.3.3 Ferramentas de alto nível (*Higher-level Tools*)

Conquanto os benefícios do uso de *toolkits*, quando se levam em conta questões como consistência e reusabilidade, sejam incontestáveis, muitas vezes faz-se necessário uma maior facilidade e automatização no processo de desenvolvimento de interfaces. *Toolkits* geralmente proporcionam uma interface procedural, baseada em APIs, o que restringe o seu uso a programadores. Ademais, *toolkits* normalmente contêm centenas de procedimentos relativos a seus *widgets*, tornando o seu uso trabalhoso mesmo a programadores. Assim, existe um enorme interesse em ferramentas menos restritivas, com técnicas mais intuitivas de desenvolvimento.

De acordo com Myers (1995), a importância de se utilizar tais ferramentas devem-se a dois fatores: (1) qualidade maior da *interface*; (2) código da *interface* mais fácil e mais econômico para se criar e manter. Como argumentos para tais afirmações, Myers, no que se refere à questão (1), cita:

- Possibilidade de rápida prototipação e implementação;
- Facilidade de incorporar mudanças obtidas mediante experimentações do usuário;
- Múltiplas interfaces para uma mesma aplicação;
- Consistência na criação de várias aplicações, pois provém de uma mesma ferramenta;
- A construção da ferramenta pode utilizar conhecimentos de especialistas em campos de conhecimento variados;

Para a questão (2), Myers argumenta:

- Menos código para se produzir e testar;
- Maior modularização, devido à separação da interface com o usuário da lógica da aplicação;
- Menor exigência no nível de especialização do desenvolvedor;
- A confiabilidade da interface será maior, desde de que o código é criado automaticamente a partir de uma especificação em alto nível;
- Facilidade de se portar uma aplicação para diferentes dispositivos;

Existem diversas classificações para ferramentas que apoiam a construção de *interfaces*. Uma bastante difundida as classifica de acordo com a fase de atuação. Por exemplo, podem auxiliar na construção da interface mediante um editor gráfico ou um compilador que processa uma especificação da interface. Podem dedicar-se ao tempo de execução, em que o usuário está efetivamente usando a interface. Podem ainda apoiar a fase de pós-execução cuja finalidade seria depurar e avaliar a interface. Quando a ferramenta atua nas três fases,

principalmente gerenciando a interface em tempo de execução, nomina-se *User Interface Management System*. Maiores informações sobre UIMS podem ser encontradas em Olsen (1992).

No que se refere este trabalho, maior atenção é dada à fase de construção de interfaces, em função do relacionamento direto de tal enfoque com o presente trabalho. Essas ferramentas são subdivididas em:

- Baseadas em linguagens;
- *Frameworks*;
- Geração automática baseada em modelo;
- Especificação gráfica direta.

Essa classificação não restringe uma ferramenta a uma classe. Normalmente, uma ferramenta pode ter características que a enquadrem em mais de uma classe. O importante é a sua característica principal que corresponde ao fator relevante para a sua inclusão em uma determinada classe. Nas próximas seções, cada uma dessas classes e suas ramificações, quando existentes, serão descritas.

3.3.3.1 Baseada em linguagens

Correspondem às mais antigas ferramentas, em que se requer do desenvolvedor a utilização de uma linguagem específica para construir a interface. Existem muitas notações utilizadas por essas linguagens, sendo as principais baseadas em diagramas de transição de estados, em gramáticas, em linguagens declarativas e em eventos. A ideia geral consiste em especificar, a partir da linguagem e de sua sintaxe, a sequência de ações de saída e entrada. A essa sequência denomina-se diálogo. É importante salientar que a notação pode vir acompanhada de ferramentas de apoio para a construção e execução da interface.

A notação baseada em redes de transição foi amplamente utilizada, sendo o diagrama de transição de estados (DTEs) a mais representativa. Nesse modelo, a interface é considerada como uma coleção de estados ligados por arestas rotuladas com eventos. Associam-se estes eventos a ações do usuário, tempo e saída da interface, consistindo os eventos, a condição necessária para que ocorra uma transição entre estados. Uma característica interessante do DTEs é a sua representação gráfica. Um dos problemas dessa abordagem é o seu uso na construção de sistemas altamente interativos, em que determinado instante, existem diversas opções de escolha ao usuário. Em tais sistemas, a notação exigiria uma vasta quantidade de arcos em um determinado estado. Também, diagramas de estado não podem controlar situações em que o usuário pode operar sobre múltiplos objetos ao mesmo tempo.

Notações baseadas em gramáticas representam o diálogo entre o usuário e a aplicação mediante uma linguagem, restrita por uma gramática. A gramática é estruturada em BNF

(Backus-Naur Form), amplamente empregada na especificação formal da sintaxe de linguagens de programação. Da mesma forma que a notação de transição de estados, gramáticas não são apropriadas para especificar sistemas altamente interativos, pois são orientados por processamento de *strings* com estruturas sintáticas complexas. Atualmente, essa abordagem é pouco utilizada, quase inexistente para a construção de interfaces.

Linguagens baseadas em eventos oferecem facilidades para a descrição de diálogos complexos. Nessa abordagem, a interface é vista como um conjunto de eventos e tratadores para esses eventos. Na ocorrência de um evento significativo como a manipulação de um *widget* de entrada pelo usuário, esse evento é colocado em uma estrutura de dados, e enviado para um tratador de eventos relacionado ao tipo de evento gerado. Este possui o código necessário para apropriadamente responder a ação do usuário. A resposta pode causar uma saída externa, uma mudança no estado interno do sistema, ativando outro tratador de eventos, ou apenas chamar uma rotina da aplicação.

Diferente das abordagens anteriores, linguagens baseadas em eventos permitem controlar sistemas de alta interatividade, à medida que possibilita controlar múltiplas opções de escolha do usuário, em um determinado instante. Têm sido amplamente utilizadas, com sucesso, principalmente, em sistemas gráficos que utilizam manipulação direta de componentes visuais, em que são gerados eventos para cada ação que o usuário pratica, utilizando o teclado ou o mouse.

Outro modo bastante difundido de se especificar interfaces consiste na utilização de linguagens declarativas. A idéia é tentar declarar o que deve acontecer ao invés de especificar o procedimento de como fazer acontecer, ou seja definir em alto nível a interface. Vem sendo bastante utilizados no apoio a construção de formulários que correspondem a um conjunto de *widgets* de entrada de dados (*textfields*, *options*) auxiliados semanticamente por itens textuais e gráficos.

Também, vêm sendo usadas na descrição do *layout* de interfaces, em conjunto com muitos *toolkits*. Por exemplo, Motif's User Interface Language (UIL) permite a definição do layout de *widgets*. Como UIL é interpretado, o usuário pode personalizar a interface editando o código UIL, em vez de criar a interface programando de forma usual. Essa técnica tem tido grande aceitação, e no capítulo dos trabalhos relacionados será exemplificada com XUI.¹³ e XAML.¹⁴ O uso de linguagens declarativas tem a vantagem que o desenvolvedor não precisa se preocupar com a sequência de eventos, concentrando-se nas informações que precisam ser trocadas. A desvantagem é a limitação nos tipos de interfaces que podem ser criadas, já que as formas de interação são pré-programadas. No entanto, essas linguagens vêm sendo

¹³ XUI, <http://www.xulplanet.com>

¹⁴ XAML, <http://msdn.microsoft.com/longhorn/>

amplamente utilizadas como linguagens intermediárias. Muitas pesquisas têm estendido as linguagens declarativas para apoiar interações mais sofisticadas, emergindo modelos mais abrangentes. A seção 3.3.3.3 trata desses modelos.

3.3.3.2 Frameworks

Não existe uma definição formal dominante para *framework*, mas, compendiando diversos conceitos, pode-se dizer que consiste em um software auxiliante em determinado domínio, com estruturas bem definidas, estabelecidas a partir de um amplo conhecimento do domínio em questão. A característica mais marcante é o estímulo ao reuso não apenas o reaproveitamento de código, mas também os esforços de todas as fases do processo de desenvolvimento de software. Não se restringem ao domínio de interface com o usuário, apesar de esse ser o primeiro domínio em que foram utilizados.

Em geral, tem a função de guiar o desenvolvedor no ofício de desenvolver a aplicação. Para isso, partes importantes do domínio ficam a cargo do *framework*. No contexto de criação de interfaces, classes são providas para o controle de partes essenciais como janelas, comandos, eventos, etc. O desenvolvedor encarrega-se de especializar tais classes, ou estendendo uma classe, ou utilizando uma classe padrão, municiando, então, a aplicação com detalhes, tal como o que vai ser colocado nas janelas, os comandos necessários, os eventos gerados.

MacApp (Schmucker, 1986) (Wilson, 1990) foi pioneiro em utilizar essa abordagem no caso para simplificar a escrita de aplicações para Macintosh. Da mesma forma *Unidraw* (Vlissides & Linton, 1990), mas orientado a construção de editores gráficos e programas CAD. Citam-se também importantes referências no contexto Andrew Toolkit, da Universidade Carnegie Mellon (Paley et al., 1988) e InterViews, de Stanford (Linton et al., 1989)

Os *frameworks* são uma tendência nas mais diversas áreas de computação com ampla utilização. Lidam satisfatoriamente com complexidades, tempos de desenvolvimento e padrões de projeto, diminuindo os custos de desenvolvimento ao mesmo tempo em que garante, na medida do possível, qualidade.

3.3.3.3 Geração automática baseada em modelos

Consistem em ferramentas que possibilitam a geração automática da interface, a partir de uma especificação de alto nível. Diferente de ferramentas baseadas em linguagem e de *frameworks*, o desenvolvedor não se preocupa em criar a interface diretamente, mas sim por meio de um modelo de alto nível que atenda os requisitos pretendidos pelo usuário final, relegando os detalhes de implementação para a ferramenta. Esses modelos permitem mudar o foco de desenvolvimento, saindo do projeto centrado nos requisitos da aplicação para um projeto centrado no usuário final.

Por exemplo, UIDE (User Interface Design Environment) (Sukaviriya et al. 1993) requer que a semântica da aplicação seja definida em uma linguagem especial. No entanto essa linguagem descreve as funções que a aplicação deve prover e não a interface pretendida. UIDE classifica-se como baseada em modelo, porque a linguagem especifica a semântica da aplicação em alto nível, encarregando-se a ferramenta de processar instâncias desse modelo e gerar a interface correspondente. Outra sistema baseado em modelo é o Humanoid (Szekely et al., 1993), que apóia a modelagem da apresentação, do comportamento e do dialogo de uma interface. A linguagem de modelagem do Humanoid inclui abstrações, composições, recursões, iterações, e construções condicionais com o objetivo de apoiar a construção de interfaces sofisticadas.

Ferramentas baseadas em modelos são atraentes, principalmente por permitirem geração automática de interfaces, e, também, pelo fato de desenvolvedores inexperientes poderem construir interfaces, somente descrevendo as funcionalidades. No entanto elas não tem tido uma grande aceitação devido à imprevisibilidade dessas ferramentas quanto ao resultado, ou seja, interface gerada. A relação entre o modelo de alto nível e a interface gerada, geralmente, não é bem compreendida pelo desenvolvedor o que acarreta, muitas vezes, baixa satisfação. Devido à dificuldade de se gerar automaticamente interfaces, há uma limitação quanto ao tipo de interface que pode ser produzida. Geralmente, o resultado não é tão bom como uma interface feita com técnicas convencionais de programação.

3.3.3.4 Especificação Gráfica Direta

Tratam-se de ferramentas que permitem especificar a interface, ao menos parcialmente, manipulando objetos visuais na tela. Fundamenta-se no fato que a apresentação visual é uma característica essencial de uma interface, sendo, portanto, tais ferramentas mais apropriadas para defini-la. A vantagem dessas ferramentas é a facilidade de uso, podendo, grande parte das vezes, serem usadas por pessoas leigas em termos de programação. Especificação gráfica direta vem sendo, freqüentemente, denominada programação de manipulação direta visto que o usuário manipula diretamente os widgets da interface e seus elementos.

Myers (1995) subdivide as ferramentas baseadas em programação de manipulação direta em:

- *Ferramentas de prototipação*
- *Cards*
- *Interface Builders*

As ferramentas de prototipação caracterizam-se por proporcionar ao desenvolvedor a construção rápida de interfaces, ensejando um grande número de instancias de exemplo. Usualmente essas ferramentas não são apropriadas para criar a interface definitiva, mas servem como uma base para se contemplar aspectos da interface projetada. Com efeito, muitas partes das interfaces que são geradas nessas ferramentas não são operáveis, com

widgets podendo ser apenas ilustrações estáticas. A desvantagem dessas ferramentas é que elas não geram o código da interface, necessitando codificação posterior.

Existem aplicações que utilizam interfaces restritas a uma sequência de partes estáticas interligadas, recebendo, algumas vezes, o nome de *cards*, *forms* ou *frames*. O mais famoso sistema que emprega essa idéia é o Hypercard da Apple. Nesse sistema, o desenvolvedor pode criar *cards* contendo variados *widgets* (*textfields*, *buttons*, etc) juntamente com adornos gráficos. Também é disposta uma linguagem de *scripting*, chamada *Hypertalk*, para gerenciar os potenciais eventos de determinados *widgets*. O HyperTalk corresponde a uma linguagem baseada em eventos, já tratada na seção 3.3.3.1. Um recurso interessante é o uso de *links* que possibilita a ligação de dois *cards* mediante um *button*.

Interface Builders são ferramentas interativas que permitem *widgets* pré-selecionados sejam posicionados na tela por meio do mouse, com o intuito de construir janelas, menus e caixas de diálogo da interface. Tais ferramentas tem sido amplamente utilizadas, com grande êxito, principalmente devido ao fato de relacionar o resultado interface gráfica com o meio empregado meio gráficos. Remove-se a codificação da interface, insere-se o desenho da interface, tornando o desenvolvimento mais intuitivo, e, portanto mais acessível para não programadores. Mesmo a programadores, ganha-se em velocidade no que tange o desenvolvimento.

Existem muitos exemplos de Interface Builders, entre eles incluem-se editores de recursos ou construtores de linguagens como Visual Basic, Java, C++, etc. Pioneiros como Trillium (Henderson, 1986) do Xerox PARC e MenuLay (Buxton et al., 1983). A maioria gera código baseado em *templates*, outros geram uma descrição da interface para ser interpretada posteriormente.

Embora os benefícios creditados a *Interfaces Builders* no tocante a facilidade, universalização e velocidade de desenvolvimento, tais ferramentas não possuem qualquer mecanismo que oriente a produção de interfaces qualitativas, que satisfaçam requisitos almejados. Tal responsabilidade fica a cargo do usuário, visto sua liberdade quanto a criação. Com efeito, *Interface Builders*, freqüentemente, constituem uma parte de um sistema maior, sendo utilizados conjuntamente com linguagens baseadas em eventos, especificações baseadas em modelos, frameworks, etc.

3.4 Avaliação de ferramentas

Segundo Myers et al (2000), as seguintes funcionalidades deveriam ser providas por uma ferramenta que visa apoiar a construção de interfaces com o usuário:

- Ajudar o desenho da interface a partir de uma especificação de tarefas do usuário
- Ajudar a implementar a interface a partir de uma especificação de projeto
- Ajudar a avaliar a interface depois do seu desenvolvimento e propor melhorias ou, ao menos, prover informações que permitam ao desenvolvedor avaliar a interface.
- Criar interfaces de fácil utilização
- Permitir ao desenvolvedor investigar diferentes desenhos rapidamente
- Permitir leigos desenhar e implementar interfaces
- Permitir ao usuário final personalizar a interface
- Prover portabilidade

Na realidade, dificilmente uma ferramenta conseguirá prover todas essas funcionalidades propostas, mas tais itens representam uma importante fonte de avaliação de uma nova ferramenta. No que se refere à ferramenta *mCards*, resultado direto do presente trabalho, somente os três primeiros itens não fazem parte dos objetivos, embora sejam tratados como trabalhos futuros. Os outros se relacionam fortemente com os objetivos centrais da pesquisa, principalmente a questão de facilitar o desenvolvimento, proporcionando a pessoas não especialistas da área de desenvolvimento de software a possibilidade de criação de aplicativos.

Myers et al (2000) também, indicam algumas questões importantes que determinaram o sucesso de ferramentas passadas e que provavelmente definirão o sucesso de futuras. Na sua visão, os seguintes itens deveriam ser analisados com atenção:

Foco de atenção

Entende-se por foco de atenção, a constatação de que ferramentas de sucesso, geralmente, auxiliam onde são efetivamente necessárias. Definir esse alvo e dedicar-se completamente a ele, eis um dos caminhos do êxito. *mCards* tem como objetivo auxiliar na construção de aplicativos para dispositivos portáteis, tecnologia recente que demanda por ferramentas especializadas.

Threshold e Ceiling

Threshold corresponde a quão difícil é aprender a usar a ferramenta, e *Ceiling* o quanto pode ser feito usando a ferramenta. A maioria dos atuais sistemas de sucesso combina ou baixo *threshold/ceiling*, ou alto *threshold/ceiling*. O ideal seria unir um baixo *threshold* com alto *ceiling*, sendo, portanto, este o objetivo a ser alcançado por qualquer novo sistema. No que se refere a *mCards*, de fato, um dos objetivos é tornar o *Threshold* mínimo, por mais que essa tarefa seja subjetiva. Ao mesmo tempo, a ferramenta ao combinar uso de Cards como elemento de construção, edição visual com mecanismos de manipulação direta, controle de persistência e comunicação automática ensina o desenvolvimento de aplicativos complexos.

Direcionamento Qualitativo

Ferramentas influem no tipo de interfaces que podem ser criadas. Assim, direcionar o desenvolvimento no caminho certo para produzir artefatos de qualidade, por meio de guias e dicas, e privá-lo da direção errada, trata-se de um fator relevante no sucesso de uma ferramenta. No que se refere esse tema, *mCards* tem controle sobre erros, pois gera código automático, de acordo com a manipulação efetuada pelo usuário em um editor visual. Quanto ao direcionamento à qualidade, existe um forte aspecto subjetivo inerente à tarefa que a ferramenta não tem controle. Na realidade, a ferramenta não interfere no processo de criação, dando ao usuário total liberdade a usar sua criatividade.

Previsibilidade

Previsibilidade diz respeito à capacidade do desenvolvedor, ao usar a ferramenta, presumir o resultado de suas ações. De acordo com Myers et al (2000), ferramentas com técnicas automáticas geralmente não conseguem sucesso, devido, muitas vezes, a imprevisibilidade. *mCards* automatiza o processo de geração de código, apresentando ao usuário, no decorrer do desenvolvimento, a cada passo, os futuros resultados. O problema não está em automatizar o processo, mas em não relacionar efeitos com suas causas.

Mudança de alvo

A questão de mudança de alvo refere-se ao problema das ferramentas, que tem de lidar com o rápido desenvolvimento de novas tecnologias e novas técnicas de interface. Como a construção de ferramentas normalmente exige tempo de experiência e entendimento das tarefas que visa apoiar, pode acontecer que, quando a ferramenta atinge a maturidade para ser empregada satisfatoriamente, a tarefa apoiada ou mesmo os métodos empregados já não são importantes, ou mesmo obsoletos. Esse tema envolve fatores externos não controláveis e dificilmente previsíveis. Para suavizar o problema, *mCards* faz uso de uma linguagem abstrata para representar a interface, permitindo a inserção de tradutores específicos. Isso ao menos contorna a questão de mudanças tecnológicas.

Os temas acima propostos por Myers, não obstante o caráter subjetivo e preditivo de alguns, constituem importante fonte de referência para novas ferramentas, sendo, por esse fato, conforme descrito dentro de cada tema, considerado na realização de *mCards*.

3.5 Considerações Finais

O objetivo desse capítulo foi contextualizar a área de desenvolvimento de interfaces com o usuário. Primeiramente, mostrou-se a divisão do software atualmente existente na área de interfaces com usuário, desde o software de baixo nível como sistemas de janelas e *toolkits* até as ferramentas de alto nível. Sistemas de janelas e *toolkits* foram descritos sucintamente, enquanto maior atenção foi reservada para as ferramentas de alto nível, devido a sua relação direta com o trabalho.

No tópico de ferramentas de alto nível, mencionou-se a divisão das ferramentas de acordo com a fase de atuação, enfatizando a fase de desenvolvimento de interfaces. As outras fases não foram tratadas, devido esses temas fugirem do escopo do trabalho. Assim, omitiu-se detalhes de sistemas como UIMS (User Interface Management System) e conceitos aprofundados sobre avaliação de interfaces. Dentro das ferramentas de desenvolvimento, relatou-se diversas abordagens, desde as baseadas em linguagens até as de manipulação direta gráfica.

No tocante a esse trabalho, várias abordagens podem ser consideradas mais relevantes. Diretamente especificação gráfica por manipulação direta por ser a maior fonte de influência do trabalho desenvolvido, principalmente a subárea de *Interface Builders* em conjunto com a de *Cards*. Periféricamente, linguagens declarativas utilizadas na especificação da interface abstrata e *frameworks*, mais especificamente na parte de geração de código.

Na seção de avaliação de ferramentas, estabeleceram-se algumas métricas para avaliar a ferramenta proposta, de acordo com as funcionalidades e alguns temas encontrados na literatura.

Enfim, buscou-se neste capítulo fazer uma sucinta exposição do ferramental para desenvolvimento de interfaces com o usuário, historiando, classificando, e expondo o estado de arte da área. Mediante a apresentação efetuada, estabeleceu-se os meios para substanciar, contextualizar e avaliar o presente trabalho.

Trabalhos Relacionados

4.1 Considerações iniciais

Para uma melhor compreensão, os trabalhos relacionados foram divididos em: especificações de interfaces abstratas e ferramentas para construção de interfaces. Tal divisão é importante, pois as especificações são, em sua grande maioria, utilizadas pelas ferramentas, e abordando-as separadamente, auxilia-se o entendimento mais amplo das ferramentas.

4.2 Especificações de Interface abstratas

Nessa seção, algumas especificações abstratas baseadas no padrão XML serão descritas. Tratam-se de linguagens de alto nível cujo propósito é facilitar o desenvolvimento de interface, dispondo marcações representativas de elementos de interface de uma forma independente de dispositivo (Repo, 2004). A escolha por especificações que seguem o padrão XML deve-se a ampla aceitação dessas tecnologias, tanto no mercado corporativo como na área acadêmica. Com efeito, existem inúmeras especificações que se consagram ao tema de interfaces, de modo que apenas algumas foram relatadas.

A opção por XForms (Dubinko et al., 2001), XUL¹⁵ e XAML¹⁶ é motivada pelo potencial que se divisa para tais linguagens, no presente, e para o futuro. Outras linguagens se destacam, não sendo descritas nesse trabalho, apesar da importância que desempenham, por motivos de limite de tempo e espaço. Linguagens recentes como MXML¹⁷ (*Macromedia Flex Markup Language*) e XIML (Pucra & Eisenstein, 2002), ou mais antigas como UIML (Abrams et al, 1999) e UIDF são referências importantes.

4.2.1 XForms

Tradicionais formulários HTML frequentemente misturam os dados e a forma de apresentação. Com *XForms*¹⁸, recomendação para especificação de formulários W3C, o conteúdo é separado do layout, podendo, um formulário especificado com *XForms* ser apresentado por diversas plataformas. Adicionalmente dados podem ser pré-carregados, ou automaticamente preenchidos, bem como validados antes de serem enviados ao servidor por meio de um modelo de validação próprio.

XForms foi projetado para trabalhar conjuntamente com outras linguagens de marcação baseadas em XML. XHTML¹⁹ que corresponde a HTML em conformidade com XML,

¹⁵ XUL <http://www.xulplanet.com>

¹⁶ XAML <http://msdn.microsoft.com/longhorn/>

¹⁷ MXML : <http://www.macromedia.com/software/flex/>

¹⁸ XForms <http://www.w3.org/Markup/Forms/>

¹⁹ XHTML <http://www.w3.org/TR/xhtml1/>

SMIL²⁰ (*Synchronized Multimedia Integration Languages*) para integração e sincronização de mídias e SVG²¹ (*Scalable Vector Graphics*) para definição de gráficos vetoriais podem ser embutidos dentro do *XForms*. Portanto, *XForms* pode ser usado para variados tipos de aplicações.

Estrutura-se em duas seções: *Model* e *User Interface*, como mostrado na Figura 2. A primeira especifica o que o *form* faz, a segunda a aparência, sendo facultativo o uso da especificação de interface provida pelo *XForms*.

Figura 2 – Estrutura do XForms – dividido em camadas.
<http://www.w3.org/MarkUp/Forms/>

XForms Model representa o conteúdo do formulário que independe da forma de apresentação. O modelo divide-se em duas camadas: dados de instância e componentes lógicos. A primeira representa toda informação que é trocada com o servidor mais qualquer informação necessária dentro do modelo. Os componentes lógicos definem o comportamento da instância mediante controladores de eventos, *data bindings* e informações de submissão.

Apesar do modelo poder ser usado com padrões bem estabelecidos de apresentação, *XForms* especifica um próprio. Trata-se de uma especificação abstrata de interface que define dois tipos de controle. Um para a definição de *widgets* (controles atômicos), outro usado para organizar e agrupar *widgets* (controles de composição). Estes controles possibilitam definir como os dados de instância serão utilizados pela interface, podendo ser preenchidos, coletados, calculados, mapeados, ou seja, manipulados de uma forma geral.

²⁰ SMIL <http://www.w3.org/AudioVideo/>

²¹ SVG <http://www.w3.org/TR/SVG/>

Rememorando o capítulo anterior de *User Interface Software*, *XForms* se encaixaria com uma linguagem declarativa? Considerando apenas a camada de *user interface*, a resposta seria provavelmente. No entanto, *XForms* não modela apenas a interface, mas também os dados manipulados por essa interface, de uma forma separada. Assim, *XForms* não trata apenas de interfaces abstratas.

4.2.2 XUL

XUL²² (XML User-interface Language) consiste numa linguagem declarativa (vide seção 3.3.3.1) para especificar interfaces de usuário. Permite a criação de janelas e variados *widgets*, tais como botão, caixas-de-texto, barras de ferramentas, menus, barra de menus, *checkboxes*, caixas de diálogo, árvores e outros, fazendo uso de marcações. Esses componentes visuais possibilitam a construção de interfaces sofisticadas, em termos de recursos visuais e opções de manipulação, sem as complexidades inerentes de uma linguagem de programação tradicional.

Faz parte de uma família de tecnologias chamada *Gecko*, integrante da plataforma de desenvolvimento de aplicações do Mozilla²³, que, além do XUL, permite a utilização conjunta de vários padrões bem estabilizados como CSS²⁴, DOM²⁵, XSLT²⁶, XPATH²⁷, XHTML e outros para manipulação da interface. Também, dispõe a linguagem XBL (*Extensible Bindings Language*) para se criar *widgets* adicionais, além dos dispostos por XUL.

O objetivo do XUL, segundo seus criadores, é reduzir os custos no desenvolvimento de interfaces mediante uma especificação de interfaces multi-plataforma, de fácil utilização, que possibilite criar interfaces atrativas e bastante personalizáveis. Na questão de portabilidade, assemelha-se a linguagem Java, contudo ao invés da máquina virtual, utiliza a máquina do *browser* Mozilla. Assim como Java, XUL não se limita a aplicações *on-line*, podendo ser utilizada em aplicações *off-line*, sem a necessidade do *browser*. Contudo, essa solução implica a instalação do ambiente que interprete a especificação.

Com XUL, estrutura-se a interface com o usuário em três conjuntos de arquivos, cada um deles pertencentes a um subdiretório diferente. São eles: *Content*, *Skin* e *Locale*. No *Content*, especificam-se as janelas e os seus componentes visuais associados. Incluem-se também os *Scripts*, normalmente *JavaScript*, que controlam eventos e animações. Contudo, ficam em arquivos separados. Em *Skin*, colocam-se as folhas de estilo para ornamentar as janelas, e também as imagens utilizadas na interface. A colocação das folhas de estilo em um diretório próprio facilita a modificação da aparência da interface. O último diretório, *Locale*, serve para

²² XUL, <http://www.mozilla.org/projects/xul/>

²³ Mozilla Project, <http://www.mozilla.org>

²⁴ CSS www.w3.org/Style/CSS/

²⁵ DOM www.w3.org/DOM/

²⁶ XSLT www.w3.org/TR/xslt

²⁷ XPATH www.w3.org/TR/xpath

armazenar todo o texto utilizado dentro das janelas. Isso traz vantagens quanto à portabilidade do texto.

Diferente de *XForms*, que direciona-se a um tipo de aplicação bem específico, formulários na *Web*, apesar do potencial para ser usado em outros contextos, *XUL* é bem mais abrangente, ensejando a construção de interfaces requintadas, em diferentes contextos. O exemplo mais notório é o próprio navegador Mozilla, implementado em *XUL*. No entanto *XUL* não faz uma distinção entre dados e apresentação como *XForms*, concentrando-se essencialmente em proporcionar um conjunto amplo de opções.

Comparando-se as duas especificações, percebe-se que *XForms* leva vantagem no que faz, especificar formulários, natural por ser mais específico, enquanto *XUL* proporciona construir variados tipos de interface, com mais expedientes de interatividade e personalização, sendo uma peça em uma arquitetura de construção de aplicações. Ademais, enquanto *XForms* é uma recomendação oficial de um órgão aberto, *XUL*, embora seja baseada em padrões abertos, leva o nome da Mozilla, podendo, na questão de aplicações on-line, devido a motivos de concorrência, não ser executada em qualquer tipo de *browser*.

4.2.3 Xaml

Corresponde a uma linguagem declarativa para especificar interfaces para aplicações nos moldes de *XUL*, sendo também baseada em XML. Em termos de propósitos assemelha-se fortemente com *XUL*, permitindo especificar inúmeros controles visuais facilmente, com diversas opções de personalização. Os dois pertencem a uma plataforma de aplicações, integradas com diversas tecnologias, sendo que XAML (Extensible Application Markup Language) insere-se no *Longhorn*, sistema operacional em desenvolvimento da Microsoft, cujo lançamento está previsto para meados de 2006.

XAML faz parte do Avalon²⁸, camada do Longhorn responsável pela interface com o usuário. A Figura 3 ilustra a arquitetura do Longhorn. O Avalon é uma das atrações do *Longhorn*, mas, segundo a Microsoft poderá ser usado em outros ambientes. Um dos pontos fortes é a representação gráfica dos *widgets* como objetos vetoriais, como SVG (Ferraiolo et al., 2001), em vez de uma interface gráfica baseada numa matriz de pontos. O uso de gráficos vetoriais permite definições mais altas com possibilidade de ampliações irrestritas, embora exija muito poder de processamento, um requisito que talvez não seja problema em 2006.

O Avalon possibilita a divisão entre apresentação e lógica, dispondo a desenvolvedores o XAML para especificação da apresentação e diversas linguagens para codificação tradicional. Objetiva com XAML, especializar a tarefa de construir interfaces, facultando a designers, habituados a linguagens de marcação, exercer toda a sua criatividade nas interfaces dos programas sem se preocupar com a lógica de negócio ou a lógica dos programas. No entanto,

²⁸ Avalon <http://msdn.microsoft.com/longhorn/>

na arquitetura Avalon, XAML é uma opção, facultando ao desenvolvedor usá-la, podendo também utilizar apenas as linguagens convencionais disponíveis, ou, como entrevêem os projetistas, mesclá-las.

Figura 3 - Arquitetura do LongHorn. Adaptado de <http://www.fxconsult.com.br/xando/PDC2003/default.asp>

Em termos de implementação, cada elemento de marcação de XAML representa uma classe Avalon, sendo que arquivos XAML, contendo a especificação da interface, geram uma hierarquia de objetos que pode ser utilizado diretamente no Longhorn. Tal fato deve-se a possibilidade de compilação dos arquivos XAML em objetos executáveis, ao invés de interpretá-los em tempo de execução. No entanto, essa abordagem torna a tecnologia dependente da plataforma Microsoft. Uma das promessas em torno da construção de aplicações, utilizando XAML, é a possibilidade de se definir, mediante apenas um atributo, se a aplicação rodará em um navegador como um aplicativo do sistema operacional.

4.3 Ferramentas de construção de User Interface

Nesta seção, descrevem-se ferramentas que apóiam a construção de interface com o usuário, enfatizando, em cada uma delas, os pontos similares ao trabalho realizado. Entre os trabalhos relacionados, selecionou-se:

- Hypercard devido ao seu conceito de *cards* e tratamento de persistência.
- Ferramentas para construção de interfaces, baseadas em especificações de alto nível.
- Editores visuais devido a sua ampla utilização nos diversos meios.

4.3.1 Hypercard

Hypercard consiste num ambiente de programação gráfico concebido pela Apple para rodar em seu sistema operacional *MacOS*²⁹. Sua motivação era prover mecanismos flexíveis e trivialmente fáceis para o desenvolvimento de um tipo específico de aplicação. Esse tipo baseia-se no conceito de pilhas virtuais de *cards*, em que cada *card* pode incluir campos que armazenam dados, e também conter elementos visuais, denominados *background*. Entre os

²⁹ MacOS www.apple.com/macosx/

elementos, estão figuras, botões, textos, campos de texto, e outros elementos comuns de uma interface gráfica.

Usuários do Hypercard podem construir bases de dados mediante um editor gráfico que permite a manipulação de itens visuais, de modo que o controle das operações relacionadas ao armazenamento e recuperação dos dados fica a cargo do ambiente. Desse modo, uma aplicação para controle de endereços pode ser facilmente criada, fazendo uso apenas de alguns controles. Em tempo de execução, o usuário pode utilizar operações básicas como *search*, *add*, *delete*, proporcionando um conjunto de funcionalidades necessárias para a execução da aplicação. Essas funcionalidades são construídas, automaticamente pelo Hypercard, ensejando o rápido desenvolvimento de aplicações funcionais.

Para tornar o ambiente mais extensível, inclui-se no Hypercard uma linguagem de scripting, denominada HyperTalk. Tal linguagem tem uma sintaxe mais próxima de uma linguagem natural, facilitando o seu entendimento a usuários menos familiarizados com linguagens de programação tradicionais. Provê mecanismos para modificação de dados, bem como da interface.

Um dos recursos mais interessante do Hypercard é a capacidade que se tem para estabelecer ligações entre *cards* mediante elementos de ligação. Tais elementos funcionam como *links* entre contextos distintos propiciando a criação de interfaces navegáveis. O uso de *links*, mais o ambiente automático de armazenamento de dados que propicia a persistência de dados são o maior foco de interesse desse trabalho em relação ao *mCards*.

4.3.2 ARTStudio

Trata-se de um *framework* que tem o objetivo ajudar no desenvolvimento de *User Interfaces* para múltiplos dispositivos (Thevenin, 2002). Sua motivação é o fato que as ferramentas atuais, em sua grande maioria, dedicam-se, primeiramente, ao desenvolvimento de interfaces para um determinado alvo, relegando outros potenciais dispositivos, ao desenvolvimento *ad-hoc*. Pensando nisso, o objetivo do *framework* ARTStudio é gerar interfaces para múltiplos alvos, de uma forma consistente e automática.

Antes de caracterizar o *framework*, faz-se necessário introduzir o conceito de plasticidade. Este termo foi cunhado para se referir à questão de interfaces adaptáveis (Thevenin & Coutaz, 1999). O termo é inspirado na propriedade dos materiais plásticos que se expandem e contraem sobre condições naturais sem quebrar, preservando, assim seu continuado uso. Por analogia, plasticidade, no contexto de interfaces, consiste na capacidade da *interface* resistir a variações no que se refere características físicas de um ambiente, enquanto preserva sua usabilidade. Em outras palavras, uma *interface* plástica deve ser especificada uma vez e servir para diversos alvos.

Tecnicamente, plasticidade requer portabilidade, em que se escreve uma vez e utilizam-se múltiplas vezes (Calvary et al., 2001). Embora portabilidade seja realidade mediante o uso de máquinas abstratas como Java, tais mecanismos são limitados em oferecer configuração automática de *interface* em resposta a variações dos dispositivos de interação, gerando geralmente interfaces sem usabilidade para um determinado dispositivo.

Assim, um framework que permita a desenvolvedores especificar uma vez e produzir interfaces com usabilidade várias vezes é a base do que propõe ARTStudio, uma ferramenta para a concretização do conceito de plasticidade. Para isso, propõe um gerador de interfaces baseado num modelo de alto nível. A Figura 4 mostra a estrutura do ARTStudio.

Figura 4 – Percurso realizado por ArtStudio para gerar os resultados de um modelo de alto nível até a interface final (Thevenin, 2002)

No modelo de tarefas descreve-se como o usuário final, aquele que usará a interface alcança seus objetivos com o sistema. Nessa etapa, geralmente, utilizam-se análises realizadas por especialista em fatores humanos, a fim de otimizar os resultados. A partir desse modelo, um modelo conceitual é produzido com os conceitos do domínio que denotam as entidades que o usuário manipula em suas tarefas, e suas relações. Disso deriva-se uma interface abstrata que corresponde a uma expressão, representando a renderização abstrata dos conceitos e funções do domínio em conformidade com o modelo de tarefas.

A partir da interface abstrata, várias instâncias podem ser produzidas, cada uma objetivando um determinado dispositivo. Esse processo utiliza um conjunto de modelos baseados nos *widgets*, características físicas do dispositivo alvo e contexto de uso. O resultado do processo é

a geração da interface para um determinado alvo. ARTStudio, no momento gera código para Java/Swing e Waba³⁰ que corresponde a uma versão simplificada da máquina virtual Java, criada para ser rodada em dispositivos portáteis.

4.3.3 Teresa

Teresa (*Transformation Environment for Interactive Systems Representations*), assim como ArtStudio, é uma ferramenta de autoria para criação de *interfaces* para diversos dispositivos (Mori et al., 2003). Utiliza padrões XML, de modo a resolver alguns dos problemas associados à geração de interfaces nômades, conceito similar a interfaces plásticas. Entre os padrões utiliza XSL³¹ (eXtensible Stylesheet Language) para produzir apresentações em diferentes dispositivos e WML³² (Wireless Markup Language) para criar interfaces para um conjunto de dispositivos de tela pequena.

Tais linguagens facilitam na construção de interfaces nômades, mas são limitadas pelo fato que diferentes projetos de interação são necessários, quando se têm plataformas radicalmente distintas. Para resolver esse problema, além de padrões XML, Teresa utiliza a ideia de um modelo para muitas interfaces (Paterno & Santoro, 2002). Nessa abordagem inicia-se o desenvolvimento com uma descrição abstrata das atividades a serem cumpridas pela aplicação e posteriormente, após uma série de passos, obtém-se a interface pretendida.

A ferramenta, mediante uma série de transformações, divididas em quatro camadas, enseja a geração de interfaces para múltiplos dispositivos. A primeira camada trata-se de uma descrição abstrata das atividades, mediante uma notação hierárquica de tarefas a serem executadas pelo usuário final e os objetos manipulados para tal objetivo. É utilizada a notação ConcurTaskTrees (Mori & Santoro, 2002). Após a criação desse modelo de alto nível, na segunda camada, muda-se o foco para os objetos de interação necessários a execução das tarefas pretendidas. Para tal, produz-se uma especificação da interface compreendendo um conjunto de apresentações abstratas, cada um representado por elementos de interfaces. Cada apresentação abstrata é constituída por *interactors* que são objetos de interação abstratos identificados em termos de seu significado, ou seja, a tarefa básica que assistem.

Em seguida, vem a camada de concretização dos *interactors* em que são substituídos por objetos de interação concretos, de acordo com o tipo da plataforma. Tal camada permite a atribuição de diversos atributos relacionados à aparência e ao comportamento. Enfim, na última camada, a interface concreta é traduzida para uma determinada linguagem.

A ideia geral do Teresa assemelha-se muito ao ARTStudio, no que concerne a definição de um modelo de alto nível e, após uma série de transformações, gerar interfaces para múltiplos

³⁰ Waba waba.sourceforge.net/

³¹ XSL <http://www.w3.org/Style/XSL>

³² WML <http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html>

alvos. Contudo, Teresa permite diferentes pontos de entrada ao desenvolvedor tornando-o mais flexível. Assim, a ferramenta pode ser usada tanto para especificar em alto nível mediante o modelo de tarefas como também por meio da descrição lógica da interface propriamente dita. Tal característica possibilita diferentes níveis de auxílio e automatização, abrangendo desde soluções automáticas até soluções muito interativas em que o desenvolvedor tem a permissão de mudar os resultados gerados pela ferramenta.

No momento, a ferramenta direciona-se para interfaces Web, embora extensões para outros ambientes sejam permitidas, bastando modificar a última camada de transformação da interface concreta para a interface final.

4.3.4 Editor UIML

Trata-se de uma ferramenta de autoria para a construção de interfaces genéricas com tradução automática de código para diversas linguagens (Mayora-Ibarra et al, 2003). Utiliza um ambiente visual, com recursos *Drag and Drop* em que *widgets* associados a elementos UIML podem ser manipulados. UIML corresponde a uma especificação abstrata de interfaces, nos moldes do que foi descrito na seção 4.2. Assim o desenvolvedor, ao utilizar o ambiente visual manejando os componentes visuais, estará produzindo, de forma automática, uma descrição abstrata da interface representada em um artefato UIML. A partir dessa descrição, vários tradutores são empregados a fim de se criar interfaces para várias linguagens. O objetivo de tal abordagem é prover flexibilidade, facilidade, consistência e, principalmente, diminuir o tempo de desenvolvimento.

O ambiente visual proposto tem duas características principais. A primeira é o recurso *Drag and Drop* de *widgets* que ajuda, conforme já mencionado, a reduzir o tempo de desenvolvimento bem como prevenir erros de programação, pois o desenvolvedor não precisa codificar a interface, apenas desenhá-la. Outra vantagem é a familiaridade que tal abordagem propicia devido sua similaridade com outros ambientes visuais de programação, as chamadas IDEs (*Integrated Development Environment*). A Figura 5 ilustra o ambiente visual da ferramenta:

A segunda característica do ambiente visual consiste na utilização de um analisador léxico que tem a responsabilidade de verificar a validade da sintaxe do artefato que está sendo produzido e também auxiliar a tarefa dos tradutores na geração de interfaces. Após o processamento do analisador léxico, os tradutores encarregam-se de efetuar a conversão de UIML para a interface pretendida. Atualmente, a ferramenta dispõe de tradutores para formatos Web, como VoiceXML, um padrão para aplicações de voz na Internet e HTML/WML. Para tais linguagens, pelo fato de serem linguagens de marcação tal como UIML, utiliza-se XSLT (Extended Style Sheet Transformations). Outro tradutor disponível é para linguagem J2ME³³,

³³ Java 2 Platform Micro Edition. <http://www.java.sun.com/j2me>

usada para desenvolvimento de aplicações para PDAs. Pelo fato de não ser uma linguagem de marcação, não se utiliza a especificação XSLT, mas J2SE(Java 2 Standard Edition).

Figura 5 – Editor visual proporcionado pela ferramenta Editor UIML (Mayora-Ibarra et al, 2003)

A Figura 6 ilustra o funcionamento geral da ferramenta, descrito nos parágrafos acima, ilustrando desde o ambiente de edição visual até as interfaces finais:

Figura 6 – Esquema de funcionamento da ferramenta Editor UIML (Mayora-Ibarra et al, 2003)

Tal sistema, dos vistos até o momento, corresponde ao de maior similaridade com o trabalho realizado. O ambiente visual com recursos *Drag & Drop*, o uso de um documento XML representando a interface manipulada e os tradutores são, de fato, recriados por *mCards*. Diferente dos trabalhos anteriores, não existe tratamento para especificações de alto nível, em termos de modelo de tarefas e correlatos, mas, sim, dedicação direta ao processo de se construir visualmente a interface.

4.3.5 AUIT

Muitos sistemas de informação baseados na Web necessitam de interfaces adaptáveis, devido as diversas plataformas, tarefas e usuários presentes. Isto significa prover interfaces que rodarão em navegadores convencionais por meio de *Hypertext Markup Language* (HTML), bem como em PDAs, celulares e *paggers* usando *Wireless Markup Language* (WML). Também, faz-se necessário considerar o tipo de usuário e a tarefa apoiada. Por exemplo, mostrar o botão “Delete” ou “Update” de acordo com o tipo de usuário: cliente ou funcionário. Construir interfaces adaptáveis para sistemas de informação Web com as tecnologias atuais é custoso em termos de dificuldade, tempo e manutenção.

Diversas soluções utilizam *proxies* para converter automaticamente formatos. Tipicamente, essa abordagem produz interfaces de baixa qualidade, e geralmente são limitadas no grau de adaptação que permitem. Recentemente, surgiram propostas de descrições de interfaces genéricas, utilizando o padrão XML. Contudo essas linguagens geralmente não apoiam adaptação baseada nas tarefas dos usuários e são dirigidas a um tipo específico de aplicação. Nesse contexto, a arquitetura AUIT (*Adaptable User Interface Technology*) (Grundy & Zou, 2002) visa auxiliar desenvolvedores mediante uma linguagem genérica de interfaces que propicia a construção de interfaces adaptáveis. A solução é implementada conjuntamente com JSP ou ASP, funcionando como um serviço no servidor Web.

Para descrever interfaces no AUIT, desenvolvedores codificam a interface utilizando marcações XML que correspondem aos elementos visuais da interface (*labels, edit fields, radio buttons, check box, etc*). Adicionalmente, anotações podem ser feitas com informações a respeito dos usuários e tarefas a que se relacionam. Ademais, ações pré-definidas podem ser selecionadas para os elementos tal como: esconder, desabilitar e destacar. Documentos AUIT são implementados por Java Server Pages (JSPs) podendo ser mesclados com outros recursos Java como base de dados e sistemas legados mediante *JavaBeans*. Em tempo de execução, o documento é transformado em HTML ou WML, tendo a composição de elementos determinada pelo tipo de usuário e tarefa a ser realizada. A Figura 7 demonstra a arquitetura do AUIT.

Figura 7 - Arquitetura proposta por AUIT (Grundy & Zou, 2002)

Para facilitar o desenvolvimento de documentos AUIT, um ambiente visual foi desenvolvido no qual elementos visuais podem ser gerenciados graficamente por meio do mouse, gerando-se as marcações correspondentes de forma automática. A Figura 8 ilustra o ambiente visual disposto. Em (1) opções padrões de ferramentas visuais são disponibilizados, em (2) várias paletas podem ser selecionadas representando visões da interface: layout, estrutura lógica, artefato XML. Essas visões são apresentadas em (4). Em (3) são disponibilizados os elementos que podem ser inseridos na interface. Esses dividem-se em elementos visuais, elementos de navegação, scripts, JavaBeans. Esses elementos e também a estrutura lógica da interface podem ser atribuídos propriedades, conforme representado em (5).

Figura 8 - Ambiente GUI para a criação de artefatos AUIT (Grundy & Zou, 2002)

De interessante nesse trabalho em relação ao *mCards* é o uso de uma linguagem baseada em XML para representar a interface, bem como o uso de um editor visual *Drag & Drop* para a construção da interface. A possibilidade de descrever a interface tanto em termos visuais como em termos de uma linguagem proporciona flexibilidade no desenvolvimento de interfaces. Além de elementos visuais convencionais, a arquitetura permite a inserção de elementos navegacionais como *Links*, importante requisito de *mCards*. O alvo de AUIT é produzir interfaces adaptáveis para Web, sendo a definição em tempo de execução. *mCards* permite a construção de formulários para dispositivos móveis baseados em caneta em tempo de desenvolvimento. São abordagens diferentes, mas com idéias de automatização de interfaces semelhantes.

4.4 Outros Trabalhos

Muitos softwares comerciais utilizam a idéia de editores visuais para a construção de interfaces. Atualmente são comuns as IDEs (*Integrated Development Environment*) que integram linguagens de programação com editores de interfaces a fim de proporcionar um ambiente de desenvolvimento mais amigável. Nesses sistemas a interface pode ser desenhada, sendo o código da interface produzido automaticamente. Exemplos como *NetBeans*³⁴ e *JavaBuilder*³⁵ e o ambiente de desenvolvimento *Delphi*³⁶ são algumas das inúmeras aplicações que propiciam a construção visual da interface por mecanismos de manipulação direta.

Mais direcionados para o tema de produção de interfaces para múltiplos dispositivos, fazendo uso de uma linguagem abstrata, semelhantes aos trabalhos descritos na seção 4.3, menciona-se Liquid of Harmonia³⁷ que permite a manipulação de UIML, gerando interfaces para WML, Java, HTML e VoiceXML³⁸. Cita-se também *XPToolkit*³⁹ que proporciona um ambiente para trabalhar com elementos XUL. De fato, existem muitas outras ferramentas que tem o objetivo de apoiar a construção de interfaces especificadas por linguagens abstratas mediante ambientes visuais.

Em contraste com essas ferramentas que focam o desenvolvimento na aparência, outros centram o desenvolvimento de acordo com a natureza da tarefa que o usuário precisa realizar. Utilizam especificações de alto nível, como o modelo de tarefas, para representar a interação com o usuário, sem preocupação como o tipo de dispositivo. *Teresa* e *ARTStudio*, descritos nas seções 4.3.2 e 4.3.3 respectivamente, usam essa abordagem, mas existem outros trabalhos que seguem a mesma idéia. *Mobile (Model-Based Interface Layout Editor)* (Puerta et al, 1999), e *MOBI-D* (Eisenstein & Puerta, 2000) partes de um *framework* que faz uso de

³⁴ NetBeans <http://www.netbeans.org/>

³⁵ JavaBuilder <http://www.borland.com/jbuilder/>

³⁶ Delphi <http://www.borland.com/delphi/>

³⁷ Harmonia, inc. <http://www.harmonia.com/resources>

³⁸ VoiceXML Forum. <http://www.voicexml.org>

³⁹ XPToolkit Architecture <http://www.mozilla.org/xpfc/aom/AOM.html>

um conjunto de *guidelines*, algoritmos de adaptação, além do modelo de tarefas no processo de construção da *interface*, e Seescoa (Luyten & Coninx, 2001) direcionado para execução em tempo real para dispositivos portáteis são alguns exemplos desses trabalhos.

Outros trabalhos utilizam linguagens formais para representar a interface ou a interação com usuário que resultará na interface. UAN (*User Action Notation*) (Hartson et al., 1990) utiliza uma notação formal para representar a interação, permitindo a modelagem do projeto de interfaces de manipulação direta. Com UAN, interfaces são especificadas como uma hierarquia de tarefas assíncronas mediante o uso de símbolos sucintos. Embora trabalhos que utilizam esses formalismos não se relacionam diretamente a *mCards*, consistem em outro tratamento existente na literatura para a questão de se construir interfaces.

4.5 Considerações Finais

O capítulo apresentou trabalhos alusivos à questão de especificação e construção de interfaces. Caminhou-se desde linguagens abstratas para especificação da interface até ferramentas que automatizam o processo de desenvolvimento. Na parte relativa a ferramentas que apóiam a construção de interfaces, foram citadas ferramentas que utilizam especificações em alto nível bem como aquelas que se atentam diretamente para a aparência da interface. Nesses trabalhos, a ênfase maior está no desenho da interface por intermédio de recursos *Drag and Drop* de manipulação direta. Nos próximos parágrafos correlaciona-se esses diferentes tipos de trabalhos com o *mCards*, de modo a explicitar os requisitos e razões do projeto em questão.

Ao analisar as linguagens abstratas, alguns pontos puderam ser inferidos e observados. O uso de especificações abstratas como forma de representação torna-se imperativo atualmente, quando se almeja flexibilidade, em função da disseminação de formas de apresentação em termos da quantidade de dispositivos e meios de interação. O uso de XML para especificar a interface abstrata também é imprescindível, pois é amplamente aceito nos meios acadêmicos e empresariais, sendo de fato um padrão. Com base nisso, *mCards* faz uso de uma especificação abstrata, definida em XML, para descrever a interface a fim de que ele seja flexível e por consequência extensível.

Em relação aos trabalhos que fazem uso de especificações de alto nível, o problema, conforme foi discutido, na seção 3.3.3.3, é a imprevisibilidade quanto ao resultado. Para contornar esse problema, as ferramentas permitem o controle do que está sendo produzido, estabelecendo associações claras entre o modelo de tarefas e a interface resultante. Contudo, as razões que levam um determinado conjunto de tarefas a uma interface permanecem implícitas, o que torna essa abordagem nebulosa, embora interessante. É difícil, por exemplo, derivar objetivamente o *layout* baseado num modelo de tarefas, ainda mais quando se

consideram diversos dispositivos. Fatores como criatividade e subjetividade fazem parte da operação em questão.

Ferramentas como *Teresa* possuem um editor visual como complemento ao modelo de tarefas. Devido a dificuldade de se associar, de forma automática, especificações de alto nível com *widgets*, *mCards* não utiliza essa abordagem, embora considere-a como trabalho futuro, não como meio de se gerar automaticamente a interface, mas sim como um recurso de auxílio para a construção de interfaces.

A popularidade dos editores visuais para manipulação de *widgets*, como forma de produzir *interfaces*, deve-se amplamente a sua facilidade de uso. É natural pensar na construção de interfaces por meio da colocação de objetos visuais e manipulação destes. Além disso, o resultado faz parte do processo, pois o produto final é construído paulatinamente à medida que o usuário manipula os componentes visuais. Levando em conta principalmente a facilidade e a familiaridade desses sistemas, *mCards* baseia-se fortemente nessa abordagem.

MCards

5.1 Considerações Iniciais

Este capítulo descreve o *mCards*, ferramenta que tem por objetivo apoiar a construção de *Cards* para dispositivos baseados em caneta. Sua idéia e definição ocorreu gradualmente no decorrer do trabalho de pesquisa do aluno, sendo fruto da derivação de idéias concebidas anteriormente pelo aluno e orientadora. Inicialmente, a pesquisa abordaria anotações para dispositivos do tipo PDA, abrangendo temas relativos a privacidade de anotações e sua manipulação na tríade *antes, durante e depois* relativas a atividades de reunião. Embora o potencial relevante dessa abordagem, na execução prática, quando se iniciou a implementação de um sistema de anotações que validaria a inquirição realizada, vislumbrou-se uma modificação no trabalho de pesquisa que tornaria o trabalho mais abrangente.

Ao invés de dedicar-se a uma solução específica para um ambiente restrito como o de reunião, por intermédio de uma aplicação inteiramente especializada e focada, deslocou-se o objetivo central para a geração de aplicações de forma automática, baseada em uma ferramenta de autoria. Na primeira solução, o problema estava bem caracterizado num domínio bem definido; na outra o domínio é abrangente, apresentando, de fato, como principal característica a abrangência. Basicamente ocorreu uma generalização em relação ao previamente estabelecido, objetivando com isso uma maior contribuição. Naturalmente, devido a tal alteração, novos fatores emergiram tais como: formas de se especificar genericamente aplicações para dispositivos portáteis, nível de flexibilidade e tipo de aplicações.

O capítulo está estruturado da seguinte maneira. Na seção 5.2 as motivações do trabalho são apresentadas, havendo também uma breve contextualização da área em que o trabalho está inserido. A seção 5.3 narra um cenário envolvendo o trabalho a fim de torná-lo mais palpável no que se refere a sua visualização prática. Posteriormente na seção 5.4 é exposta a arquitetura do *mCards*, seus componentes e divisões. Após a parte descritiva, na seção 5.5, são expostos os resultados obtidos na pesquisa que culminou na criação de *mCards* e, por fim, na seção 5.6, são tecidas as considerações finais do capítulo.

5.2 Motivação

Uma das dificuldades encontradas atualmente por desenvolvedores de sistemas é a vasta quantidade de linguagens de programação, ambientes operacionais e dispositivos existentes. Nessa panacéia, com a velocidade com que a tecnologia se transforma, faz-se mister ferramentas que auxiliem o processo de desenvolvimento. Tempo é um recurso precioso, e otimizá-lo é sempre um objetivo, mesmo que muitas vezes seja implícito. Nesse ponto o uso

de expedientes que diminuam o tempo de desenvolvimento não apenas se faz necessário, mas trata-se de uma questão de sobrevivência.

O problema agrava-se quando se considera o desenvolvimento para novos dispositivos com mecanismos de interação diferentes do paradigma *Desktop*. Geralmente o desenvolvedor tem que se habituar a esse novo dispositivo, aprendendo novas linguagens e conceitos pertinentes. Esse procedimento consome tempo e, conforme discutido no parágrafo anterior, isso é, quase sempre, indesejável. Com base nessa realidade, a principal motivação para consecução do *mCards* é automatizar o processo de desenvolvimento para dispositivos baseados em caneta do tipo PDA, diminuindo o tempo despendido nessa tarefa. Evidentemente há motivações periféricas tais como contemporaneidade da área e a pouca quantidade de ferramentas de apoio.

Por ser um tema relativamente novo, devido ao surgimento recente de dispositivos baseados em caneta como os PDAs, tal área desperta um interesse crescente, tanto na comunidade científica como no mercado corporativo. Pelo mesmo motivo existe demanda por ferramentas especializadas que venham auxiliar os desenvolvedores na tarefa de construir aplicações, de modo que a novidade não seja um entrave ao desenvolvimento. O conhecimento acerca de um novo dispositivo, por exemplo, muitas vezes, torna-se dispensável para a produção de pequenas aplicações quando se utilizam ferramentas automatizantes que fazem uso técnicas de construção bem estabelecidas. A exploração de técnicas de manipulação direta com mecanismos *Drag and Drop* corresponde a uma das motivações desse trabalho, de aspecto relevante, embora não seja o motivo primordial para sua consecução.

Permitir construir aplicações simples de forma que não requeira conhecimentos específicos por parte do desenvolvedor é uma das metas do *mCards*. Isso potencializa mais usuários desenvolvedores, pois a ferramenta não condiciona sua utilização a conhecimento técnico especializado. Realmente, os recursos disponibilizados pela ferramenta exigem apenas um domínio básico de elementos de desenvolvimento. No entanto, o principal alvo do *mCards* são desenvolvedores de aplicações que possuem uma certa experiência com desenvolvimento de aplicações. Acredita-se que esses, devido a familiaridade maior com o processo de desenvolvimento, possam usar *mCards* com pouco treinamento. A curva de aprendizado para usuários especializados deve ser mínima, para que o foco seja no desenvolvimento ao invés do aprendizado da ferramenta.

5.3 Cenário de uso

Para melhor compreensão de *mCards*, a seguir é mostrado um diagrama de casos de uso sem detalhes aprofundados, mas com os principais requisitos sustentados pela ferramenta:

Figura 9 - Diagrama de casos de uso de *mCards*

A Figura 9 permite listar as principais atividades sustentadas por *mCards*, ilustradas em cada caso de uso de primeiro nível. Assim, tem-se o processo de especificação da interface, abrangendo a manipulação de *Cards* e *Widgets* em conjunto com parâmetros relacionados. Posteriormente, a etapa de geração de código e após, quando requerida, a integração com um aplicativo Web. Salienta-se, embora não demonstrado na figura, a dependência entre os casos de uso, sendo a integração dependente da geração e esta da especificação.

Embora o diagrama de casos de uso seja didático para mostrar o cenário que se insere *mCards*, ele não exterioriza muitos detalhes implícitos, e também não abrange contextos, motivações e circunstâncias. A fim de mostrar tais detalhes, tornando o cenário mais real, é narrada uma situação descritiva hipotética, mas factível, que permite situar a ferramenta em um contexto. Tal cenário não esgota as possibilidades da ferramenta, porém reporta a mais importante, sendo, de fato, a original. A seguir tem-se a narração:

Fernando, analista de sistemas, é proprietário de uma firma de desenvolvimento de sistemas voltado para Web. Ele desenvolve aplicações para pequenas firmas interessadas em dispor seu conteúdo na Internet, fazer vendas on/ e estabelecer comunicação com clientes. Trabalha principalmente com a criação de interfaces visuais, ou seja, o front-end das aplicações, mas, algumas vezes, é solicitado a gerenciar mecanismos de persistência. Utiliza basicamente tecnologias como HTML, JavaScript, Flash para o lado cliente e PHP, JSP e ASP para o lado servidor. Ultimamente, tem recebido solicitações de seus clientes para desenvolver as interfaces já existentes na Web também para dispositivos móveis.

No entanto, os clientes não querem apenas adaptar, mas principalmente aproveitar os meios de interação propiciados por esses dispositivos. Por exemplo, há demanda por usar entrada de dados mediante caneta em que o usuário pode estar efetuando traços a mão livre. Entretanto o recurso mais interessante é a possibilidade de persistência de dados no dispositivo, pois desobriga a necessidade de conexão permanente. Para atender a demanda dos clientes, Fernando provavelmente precisará adquirir novos conhecimentos, o que acarreta custos. Esses podem vir em função de cursos para aquisição das habilidades, ou, caso Fernando queira aprender sozinho, o custo será em termos de tempo despendido para o aprendizado. Não importa a forma de aquisição dos conhecimentos, pois invariavelmente haverá custos. Nesse panorama, a utilização de ferramentas que auxiliem o processo de criação de aplicações, e principalmente automatizem o processo de desenvolvimento é imprescindível para situações como a vivenciadas pela empresa de Fernando.

Em tal cenário, a ferramenta *mCards* propicia um ambiente visual para desenvolvimento de aplicações baseadas em *Cards* do tipo formulário, voltado para dispositivos baseados em caneta. Por meio do editor visual, em que *widgets* são disponibilizados para a manipulação e construção da interface, gera-se código de acordo com os *widgets* e parâmetros estabelecidos. Inserido nesse código tem-se:

- Interface com o usuário;
- Controle de persistência no dispositivo;
- Comunicação HTTP com repositório Web, sendo os dados encapsulados num padrão XML definido por *mCards*, porém personalizado pelo usuário.

Ao propiciar persistência de dados no dispositivo, o usuário pode entrar com dados a qualquer momento, podendo, posteriormente, enviar os dados, quando uma conexão estiver disponível. Isso impõe menos restrições ao usuário quanto a necessidade de conexão permanente, menos restrições quanto a infra-estrutura necessária, propiciando, assim, maior liberdade ao usuário e mais flexibilidade a aplicação.

5.4 Arquitetura

mCards, consoante entrevisto nas seções anteriores, busca auxiliar usuários desenvolvedores no desenvolvimento de aplicações baseadas em *Cards* para dispositivos baseados em caneta, como, por exemplo, PDAs. O uso de *Cards* permite separar contextos dentro da aplicação, tornando o processo de construção intuitivo, pois separa-se o desenvolvimento da aplicação em partes. Trata-se do famoso conceito de dividir para conquistar amplamente utilizado na programação convencional; no entanto, em *mCards*, tal abordagem acontece na forma visual. Para concretizar essa idéia, usa-se uma série de mecanismos que vão desde a ferramenta de autoria baseada em um editor visual, passando pela transformação da manipulação direta de componentes visuais em uma linguagem abstrata até o resultado final que corresponde ao código para uma determinada linguagem.

No que se refere à construção da interface visual da aplicação pretendida, *mCards* assemelha-se a muitas ferramentas encontradas no mercado de sistemas de computação cuja principal característica é o uso de mecanismos de manipulação direta. Tal abordagem trata-se de um padrão, senão oficial, pelo menos de fato, corroborado pelo amplo uso desse paradigma de construção de interfaces. O diferencial, em termos de criação de interfaces, em comparação a maioria dos sistemas, é a possibilidade da criação de contêineres de widgets, denominados *Cards*, com a possibilidade de estabelecimento de vínculos por intermédio de ligações. Para cada *Card* criado, há o controle de persistência, de acordo com os elementos inseridos, sendo tal funcionalidade transparente ao usuário desenvolvedor. Por exemplo, para cada *Card* a ferramenta gera uma barra navegacional para em tempo de execução, controlar as várias instâncias dentro de cada *Card*. A figura abaixo ilustra a idéia:

Figura 10 - Aplicativo típico gerado por *mCards*

A Figura 10 mostra o tipo de aplicativo que permite ser gerado, baseado em *Cards* e ligações. O usuário desenvolvedor encarrega-se de inserir *Cards* e *Widgets* bem como estabelecer ligações e a ferramenta controla o restante, gerando código apropriado. Essa abordagem propicia ao usuário desenvolvedor dedicar-se simplesmente a tarefa de montagem da aplicação, inserindo, dimensionando e movimentando visualmente componentes. Como questões relativas a codificação não são alvo de preocupação do usuário desenvolvedor, sua atenção é direcionada completamente para a tarefa de montagem proporcionada pelo editor visual, potencializando sua criatividade em tal tarefa.

Para isso, em conjunto com o editor visual, atua o módulo de geração de código, responsável por gerar o artefato executável. O código produzido espelha a manipulação ocorrida no editor, de modo transparente ao usuário, ou seja, este não precisa manusear o código. O propósito é o usuário somente desenvolver por meio do editor visual como se essa fosse, de fato, a linguagem de desenvolvimento. Além do módulo de geração de código e o editor visual, *mCards* possui um módulo cujo objetivo é ensejar a troca de mensagens entre a aplicação que está sendo produzida e uma determinada aplicação Web. Para tal fim,

estabelece um formato da mensagem a ser trocada e os meios necessários para recuperação desses dados no lado servidor.

A Figura 11 ilustra a arquitetura de *mCards*, composta por três partes: editor visual, geração de código e controle de mensagem. Na figura mostra-se o fluxo informações entre os módulos, sendo a seta tracejada o processo de criação da aplicação pretendida para o dispositivo baseado em caneta, e a seta sombreada o processo de controle da mensagem a ser trocada bem como a possível integração com determinado aplicativo Web. Nas subseções seguintes cada um dos módulos da ferramenta será explicada em seus pormenores, permitindo conhecer as características desses fluxos de informações.

Figura 11 - Arquitetura e fluxo de informações dos módulos de *mCards*

5.4.1 Editor Visual

Refere-se ao componente visual da ferramenta, em que são disponibilizados os recursos necessários a sua utilização. Possibilita a inserção de *widgets*, em uma tela virtual que representa o dispositivo pretendido, de modo que a posição e tamanho dos componentes, quando pertinentes, podem ser manipulados diretamente via *mouse* mediante o recurso *drag and drop* proporcionado por esse periférico de entrada. Igualmente, parâmetros alusivos ao tamanho da tela virtual, tipo do dispositivo e *URL* de comunicação podem ser manipulados.

O resultado do processo de manipulação do usuário no editor visual corresponde a um artefato XML que reflete os *widgets* inseridos, seus atributos, *Cards* associados e ligações quando existentes. Qualquer manipulação efetuada no editor visual produz em segundo plano um documento XML que segue um esquema bem definido, no caso fez-se uso do *XMLSchema*. A vantagem dessa abordagem é a abstrair a especificação da aplicação de um dispositivo alvo específico, ensejando a reutilização do resultado da manipulação visual. Permite, por exemplo, o uso de n transformadores que interpretam e transformam o documento XML para n dispositivos. Desvincula-se assim a construção da aplicação do dispositivo alvo, importante questão no cenário de dispositivos móveis que está em constante mudança atualmente. A Figura 12 ilustra o editor visual e seus componentes principais.

Figura 12 - Editor Visual de *mCards*

Na figura 12, em (a) tem-se a barra de componentes visuais que contém elementos amplamente difundidos em ferramentas de autoria como *Label*, *Text*, *Edit*, *ComboBox*, *List* e *Button*, e elementos específicos para outros ambientes como *Strokes* para dispositivos baseado em caneta e *Link* para o estabelecimento de ligações entre contextos distintos. O elemento de ligação é fundamental na aplicação alvo de *mCards*, baseado em *Cards*, porque permite a criação de vínculos entre contêineres de componentes distintos. *Ligações* também podem ser estabelecidas por outros elementos, tais como *Button*, *List* e *Combo*, havendo em tal ocorrência a agregação de funcionalidade para tal elemento.

Quando o usuário clica em algum dos componentes, tal elemento é inserido na área indicada em b) onde, mediante o mouse, pode-se arrastar e dimensionar o componente via botão direito e estabelecer atributos, quando existentes, por meio do botão esquerdo. Tal área permite antever o resultado no dispositivo, sendo um importante *feedback* ao usuário desenvolvendo. Em c) é mostrado o menu do editor visual, em que o usuário define parâmetros utilizados para a geração de código. Dispositivo alvo, código a ser gerado, tamanho da área de edição e URL para a comunicação são os parâmetros definíveis por meio desse menu. Implementou-se em *mCards* dois tipos de dispositivos (Palm e PocketPC), uma plataforma alvo (SuperWaba) e alguns tamanhos padrões de tela mais a possibilidade de personalização dessa. Vale salientar que a ferramenta, devido o uso de padrões abertos para representar a manipulação dos componentes visuais e parâmetros, permite a extensão no que tange a dispositivos e plataforma alvo, mediante a inclusão de transformadores.

A barra de ferramentas ilustrada por c) contém o atributo *store* que informa aos transformadores a necessidade ou não de se controlar a persistência, os dois formatos XML com seus respectivos esquemas, sendo o primeiro (*forML*) a representação das ações do usuário no editor visual cujo objetivo é ser processado pelos transformadores para geração de código e o segundo (*MessageML*) a mensagem a ser transmitida a um aplicativo Web. Logo abaixo, têm-se os botões representando o código gerado, no caso dessa versão de *mCards* há apenas *SuperWaba*. Por fim, em e) tem-se o botão de execução no qual a instância de *forML* é enviada aos transformadores para a geração de código.

O editor visual foi construído a fim de que o processo de criação de aplicações fosse mínimo em termos de tempo e dificuldade. Manejando apenas o mouse, exceto para digitar a URL de comunicação, o usuário é capaz de gerar aplicações funcionais, formulários baseados em *cards* com controle de persistência e passíveis de ligações para dispositivos baseado em caneta. O fato de o usuário precisar usar apenas o mouse evita o processo de especificar código. Livrando-o da codificação, deixando a preocupação quanto a aspectos tecnológicos à ferramenta, o usuário pode dedicar-se integralmente a montar visualmente a aplicação almejada, potencializando sua criatividade e comprometimento na tarefa.

Direcionar o desenvolvimento a aspectos mais relevantes, abstraindo os detalhes tecnológicos, vem acontecendo continuamente na área de desenvolvimento. Ao analisar a história da computação, percebe-se que a codificação de software em linguagem de máquina foi o começo para o surgimento das linguagens de montagem e posteriormente as linguagens alto nível. Essas, por sua vez, propiciaram o surgimento de ferramentas case, IDEs, frameworks sempre com a finalidade de otimizar o processo de desenvolvimento, tornando-o mais intuitivo as pessoas de desenvolvimento. De linguagens quase inteligíveis a ferramentas intuitivas altamente especializadas, num período de algumas décadas, foi um salto significativo que traz perspectivas quanto ao futuro próximo.

Seguindo a lógica do que vem ocorrendo, antevê-se a democratização do desenvolvimento no qual usuários não especialistas têm a disposição um conjunto de ferramentas intuitivas, transparentes quanto à utilização, mediante o uso de recursos mais naturais às pessoas. É o desenvolvimento ubíquo, mais acessível, mais natural. *mCards* não apresenta em sua essência tal propriedade, mas tem nas suas razões e motivações essa idéia, principalmente quanto ao aspecto de agente facilitador no desenvolvimento para dispositivos baseados em caneta genéricos.

5.4.2 Geração de código

O módulo de geração de código encarrega-se de transformar o artefato XML, denominado *forML*, gerado pela manipulação do usuário no editor visual, em código para um determinado dispositivo. Como os dados XML seguem um esquema pré-definido, a ferramenta utiliza recursos previamente programados a fim de facilitar a criação de tradutores que transformem instâncias de *forML* em código aplicável para determinado dispositivo. A Figura 13 delinea a abordagem do módulo.

Figura 13 – Estrutura do módulo de geração de código de *mCards*

Normalmente, a manipulação de dados XML, necessitaria de SAX ou DOM, APIs (*Application Program Interface*) amplamente utilizadas no tratamento de dados XML. Embora o uso consagrado dessas APIs, *MCards* faz uso de JAXB⁴⁰ para tratar o *forML*, devido seu relacionamento intrínseco com a plataforma Java que consiste na tecnologia empregada por *MCards* e seus módulos. JAXB permite que a partir de um esquema, no caso XMLSchema, gere-se classes Java correspondentes em que cada elemento torna-se uma classe e cada atributo uma variável de instância. Mediante o processo denominando *Unmarshal* permite-se a associação de dados XML, comumente por intermédio de um arquivo XML, com as classes geradas. A vantagem de JAXB, quando comparado a outras APIs, é que o acesso aos dados XML pode ser realizado instanciando classes e métodos representativos. Ao invés de se concentrar no aprendizado de como o *parser* funciona, há maior preocupação nos dados propriamente ditos. O funcionamento de JAXB e sua utilização no módulo de geração de código é mostrado na Figura 14.

⁴⁰ JAXB <http://java.sun.com/xml/jaxb/>

Figura 14 - Funcionamento de JAXB (adaptado de <http://www.sun.com/software/xml/developers/jaxb/index.xml>)

A figura acima permite enxergar que os tradutores não trabalham diretamente na especificação abstrata do aplicativo pretendido, no caso o documento XML, mas sim em objetos Java provenientes desse documento, gerados pelo processo de *Unmarshaling* já descrito no parágrafo anterior. Além do processo de *Unmarshaling*, a figura ilustra também a processo de *Marshaling* em que um objeto Java pode ser transformado num documento XML, processo inverso do utilizado por MCards.

5.4.3 Controle de Mensagem

O resultado da manipulação do editor visual não consiste apenas em código executável para um determinado dispositivo. Além de gerar aplicações do tipo formulário com tratamento de persistência, *mCards* provê também tratamento dos dados gerados em tempo de execução pelo aplicativo, estabelecendo um formato de mensagem, e meios de comunicação via http. Ou seja, os potenciais dados a serem produzidos pela aplicação em execução no dispositivo são considerados por *mCards* no momento de edição da aplicação. Ao fazer isso, o desenvolvedor da aplicação, ao utilizar a ferramenta, tem ciência das informações que podem ser enviadas pelo aplicativo que está produzindo. Nesse ponto, *mCards* diferencia-se de outras tecnologias similares por tratar não apenas a construção da aplicação, mas também os dados a ser produzidos por essa, ensejando a interação com aplicativos Web.

Bem como o formato utilizado para a geração de código, no caso forML, a representação da mensagem a ser enviada, denominada MessageML, também faz uso de XML devido a vantagens já mencionadas. Ademais, o uso desse padrão permite ao desenvolvedor integrar aplicações geradas por *mCards* com sistemas legados residentes em ambiente Web. Tal abordagem faz da ferramenta um gerador de aplicativos que podem funcionar como mecanismo de entrada para sistemas pré-existentes.

5.5 Resultados Obtidos

O resultado mais palpável desse trabalho é a ferramenta *mCards*. Contudo, mais importante que a aplicação são os passos que tornaram seu desenvolvimento possível, ou seja, o processo de inquirição efetuado. Em tal processo, devem ser considerados tanto os caminhos que não se consolidaram até àqueles que efetivamente foram concretizados. Conforme delineado nas considerações iniciais, a ferramenta foi concebida a partir de idéias anteriores que paulatinamente foram se transformando. De uma aplicação específica - sistema de anotações para PDAs - totalmente voltada para um cenário restrito, no caso ambiente de reunião, cujo principal objetivo era a ubiquidade, ampliou-se para um ambiente gerador de aplicativos em que a primeira pode ser uma instância.

No estudo de sistemas voltados para a plataforma de dispositivos móveis, relativos ao sistema de anotações, percebeu-se a grande variedade de tecnologias e plataformas envolvidas, uma panacéia de matizes tecnológicas com um mercado em grande expansão que demandava ferramentas apropriadas para desenvolvimento. Assim, optou-se por aplicar esforços na criação de um ambiente de desenvolvimento genérico na medida do possível, porém aplicável para uma determinada classe de aplicativos. Em tal escolha, pesou a pesquisa sobre sistemas baseados em *Cards*, efetuado na área de hipermídia que se mostra apropriado para dispositivos de tamanho reduzido como os PDAs. Mediante o uso de *Cards* e a possibilidade de ligações, aplicações podem ser divididas em *n* espaços diferentes, facilitando o desenvolvimento, além de não sobrecarregar a tela reduzida do equipamento, um dos fatores limitantes de tal tecnologia. Ao utilizar essa abordagem de construção, conjuntamente com um ambiente automatizado, potencializa-se a criatividade do usuário desenvolvendo na criação da interface com usuário à medida que sua preocupação volta-se completamente para tal tarefa. Detalhes de implementação e tecnologia ficam a cargo da ferramenta que se encarrega de garantir o funcionamento adequado.

No que se refere à computação ubíqua, objetivo do sistema de anotações pretendido anteriormente, dois pontos foram também considerados. Primeiramente, prover meios da aplicação resultante se tornar onipresente e ao mesmo tempo possibilitá-la comunicação. Para isso, fez-se uso de mecanismos de persistência, e implementação de módulo de comunicação http, de forma transparente ao usuário desenvolvendo. Com isso o usuário final tem a possibilidade de interação a qualquer momento, podendo armazenar informações e transmiti-la quando possível. Segundo, tornou-se o ambiente de fácil utilização, por meio de técnicas *Drag and Drop*, automatização, por meio da geração de código, almejando com isso tornar o desenvolvimento intuitivo.

A abordagem de *mCards* foi empregar o conceito de *Cards*, amplamente difundido na área de hipermídia, por motivos já mencionados, como mecanismo de desenvolvimento para dispositivos portáteis do tipo PDA, de uma forma intuitiva. Ou seja, almeja-se que o

processo de desenvolvimento assemelhe-se ou mesmo equipare-se, em termos de dificuldade, a atividade de esboçar a aplicação em uma folha de papel, em que o usuário desenvolve projeta sua aplicação, desenhando interfaces e estabelecendo caminhos de navegação.

5.6 Considerações Finais

O capítulo tratou de descrever a ferramenta *mCards*, expondo as motivações para sua concepção, a expansão da idéia inicial específica até sua generalização mais abrangente, considerando, sempre, os requisitos iniciais propostos, embora tenha sido necessário considerar outros mais. Depois, descreveu-se a arquitetura, apresentando seus módulos constituintes e dentro desses, as características e detalhes principais. Por fim, foi delineado os resultados obtidos, sintetizando a pesquisa efetuada em relação a *mCards* bem como tudo que envolveu o processo de sua obtenção.

Não obstante os resultados alcançados, a ferramenta tem muito a progredir com trabalhos futuros, pois não apresenta na totalidade os resultados pensados em sua concepção inicial. A questão da automatização e facilidade de uso precisa ser trabalhado por meio de experimentos com o usuário, o código gerado precisa ser otimizado para melhor performance, novos transformadores são necessários para maior abrangência da ferramenta bem como para testar o reuso da especificação abstrata *forML*. Ou seja, a ferramenta está em fase inicial, precisando de ajustes e principalmente de extensões.

Conclusão

6.1 Considerações Iniciais

Este capítulo discute os resultados obtidos pela pesquisa que teve como principal resultado a construção da ferramenta *mCards*. Primeiramente, os resultados alcançados são sumarizados, posteriormente, diversos trabalhos futuros são discutidos e, por fim, as considerações finais concluem trabalho realizado.

6.2 Resultados

Objetivamente, o resultado mais concreto do trabalho é a própria ferramenta, embora sua criação seja resultado de uma sucessão de idéias e investigações que foram evoluindo. Se inicialmente o objetivo era criar um sistema de anotações para dispositivos do tipo PDA de modo a permitir interação a qualquer momento, e chegou-se a uma ferramenta de criação de aplicações que contempla também a idéia pretendida inicialmente, então a série de investigações ocorridas foi válida.

Quando se analisa as motivações para este trabalho, a questão do desenvolvimento para tecnologias de PDAs, a necessidade de tempo de habituação com a tecnologia, a restrição existente no processo de desenvolvimento, e confronta-se tal realidade com as possibilidades oferecidas pela ferramenta *mCards* chega-se as seguintes conclusões:

- (1) O tempo de desenvolvimento é reduzido.
- (2) Diminui-se a necessidade de conhecimento específicos, no que alude o desenvolvimento de aplicativos para PDAs.
- (3) Possibilidade de construir aplicações que exploram vários tipos de interação

Para fundamentar a primeira asserção, embora experimentos precisem ser realizados para a confirmação, alega-se: (a) o fato do usuário programador não ter que especificar qualquer linha de código, podendo criar aplicativos apenas utilizando a interface visual do *mCards*. (b) o fato do usuário visualmente antever o aplicativo que se está produzindo, guiando a orientação do usuário quanto ao reconhecimento dos resultados a cada manipulação executada. Aceitando como verdade o item (1), a afirmação, contida em (2), torna-se evidente, pois, por não necessitar codificar, tendo a disposição modos visuais de estruturar o aplicativo, o usuário desenvolvedor não precisa de muito conhecimento específico para desenvolver aplicativos para PDAs.

Em relação à conclusão de que *mCards* facilita a criação de aplicações com vários tipos de interação argumenta-se: (a) controle de persistência tratado automaticamente pela ferramenta; (b) Possibilidade de inserção de componentes visuais como rótulos, caixas de edição, itens

baseado em lista ou em caixa, áreas de texto, espaço para entrada de traços a mão livre e ligações entre *Cards*, via a interface visual da ferramenta. Nessa os componentes podem ser alterados, dimensionados e movimentados por meio de manipulação direta, produzindo como resultado código pronto.

Em termos de usabilidade do *mCards*, apesar da natureza subjetiva do tema e da necessidade de experimentação, a ferramenta apóia-se no fato de que diversas ferramentas alcançaram expressivos resultados, quando fizeram uso de abordagens visuais de construção.

Ao contemplar os três itens elencados acima, acredita-se que a pesquisa, concretizada na ferramenta *mCards*, atinge, ainda que na forma de protótipo, os objetivos iniciais propostos de facilitar o desenvolvimento no contexto de dispositivos portáteis, tornando-o mais intuitivo, e, assim, minorando as restrições para construção de aplicativos.

6.3 Trabalhos futuros

No decorrer da investigação, entrementes a implementação e a escrita da dissertação, anteviram-se diversos trabalhos futuros. Alguns foram efetivamente realizados, uns delegados a pessoas do grupo de pesquisa e outros ainda estão para serem investigados. A seguir têm-se esses trabalhos futuros:

- Experimentações e testes com a ferramenta a fim de averiguar sua eficácia e utilidade
- Explorar a questão do desenvolvimento ubíquo, por meio de pesquisas que tratem a questão da produção de aplicações sem codificação direta, baseada em formas naturais de interação.
- Construção de tradutores que transformem a especificação abstrata *forML* – resultado da manipulação direta de componentes no editor visual - para linguagens declarativas de interface com o usuário como HTML, XFORMS, XAML, XUL, etc. Também tradutores que gerem a partir da especificação abstrata código para linguagens de programação voltadas para dispositivos portáteis como J2ME, PalmOS⁴¹ e também código para linguagens tradicionais que possuem interface de usuário sofisticada como Java, por exemplo.
- Estabelecer mecanismos que permitam auxiliar o programador na construção da aplicação, mediante modelos de alto nível. Permitir, por exemplo, a geração da interface visual a partir de uma especificação baseada em conjunto de tarefas, deixando a manipulação visual como forma de ajuste.

⁴¹ PalmOS C <http://www.palmsource.com/palmos/>

- Melhorar a percepção da aplicação que está sendo construída, fazendo-a executar na própria ferramenta. *mCards* realiza tal simulação, mas de forma parcial, apenas no sentido de navegação entre *Cards* por meio de ligações. Emular a persistência e a barra de navegação, permitindo a entrada de dados nos componentes inseridos, bem como sua manipulação, tornaria a percepção de execução do aplicativo que está sendo criado mais real.
- Alterar *mCards* para ser usado em ambiente Groupware, permitindo a criação de aplicações por mais de um usuário.

6.4 Considerações finais

Acredita-se que os objetivos propostos foram obtidos, embora questões relativas à validação com usuários finais estejam em aberto. O problema da dificuldade de desenvolvimento para novas tecnologias, tanto por questões de tempo como por falta transparência do processo foram bastante explorados, emergindo da inquirição a ferramenta *mCards*. Trata-se do resultado concreto, contudo, o principal, no ponto de vista do autor, são as idéias que o formaram.

Mais importante que o resultado representado pela ferramenta especificada e implementada, espera-se como fundamental contribuição o prosseguimento, por parte do grupo de pesquisa, das idéias presentes neste trabalho. Almeja-se que este trabalho seja apenas o início, uma semente para realizações vindouras.

Bibliografia

- (Abowd et al., 1997) Abowd, G. D., C.G. Atkeson, J. Hong, S. Long, R. Kooper, M. Pinkerton. 1997. Cyberguide: A mobile context-aware tour guide. *ACM Wireless Networks*, Vol. 3, pp. 421-433.
- (Abowd et al., 1998) Abowd, G.D.; Brotherton, J; Bhalodia, J. Classroom 2000: a system for capturing and accessing multimedia classroom experiences. *Proc. CHI '98*, mai. 1998.
- (Abowd, 1999) Abowd, G. D. Software engineering issues for ubiquitous computing. In *Proceedings of the 21st international conference on Software engineering*, pages 75-84, Los Angeles, California, United States. IEEE Computer Society Press.
- (Abowd & Mynatt, 2000) Abowd G.D.; Mynatt F. D., "Charting Past, Present and Future Research in Ubiquitous Computing", *ACM Transactions on Computer-Human Interaction*, 7(1):29-58, March 2000.
- (Abrams et al, 1999) Abrams M., Phanouriou C., Batongbacal A., Williams S., and Shuster J., "UIML: An Appliance-Independent XML User Interface Language," Proc. 8th WWW Conf., 1999, http://www.harmonia.com/resources/papers/www8_0599/index.htm.
- (Africano et al., 2004) Africano D. et al. Designing tangible interfaces for children's collaboration. Proceeding of CHI 04.
- (Buxton et al., 1983) Buxton, W., Lamb, M. R., Sherman, D., AND Smith, K. C. 1983. Towards a comprehensive user interface management system. *Comput.Graph.* 17,3, 35-42.
- (Calvary et al., 2001) Calvary G., Coutaz J. and Thevenin D., "A Unifying Reference Framework for the Development of Plastic User Interfaces," Proc. Eng. Human-Computer Interaction Conf., pp. 173-192, 2001.
- (Calvillo et al., 2003) Calvillo-Gómez EH, Leland N, Shaer O, Jacob RJK The TAC paradigm: a unified conceptual framework to represent tangible user interfaces. In: *Proceedings of the 1st Latin American conference on human computer interaction (CI.IHC 2003)*, Rio de Janeiro, Brazil, August 2003.
- (Cattelan, 2003) Cattelan, Renan Gonçalves, Andrade, Andréa Rodrigues, Penedo, Carlos Frederico Rocha, Pimentel, Maria da Graça Campos. iClass: um sistema para captura e acesso de sessões em ambiente educacional. *Revista Eletrônica de Iniciação Científica*. <http://www.sbc.org.br/rcic/edi> , v.III, n.III, p.10 - 28, 2003.
- (Dey & Abowd, 2000) Dey, A.K., Abowd, G.D., Towards a Better Understanding of Context and Context Awareness. In CHI 2000 Workshop on the What, Who, Where, and How of Context Awareness, April 2000.

(Dix et al., 1998) Dix, A., Finlay, J., Abowd, G., Beale, R., 2nd ed., c1998. Prentice Hall, 834p.

(Dubinko et al., 2001) Dubinko M., et al., "XForms 1.0," W3C working draft, August 2001.

(Eisenstein & Puerta, 2000) Eisenstein, J. and Puerta, A.: Adaptation in automated user-interface design, Proc. 2000 Conference on Intelligent User Interfaces, New Orleans, 9-12 January 2000, ACM Press, pp. 74-81.

(Elrod et al., 1998) Scott Elrod , Richard Bruce , Rich Gold , David Goldberg , Frank Halasz , William Janssen , David Lee , Kim McCall , Elin Pedersen , Ken Pier , John Tang , Brent Welch, Liveboard: a large interactive display supporting group meetings, presentations, and remote collaboration, Proceedings of the SIGCHI conference on Human factors in computing systems, p.599-607, May 03-07, 1992, Monterey, California, United States , CA, February 1998.

(Ferraiolo et al., 2001) Ferraiolo J. et al., "Scalable vector graphics (SVG) 1.0 specification," W3C recommendation, September 2001.

(Grundy & Zou, 2002) Grundy, J.C., Zou, W. : An architecture for building multi-device thin-client web user interfaces, Proc. 14th Conference on Advanced Information Systems Engineering, Toronto, Canada, May 29-31 2002, Lecture Notes in Computer Science.

(Harrison et al., 1998) Harrison, B.L., Fishkin, K., Gujar, A., Mochon, C., Want, R. The Design and Use of Squeezable Computers: An Exploration of Manipulative User Interfaces. *In Proceedings CHI' 98*, pp.17-24, May, 1998.

(Hartson et al., 1990) Hartson, H. Rex., Siochi, Antonio C., Hix, D. The UAN: a user-oriented representation for direct manipulation interface designs, ACM Transactions on Information Systems (TOIS), v.8 n.3, p.181-203, July 1990

(Hartson & Hix., 1989) Hartson, H. R. & Hix, D. Human-computer interface development: concepts and systems for its management. ACM Computing Surveys, New York, Mar, p. 5-92. 1989.

(Henderson, 1986) Henderson, D. A., Jr. 1986. The Trillium user interface design environment. In Human Factors in Computing Systems, Proceedings SIGCHI'86. ACM, New York, 221-227. ACM Trans. Computer-Human Interaction, Vol. 2,N.1, March 1995.

(Hix & Hartson, 1993) HIX, D.; HARTSON, H.R. Developing user interfaces: ensuring usability through product and process. John Wiley: New York, 1993.

(Mayora-Ibarra et al, 2003) Mayora-Ibarra, Oscar, la Paz-Arroyo, Oscar, Cambranes-Mart, Edgar, Fuentes-Penna, Alexandro "A visual programming environment for device independent generation of user interfaces, CIIHC 2003 Conference Proceedings, ACM Press, pp61-68, 2003.

(Ishii & Ullmer, 1997) Ishii, H., Ullmer, B. Tangible Bits: Toward Seamless Interfaces between People, Bits and Atoms, CHI 97 Conference Proceedings, ACM Press, pp234-241, March 1997.

(Kay, 1993) Kay, Alan. C. The early history of Smalltalk, ACM SIGPLAN Notices, v.28 n.3, p.69-95, March 1993

(Linton et al., 1989) Linton, M. A., Vlissides, J. M., and Calder, P. R. 1989. Composing user interfaces with InterViews. IEEE Comput. 22, 2 (Feb.), 8-22.

(Luyten & Coninx, 2001) Luyten, Kris., Coninx, Karin., An XML-Based Runtime User Interface Description Language for Mobile Computing Devices, Proceedings of the 8th International Workshop on Interactive Systems: Design, Specification, and Verification-Revised Papers, p.1-15, June 13-15, 2001

(Mori et al., 2003) Mori G., F. Paterno', and C. Santoro, "Tool Support for Designing Nomadic Applications," Proc. Conf. Intelligent User Interfaces (IUI '03), 2003.

(Mori & Santoro, 2002) Mori, G., Paterno' F. and Santoro, C. "CTTE: Support for Developing and Analysing Task Models for Interactive System Design," IEEE Trans. Software Eng., vol. 28, no. 8, pp. 797-813, Aug. 2002.

(Myers & Rosson, 1992) Myers, B. A. AND Rosson, M. B. 1992. Survey on user interface programming. In Human Factors in Computing Systems, Proceedings SIGCHI'92

(Monterey, CA., May). ACM, New York, 195-202.

(Myers et al., 2000) Myers B., Hudson S. and Pausch R., "Past, Present, Future of User Interface Tools," ACM Trans. Computer-Human Interaction, Vol. 7, N. 1, pp. 3-28, 2000.

(Myers, 1995) Myers, B. A. User interface software tools, ACM Transactions on Computer-Human Interaction (TOCHI), v.2 n.1, p.64-103, March 1995

(Nielsen, 1993) Nielsen, J. Usability Engineering. San Diego. CA: Academic Press Professional. Hardcover: ISBN 0-12-518405-0 Softcover: ISBN 0-12-518406-9

(Olsen, 1992) Olsen, D. *User Interface Management Systems: Models and Algorithms*. Morgan Kaufmann. San Mateo, CA 1992.

(Paley et al., 1995) Paley, A. J., Hansen, W., Kazar, M., Sherman, M., Wadlow, M., Neueundorffer, T., Stern, Z., Bader, M., and Peters, T. 1988. The Andrew toolkit—An overview, In Proc. Winter Usenix Technical Conference, Berkeley, Calif..

(Paterno & Santoro, 2002) Paterno' F. and Santoro C., "One Model, Many Interfaces," Proc. Fourth Int'l Conf. Computer-Aided Design of User Interfaces, pp. 143-154, 2002.

(Pedersen et al, 1993) Pedersen, E.; McCall, K.; Moran, T.P.; Halasz, F. (1993). Tivoli: an electronic whiteboard for informal workgroup meetings. *Proceedings Interchi 1993*. Amsterdam, The Netherlands, p. 391-398, apr 1993.

(Puerta & Eisenstein, 2002) Puerta A. and Eisenstein J., "XIML: A Common Representation for Interaction Data," Proc. ACM Conf. Intelligent User Interfaces (IUI '02), pp. 214-215, 2002.

(Puerta et al, 1999) Puerta A.R., E. Cheng, Ou, T., and Min, J. MOBILE: User-Centered Interface Building. CHI99: ACM Conference on Human Factors in Computing Systems. Pittsburgh, May 1999..

(Repo, 2004) Repo, Pertti Facilitating user interface adaptation to mobile devices. NordiCHI: Proceedings of the third Nordic conference on Human-computer interaction . Tampere, pp 433-436, 2004.

(Richter et al., 2001) Richter H., Geyer, W., Fuchs, L., Daijavad, S., Poltrok, S. "Integrating Meeting Capture Within a Collaborative Team Environment." In the Proceedings of the *International Conference on Ubiquitous Computing, Ubicomp 2001*, Atlanta, GA, September 2001.

(Schmucker, 1986) Schmucker, K. J. 1986. MacApp: An application framework. Byte 11, 8 (Aug.), 189-193.

(Streitz et al., 1994) Norbert A. Streitz , Jörg Geißler , Jörg M. Haake , Jeroen Hol, DOLPHIN: integrated meeting support across local and remote desktop environments and LiveBoards. In Proceedings of the 1994 ACM CSCW '94 Conference on Computer Supported Cooperative Work, p.345-358, October 22-26, 1994, Chapel Hill, North Carolina, United States.

(Sukaviriya et al., 1993) Sukaviriya, P., Foley, J. D., AND Griffith, T. 1993. A second generation user interface design environment: The model and the runtime architecture. In Human Factors in Computer Systems, Proceedings INTERCHI'93 (Amsterdam, Apr.). ACM, New York, 375-382.

(Szekely et al., 1993) Szekely, P., Luo, P., and Neches, R. 1993. Beyond interface builders: Model-based interface tools. In Proceedings INTERCHI'93 (Amsterdam, Apr.), 383-390.

(Thevenin & Coutaz, 1999) Thevenin D., Coutaz J., "Plasticity of User Interfaces: Framework and Research Agenda", Proc. of INTERACT'99 (Edinburgh, August 1999), IOS Press, 1999.

(Thevenin, 2002) Thevenin D., ARTStudio: Tool for Multi-target UI Design. Poster at UIST 2002, Paris.

(Thompson et al., 2001) Thompson et al., "XML schema part 1: Structures," W3C recommendation, May 2001.

(Truong et al., 1999) Khai N. Truong, Gregory D. Abowd , Jason A. Brotherton, Personalizing the capture of public experiences. In *Proceedings of the 12th annual ACM Symposium on User interface Software and Technology*, p.121-130, November 07-10, 1999.

(Vlissides & Linton 1990) Vlissides, J. M. and Linton, M. A. 1990. Unidraw: A framework for building domain-specific graphical editors. ACM Trans. Znf. Syst. 8, 3 (July), 204-236.

(Want et al., 1992) Want, R., Hopper, A., Falcao, V., and Gibbson, J. 1992. The active badge location system. *ACM Transaction on Information Systems*, 10(1):91-102.

(Want et al., 1993) Want R., Schilit B.N., Adams N., Gold R. and Tso M., The PARCTAB mobile computing system. In *Proceedings Fourth Workshop on Workstation Operating Systems (WWOS-IV)*, pages 34--39. IEEE, October 1993.

(Want et al., 1995) Want R., Schilit B.N., Adams N., Gold R., Petersen, K., Ellis, J., Goldberg, D. and Weiser, M. 1995. The PARCTab ubiquitous computing experiment. Xerox Palo Alto Research Center, Technical Report CSL-95-1, March.

(Weiser, 1991) Weiser, M., The computer for the 21st century. *Scientific American* vol. 265, no.3. pp. 94-104, September 1991.

(Weiser, 1993) Weiser, M., Some computer science in ubiquitous computing, *Communications of the ACM*, vol. 36 no. 7, pp. 75-84, july 1993.

(Weiser, 1994) Weiser, M., The world is not a desktop. *ACM Interactions*, vol. 1 no. 1, pp. 7-8, January 1994.

(Wilson, 1990) Wilson, D. 1990. *Programming with MacApp*. Addison-Wesley, Reading, Mass.