Um modelo de programação orientado ao desenvolvimento de sistemas ubíquos

Alex Sandro Garzão [1]
Lucian José Gonçales [1]
Jorge Luis Victória Barbosa [1]

Resumo: A tarefa de desenvolver aplicações ubíquas nos modelos tradicionais de programação torna-se um desafio, pois a maioria desses modelos baseia-se em premissas estáticas de arquitetura, dados, aplicação e sistemas operacionais. Por isso, o presente trabalho propõe o *Ubiquitous Oriented Programming* (UOP), um modelo de programação orientado ao desenvolvimento de sistemas ubíquos. O UOP utiliza os conceitos de serviços e da programação orientada a objetos, integrando-os com requisitos necessários em aplicações ubíquas como compartilhamento de informações contextuais, sensibilidade ao contexto, adaptação ao contexto, mobilidade de código e concorrência. O ambiente do UOP é composto por uma linguagem de programação (UbiL), um compilador (UbiC) e por uma máquina virtual (UbiVM) que suporta a execução das aplicações desenvolvidas em UbiL. O modelo foi avaliado de forma experimental, na qual uma aplicação foi criada e então simulada por meio de um cenário de comércio ubíquo, concluindo-se que o UOP facilitou o desenvolvimento dessa aplicação.

***Palavras-chave*:** Ambiente de desenvolvimento. Computação ubíqua. Modelo de programação.

Abstract: *The development of ubiquitous applications in traditionals programming models became a challenge because the majority of these models are based in static assumptions of architecture, datas, application and operating systems. Therefore this work proposes the Ubiquitous Oriented Programming (UOP in short), a programming model oriented to develop ubiquitous systems. The UOP model uses the services and object oriented programming concepts integrating them with required requisites in ubiquitous applications like contextual information sharing, context awareness, context adaptation, code mobility and concurrency. The UOP model is composed by a programming language (UbiL), a compiler (UbiC) and by a virtual machine (UbiVM) that supports the execution of applications developed in the UOP model. The model was assessed through an experiment where an application was created to execute in a ubiquitous commerce scenary. Throught this experiment was concluded that the UOP model has facilitated the development of this application.*

***Keywords*:** *Development environment. Programming model. Ubiquitous computing.*

1 Introdução

Segundo Weiser [1], na computação ubíqua, os dispositivos computacionais estariam tão integrados ao ambiente que não seriam percebidos pelos usuários. Porém, para que seja possível explorar esse modelo, as aplicações ubíquas deveriam utilizar os conceitos de sensibilidade ao contexto, adaptação ao contexto, mobilidade de código e concorrência [2].

O contexto mantém as informações utilizadas para caracterizar a situação de entidades (pessoas, lugares e objetos) que são consideradas relevantes para a interação entre o usuário e a aplicação [3]. No modelo computacional, denominado sensível ao contexto, as informações do contexto são combinadas com as preferências do usuário para adaptar o comportamento das aplicações. Sendo assim, sensibilidade ao contexto [3], [4], [5] refere-se à

[1]UNISINOS, Av. Unisinos, 950 - São leopoldo (RS) - Brasil
{alexgarzao@gmail.com, lucianjosegoncales@gmail.com, jbarbosa@unisinos.br}

http://dx.doi.org/10.5335/rbca.2014.3447

obtenção de informações relevantes por sensores (combinação de hardware e software) usados para medir valores contextuais.

Além disso, o relacionamento entre os interesses do usuário com a sua localização possibilita o que tem sido denominado de *Location Based Services* (Serviços Baseados em Localização – LBS [6], [7]), em que não é informada somente a localização, mas é oferecido um conjunto de serviços contextualizados [6].

A precisão dos atuais sistemas de localização permite a implementação de aplicações comerciais [8], [9], gerando, assim, novas oportunidades para a computação ubíqua em áreas como educação, comércio, redes sociais, medicina, jogos, entre outras.

Porém, a dificuldade ainda reside em como desenvolver aplicações que se adaptem continuamente ao ambiente, sem parar sua execução, mesmo com o usuário movendo-se ou trocando de dispositivo [10]. Essas aplicações são difíceis de projetar e implementar devido à necessidade de um comportamento dinamicamente adaptativo às condições ambientais em que executam [3].

As aplicações ubíquas necessitam de uma abstração entre os diferentes dispositivos e as aplicações dos usuários para que possam abstrair a complexidade do ambiente e isolar as aplicações da necessidade de gerenciar protocolos, memória distribuída e falhas de comunicação [11]. As atuais aplicações, principalmente aquelas que são desenvolvidas para o *Android* e iOS, usam linguagens de programação que não apresentam suporte nativo à ubiquidade, sendo necessário utilizar componentes para realizar tarefas como a sensibilidade e a adaptação ao contexto. Além disso, a falta de modelos conceituais e métodos, com ênfase na computação ubíqua, dificultam o uso das novas tecnologias no desenvolvimento de aplicações.

Nos modelos de programação tradicionais em uso nos sistemas distribuídos (paradigma imperativo, lógico, funcional e orientado a objetos), existem limitações que afetam a computação ubíqua [10]. Por exemplo, a abstração encapsula código e dados, mantendo-os dentro dos objetos, o que dificulta o compartilhamento desses dados. Além disso, os modelos geralmente são baseados em premissas estáticas de arquitetura, aplicação, dados e do sistema operacional. Os dados são usualmente armazenados em um formato que dificulta o seu compartilhamento e os recursos que serão utilizados devem ser conhecidos *a priori*. Como resultado, não é fácil criar aplicativos ubíquos utilizando apenas os modelos tradicionais.

Nesse contexto, está sendo proposto o UOP, um modelo de programação focado no desenvolvimento de aplicações ubíquas, que integra os conceitos de *Object-Oriented Programming* (OOP) com os conceitos de serviços, compartilhamento de contextos, sensibilidade ao contexto, adaptação ao contexto, mobilidade de código e concorrência. O ambiente UOP é composto por uma linguagem de programação (UbiL), um compilador (UbiC) e uma máquina virtual (UbiVM), que, juntos, auxiliam o desenvolvimento e a execução das aplicações.

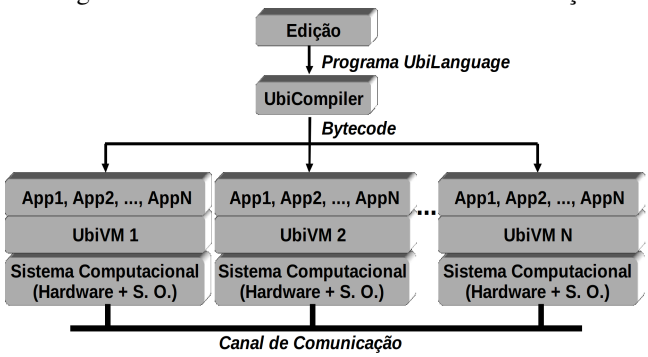
O artigo está organizado em seis seções, sendo esta a introdução a primeira. Na segunda discute-se o modelo UOP. A seção seguinte discute as características de implementação do UOP. A quarta apresenta um experimento que executa em um cenário simulado com a temática de comércio ubíquo e discute como o UOP auxiliou no desenvolvimento da aplicação ubíqua. A quinta seção destaca as contribuições do UOP, ressaltando suas diferenças em relação aos demais trabalhos pesquisados. Por fim, a sexta seção contém as considerações finais.

2 UOP

O UOP é um modelo de programação que introduz um conjunto de diretrizes para facilitar o desenvolvimento de aplicativos ubíquos. A Figura 1 apresenta o ambiente de desenvolvimento e execução de aplicações do UOP. O ambiente de desenvolvimento é composto pela UbiLanguage (UbiL) e pelo UbiCompiler (UbiC). A UbiL é a linguagem de programação que concretiza os conceitos propostos no UOP, enquanto que o UbiC é o compilador que traduz o código escrito em UbiL para *bytecode*. A plataforma de execução é composta pelo sistema computacional e pela UbiVM, a máquina virtual responsável por executar o *bytecode* gerado pelo UbiC. As UbiVMs interagem para gerenciar a distribuição dinâmica dos conteúdos e serviços disponibilizados nos contextos pelos aplicativos em execução.

O UOP integra as características da programação orientada a objeto (classes, objetos, propriedades, métodos, mensagens, sobrecarga de métodos, herança e polimorfismo) e do conceito de serviços (herdado dos *Web*

Figura 1: Ambiente de desenvolvimento e execução



Services [12]), com os seguintes conceitos da computação ubíqua: contextos [3], sensibilidade ao contexto [3], [4], [5], adaptação ao contexto [13], mobilidade forte de código [14], [15] e concorrência.

A `entidade` é a principal abstração do UOP, sendo similar ao conceito de classes. Uma entidade define quais estados (propriedades) ela é capaz de manter e o seu comportamento (métodos e serviços). O `Elemento` é a instância de uma entidade, e, apesar de similar ao conceito de objeto da programação orientada a objetos, provê serviços e adaptação ao contexto. Um programa em execução é composto de elementos que armazenam estados por meio de suas propriedades, relacionando-se com outros elementos através do contexto.

O `contexto` é qualquer informação que possa caracterizar a situação das entidades. Tipicamente é a localização, identidade e estado das pessoas, grupos e objetos físicos e computacionais [3]. O UOP propõe contextos privados e contextos públicos. Os `contextos privados`, que armazenam informações privadas de uma aplicação, são criados implicitamente na inicialização do aplicativo e nunca são destruídos. Por conterem informações privadas, apenas a aplicação que os criou acessa tais informações. Os `contextos públicos` são formados dinamicamente durante a execução das aplicações e armazenam as informações contextuais compartilhadas pelos elementos dessas aplicações. Por conterem informações públicas, qualquer elemento, de qualquer aplicação, pode acessá-las.

Um contexto público compartilha três informações contextuais: membros, conteúdos e serviços. `Membros` são os elementos que fazem parte do contexto público. Durante sua execução, um elemento pode ser membro de vários contextos simultaneamente, utilizando-os para publicar, bem como para identificar as informações contextuais. Os conteúdos e os serviços são compartilhados nos contextos pelos elementos. Os `conteúdos` referem-se às informações compartilhadas pelos elementos no contexto enquanto os `serviços` são análogos ao serviços providos por *web services*. Um elemento pode disponibilizar vários conteúdos ou serviços em vários contextos, e a decisão sobre o que publicar ou remover, e em quais contextos, é tomada dinamicamente durante a execução do elemento. Um conteúdo, após ser publicado, pode ser lido, alterado e removido por qualquer elemento. Dessa forma, tal conteúdo fica disponível no contexto até que algum elemento decida removê-lo ou até que o elemento que o publicou finalize sua execução. O mesmo ocorre com um serviço, ou seja, após ser publicado, fica disponível no contexto até que o elemento que o publicou decida removê-lo ou finalize sua execução.

O UOP também suporta a adaptação de contexto, característica que possibilita a alteração das aplicações de acordo com as informações dos contextos. No UOP, a adaptação ocorre em dois momentos: durante a execução dos eventos e durante a resolução de nomes.

A adaptação das aplicações pode ocorrer por eventos ou pela resolução de nomes. Os `eventos` utilizam dois componentes, o *publisher*, que gera eventos, e o *subscriber*, que reage aos eventos. Logo, o *subscriber* vincula uma ação a um determinado evento no *publisher* e esta somente é executada quando efetivamente ocorrer esse evento. Especificamente, os eventos expressam determinadas situações que podem ocorrer com o elemento, envolvendo os contextos públicos e privados. Exemplos disso são as alterações na localização do usuário e na temperatura do seu corpo, os quais seriam enviados para as aplicações que estão monitorando essas informações contextuais. Essas aplicações executariam ações (métodos) para tratamento dos eventos.

A `resolução de nomes` refere-se à atividade em que a UbiVM busca o nome da entidade ou do método que melhor se adapta ao contexto atual. Na UbiVM, tanto as entidades como os métodos podem ter múltiplas definições, sendo cada definição utilizada em um contexto específico. Por esse motivo, durante a execução da aplicação, a resolução de nomes identifica qual definição melhor se adapta ao contexto atual.

A mobilidade de código possibilita que um aplicativo mova-se para outro dispositivo durante sua execução. O UOP utiliza a mobilidade forte de código por meio da migração [14], [15].

Para suportar o desenvolvimento de aplicativos concorrentes, o UOP possibilita a execução de métodos concorrentes. Ao serem invocados, esses métodos criam o seu próprio fluxo de execução, concorrendo, assim, com o fluxo de execução principal com os fluxos dos outros métodos concorrentes.

Na subseção 2.1, é apresentada a UbiL, enquanto na subseção 2.2, é apresentada a arquitetura da UbiVM.

2.1 UbiL

A UbiL é uma linguagem de programação que suporta as abstrações propostas pelo UOP, reduzindo o *gap* semântico existente entre o domínio ubíquo e a implementação das aplicações, o que proporciona maior portabilidade e clareza do código.

O código abaixo apresenta um trecho da gramática da UbiL. Os programas são compostos pela descrição de suas entidades e cada entidade define o contexto a que se adapta, bem como suas propriedades, métodos e serviços. Além de herdar as estruturas de controle encontradas em linguagens tradicionais de orientação a objeto, a UbiL também possui construções específicas para serviços, adaptação e concorrência.

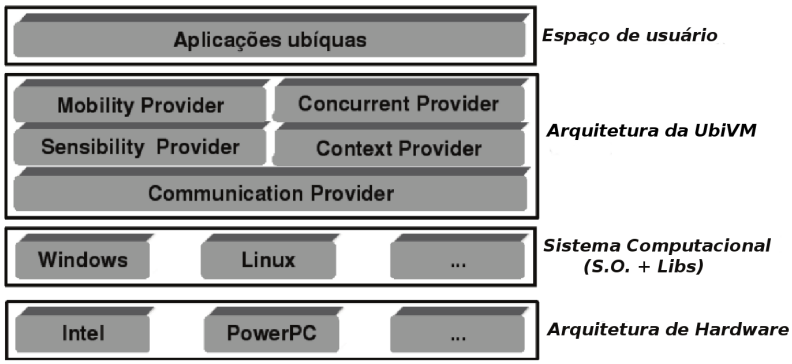
```
compilation_unit :
   (entity_def)+ EOF

entity_def :
   'entity' IDENTIFIER (valid_context)?
   (property_def)*
   (method_def|service_def)+ 'end'

property_def : ...

method_def :
  'method' IDENTIFIER '(' (params_def)?
  ')' (returns_def)?
  (var_def)*
  (statement)*
  'end'

service_def :
  'service' IDENTIFIER '(' (params_def)? ')' returns_def
  (var_def)*
  (statement)*
  'end'

params_def : ...

returns_def : ...

var_def : ...

statement : ...
```

2.2 UbiVM

A UbiVM é a máquina virtual responsável por executar os aplicativos ubíquos desenvolvidos usando a UbiL. Para que os aplicativos escritos em UbiL sejam independentes de hardware e sistema operacional, a UbiVM cria uma camada de abstração entre os programas e a máquina física em que está sendo executada, possibilitando que um *bytecode* gerado pelo UbiC execute em qualquer plataforma onde exista uma implementação da UbiVM.

A Figura 2 ilustra a visão da UbiVM, na qual constam as aplicações ubíquas, a arquitetura da UbiVM, o sistema computacional e arquitetura de hardware utilizada. A arquitetura da UbiVM é composta por cinco componentes:

Figura 2: Visão da UbiVM



- *Concurrent provider*: responsável por gerenciar a execução simultânea de vários fluxos, bem como a sincronização desses. A sincronia pode ocorrer de modo síncrono (por meio da troca de mensagens diretamente entre os fluxos) ou de modo assíncrono (indiretamente através do contexto). Sendo assim, os contextos tornam-se tanto um espaço para compartilhamento de informações contextuais, quanto um meio para realizar a sincronia entre fluxos de execução independentes;

- *Communication provider*: responsável por gerenciar a comunicação entre UbiVMs distintas por meio da troca de mensagens pela rede local, WiFi, *bluetooth* ou qualquer tecnologia disponível no momento para a qual a UbiVM tenha suporte;

- *Context provider*: responsável por compartilhar os conteúdos e serviços existentes nos contextos públicos, implementa uma arquitetura distribuída na qual cada aplicativo mantém registro apenas das informações disponibilizadas pelos seus elementos. Informações disponibilizadas por elementos de outras aplicações, quando necessárias, devem ser obtidas pelo *communication provider*;

- *Sensibility provider*: responsável por gerenciar os sensores externos, inclui o monitoramento dos dados recebidos desses sensores para popular os contextos privados e também de uma interface que modulariza o uso de vários tipos de sensores externos (GPS, sensor de batimentos cardíacos, sensor de pressão);

- *Mobility provider*: responsável por gerenciar a mobilidade forte de código [14], [15]. Nesse cenário, para que uma aplicação continue sua execução em outra UbiVM, o código, os dados e o estado dessa aplicação são transferidos.

3 Aspectos de implementação

Os protótipos do UbiC e da UbiVM possibilitaram o desenvolvimento e a execução de diversos exemplos e cenários que exploram conceitos da ubiquidade. Esses exemplos são constituídos por 387 arquivos de código fonte em UbiL, somando mais de 5.000 linhas de código, com a finalidade de realizar testes dos recursos da linguagem.

Os analisadores léxico, sintático e semântico foram gerados com o auxílio do gerador de *parsers* ANTLR [16]. A UbiL utiliza uma gramática LL(k) contendo 904 linhas e a UbiVM foi desenvolvida na linguagem C++ com aproximadamente 4 mil linhas de código. O UbiC, durante a conversão de UbiL para *bytecode*, gera o *opcode* de cada instrução, resolve endereçamentos e realiza a precedência de operadores. Por sua vez, a UbiVM é responsável por carregar e executar os *opcodes* existentes no *bytecode* gerado pelo UbiC. Para facilitar a sensibilidade ao contexto, a UbiVM utiliza sensores para obter dados pertinentes do ambiente e armazená-los nos contextos privados.

Os protótipos foram desenvolvidos e testados no Linux com o compilador GCC, versão 4.6.3. O projeto contém mais de 12.000 linhas de código fonte C++, distribuídas em 84 arquivos. A documentação do projeto foi disponibilizada pela da licença FDL, enquanto que seu código-fonte foi disponibilizado pela licença GPL. O código-fonte pode ser obtido em http://developer.berlios.de/projects/uop/.

4 Aspectos de avaliação

Para o experimento, foi definido um cenário de teste contendo sua descrição, possíveis situações envolvendo os participantes, e uma discussão com os resultados obtidos. O cenário é focado na área de exploração de negócios em ambientes ubíquos, denominada de comércio ubíquo [17]. No cenário do comércio apoiado pela computação ubíqua, novos pressupostos comerciais devem ser pensados, uma vez que produtos e serviços podem ser ofertados em qualquer lugar e a qualquer momento. O comércio, nesse cenário, é dinâmico e as oportunidades estão distribuídas em contextos. Por exemplo, baseado nos desejos de consumo do cliente, um sistema pode gerar intervenções do tipo: “foi encontrado um lojista neste local, que possui uma oferta para o seu desejo de consumo”. Como o objetivo é comprovar que o modelo do UOP facilitou o desenvolvimento desse cenário, a discussão é focada nos recursos específicos da computação ubíqua, ressaltando os trechos de código relevantes em UbiL.

Para a realização do experimento, foram utilizados dois desktops interligados por um roteador 10/100 de quatro portas. O desktop 1 é um *Pentium* IV 2.4, *dual core*, 2 GB RAM, 320 GB disco, placa de rede 10/100. O desktop 2 é um *Pentium* IV 1.4, 512 MB RAM, 160 GB disco, placa de rede 10/100.

4.1 Validação por meio de um cenário

Os cenários vêm sendo utilizados pela comunidade científica para validar aplicações sensíveis ao contexto (de acordo com Dey [18]) e ambientes ubíquos (de acordo com Satyanarayanam [19]). Seguindo essa estratégia, um cenário de comércio ubíquo foi criado para a avaliação do UOP, qual seja o “Relacionamento entre cliente e lojista no *shopping*”, na qual os lojistas publicam suas ofertas, enquanto o cliente cadastra o desejo por um determinado produto. Após visualizar as ofertas que atendem 100% do seu desejo, o cliente seleciona uma e realiza o pedido no seu smartphone, utilizando o serviço disponibilizado pelo lojista.

A seguir é descrito um cenário onde o cliente cadastra o interesse por um almoço no contexto praça de alimentação dentro do *shopping*:

Graciele está a caminho do *shopping center* para almoçar com sua família. Ela dispõe de R$ 15,00 e gostaria de comer um prato à base de massa. As ofertas do Restaurante da Olga e da Lancheria do Victório já foram publicadas. O restaurante possui cardápios diferenciados para almoço e janta, já a lancheria possui um único cardápio. Quando Graciele entra na praça de alimentação do *shopping*, a UbiVM em execução no seu smartphone verifica o “Repositório de ofertas” de todas as lojas (restaurantes e lancherias), disponível na praça de alimentação, tentando identificar pratos à base de massa que estejam dentro do valor máximo pretendido. Nesse instante, a UbiVM em execução no smartphone da Graciele emite um alerta, permitindo que ela visualize as opções. Após verificar as ofertas, Graciele decide por um prato de “massa á carbonara”, que custa R$ 13,00. Através do serviço de “Registro de pedido”, disponibilizado pelo Restaurante da Olga, Graciele recebe o número do seu pedido, e se encaminha ao caixa para realizar pagamento. Enquanto isso, seu pedido já está sendo preparado e em minutos será disponibilizado.

4.2 Discussão

Para a simulação desse ambiente, o desktop 1 foi utilizado como sendo o smartphone da cliente Graciele e o desktop 2 atuou como servidor do restaurante e da lancheria. Os seguintes aplicativos foram utilizados:

- Busca de ofertas: contendo 58 linhas de código-fonte, esse aplicativo foi executado no smartphone da Graciele com a linha de comando `ubivm busca_ofertas`;
- Ofertas do restaurante: contendo 55 linhas de código-fonte, foi executado no servidor do restaurante com a linha de comando `ubivm ofertas_restaurante`;
- Ofertas da lancheria: contendo 25 linhas de código-fonte, foi executado no servidor da lancheria com a linha de comando `ubivm ofertas_lancheria`.

A Tabela 1 apresenta os detalhes dessa execução, com uma sequência das ações de cada participante. Nesse cenário, o contexto `/shopping/praca_alimentacao` compartilha as informações sobre as ofertas dos lojistas da praça de alimentação. Cada oferta é publicada em um registro contendo uma chave composta pelo nome do lojista e o número da oferta, o que resulta no nome do prato, tipo (lanche ou refeição), base do prato, ingredientes, valor e o nome do serviço para realizar o pedido.

Tabela 1: Sequência das ações

Tempo	**Personagem**	**Ações**
1	Restaurante	Executa a aplicação “ofertas restaurante”. Cadastra a oferta “massa á carbonara” com suas características: - Lojista: Restaurante da Olga - Categoria: refeição - Componente base: massa - Ingredientes: ovos, creme de leite, bacon, queijo parmesão, macarrão. - Valor: R$ 13,00 - Serviço para realizar o pedido: `restaurante_checkout`
	Servidor do Restaurante	UbiVM publica a oferta no contexto `/shopping/praca_alimentacao`. Também publica o serviço `restaurante_checkout`, utilizado pelos clientes para realizarem seus pedidos.
2	Lancheria	Executa a aplicação “ofertas lancheria”. Cadastra a oferta “pizza presunto” com suas características: - Lojista: Lancheria do Victório - Categoria: lanche - Componente base: massa - Ingredientes: presunto, queijo, orégano - Valor: R$ 11,00 - Serviço para realizar o pedido: `lancheria_checkout`
	Servidor da Lancheria	UbiVM publica a oferta no contexto `/shopping/praca_alimentacao`. Também publica o serviço `lancheria_checkout`, utilizado pelos clientes para realizarem seus pedidos.
3	Graciele	Executa a aplicação “busca ofertas”. Cadastra o desejo “prato de massa” com as características: - Categoria: refeição - Componente base: massa - Valor: R$ 15,00
	Smartphone da Graciele	UbiVM verifica as ofertas de todas as lojas no contexto `/shopping/praca _alimentacao`. Ao identificar que uma das ofertas atende 100% das características do desejo de Graciele, gera um aviso para Graciele com a oportunidade identificada, disponibilizando o serviço `restaurante_checkout`.

4	Graciele	Visualiza a mensagem com a oferta do prato de “massa á carbonara” e solicita a realização do pedido.
	Smartphone da Graciele	UbiVM utiliza o serviço do restaurante para realizar o pedido e fica aguardando o recebimento do número do pedido.
5	Servidor do Restaurante	UbiVM executa a solicitação de serviço, vindo do smartphone da Graciele. Esse serviço executa uma ordem para a fila de preparo e de faturamento do pedido. Após isso, envia o número do pedido para o smartphone da Graciele.
6	Smartphone da Graciele	UbiVM recebe o número do pedido e exibe para Graciele. Esse aviso contém o número do pedido a ser informado no caixa, tanto para pagamento como para a retirada do prato.
7	Graciele	Verifica o número do pedido e dirige-se ao caixa. Seu prato já está na fila de preparo.

No tempo 1, as ofertas do restaurante são publicadas. Como o restaurante possui cardápios diferenciados para almoço e janta, o horário é quem determina as ofertas que serão publicadas. O trecho abaixo contém as duas versões da entidade `restaurante_offers`, responsável por publicar as ofertas. A primeira versão, na linha 1 do trecho abaixo, é adaptada para o horário do almoço, e a segunda, na linha 15, é adaptada para o horário da janta. Entre as linhas 4 e 10, são publicadas as características das ofertas no contexto `/shopping/praca_alimentacao`. Dentre as características, consta o nome do serviço que deve ser executado pelo cliente para realizar pedidos ao restaurante que, nesse exemplo, é `restaurante_checkout`.

```
1.  entity restaurante_offers when (time.hour >= 11 and time.hour <= 14)
2.     //Horario do almoco
3.      method constructor()
4.          {"/shopping/praca_alimentacao"}.
5.            cpublish(
6.             "Restaurante da Olga", "1" =>
7.             "massa a carbonara", "refeicao", "massa",
8.             "ovos, creme de leite, bacon, queijo parmesao, macarrao",
9.              13.00, "restaurante_checkout"
10.           );
11.         //...
12.      end
13.  end
14.
15. entity restaurante_offers when (time.hour >= 18 and time.hour <= 22)
16.      //Horario da janta
17.      method constructor()
18.               //..
19.      end
20. end
...  //..
40. {"/shopping/praca_alimentacao"}.spublish("restaurante_checkout");
```

No tempo 2, um trecho similar ao tempo 1 publica a oferta “pizza presunto” da lancheria. Na linha 7 do código abaixo, o serviço `lancheria_checkout` é disponibilizado no contexto para realizar os pedidos. As funcionalidades `cpublish` e `spublish`, nas linhas 2 e 7, publicam, respectivamente, os dados e serviços no contexto, resultando em um código dedicado à publicação de conteúdos e serviços.

```
1. {"/shopping/praca_alimentacao"}.
2.    cpublish(
```

```
3.     "Lancheria do Victorio", "1" => "pizza presunto",
4.     "lanche","massa", "presunto, queijo, oregano",
5.     11.00,"lancheria_checkout"
6.    );
7. {"/shopping/praca_alimentacao"}.spublish("lancheria_checkout");
```

No tempo 3, o trecho abaixo identifica, entre as ofertas disponíveis, quais satisfazem os requisitos definidos por Graciele (categoria, componente base e valor máximo). A variável `offer`, atribuída na linha 3, contém uma oferta do contexto. A condição `if`, situada entre as linhas 4 e 6, verifica se o produto pesquisado pelo usuário corresponde com a oferta e então, se ocorrer, a oferta será inclusa no menu da aplicação.

O UOP possibilita a execução de serviços dinamicamente, na qual o nome do serviço pode ser definido estaticamente ou descoberto durante a execução. O *array* `service_name_in_menu`, situado na linha 14 do código abaixo, recebe o nome do serviço que foi publicado no contexto junto com a oferta correspondente.

```
1. offer_list = {"/shopping/praca_alimentacao"}.clist();
2.  for(offer_number=1;offer_number<=offer_list.size();offer_number++)
3.       offer = offer_list[offer_number];
4.       if (offer.results[2] == identity.categoria and
5.           offer.results[3] == identity.componente_base and
6.           offer.results[5] <= identity.valor_maximo)
7.
8.           ncurses_menu.new_item(tools.pad(offer.keys[1], 20) +
9.                                 tools.pad(offer.keys[2],5) +
10.                                tools.pad(offer.results[1], 30) +
11.                                tools.ftoa(offer.results[5]));
12           count++;
13.          offer_code_in_menu[count] = offer.keys[2];
14.          service_name_in_menu[count] = offer.results[6];
15.      end
16. end
```

Na linha 2 do trecho abaixo, o serviço de *checkout* do restaurante é executado através de `srun`, o qual possui 3 parâmetros: o *array* `service_name_in_menu`, da linha 4, contém o nome do serviço obtido do contexto, o nome do cliente (`identity.name`) e o número da oferta, armazenada no *array* `offer_code_in_menu` da linha 6. Quando esse serviço é executado, o restaurante recebe o pedido de Graciele, correspondendo ao momento 5 da Tabela 1 e representado pela Figura 3. O serviço, ao ser processado no servidor da lancheria, retorna o número do pedido, que é exibido para o cliente (momento 6 da Tabela 1). A Figura 4 apresenta quando o aplicativo da Graciele recebe o número do pedido do restaurante para fazer a retirada do produto.

```
1. offer_number = ncurses_menu.show(5, 80, 11, 1);
2. identity.checkout_number = {"/shopping/praca_alimentacao"}.
3.     srun(
4.          service_name_in_menu[offer_number],
5.          identity.name,
6.          offer_code_in_menu[offer_number]
7.         );
8. ncurses.mvwrite(7, 19, identity.checkout_number);
```

O UOP é uma alternativa para facilitar a criação de aplicações ubíquas, oferecendo um ambiente de execução que suporta uma linguagem de programação orientada à ubiquidade. O UOP suportou, de forma especializada, o compartilhamento de conteúdos entre os aplicativos, a adaptação ao contexto, a sensibilidade ao contexto e a execução e descoberta de serviços, simplificando o desenvolvimento da aplicação. Entretanto, os trechos apresentados no cenário não abordaram a concorrência e a mobilidade de código. O *link* disponibilizado na seção 3 contém o

Figura 3: Aplicativo do restaurante recebe o pedido de Graciele

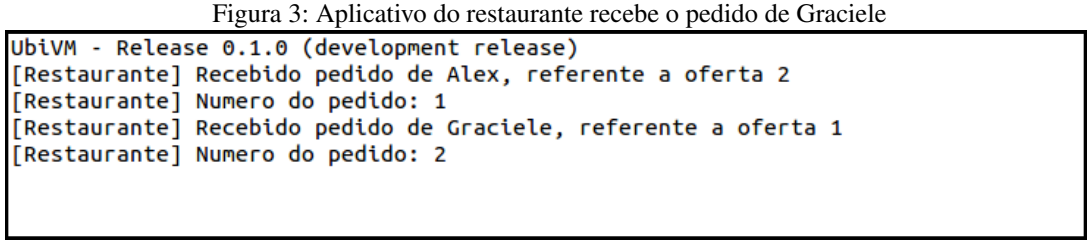
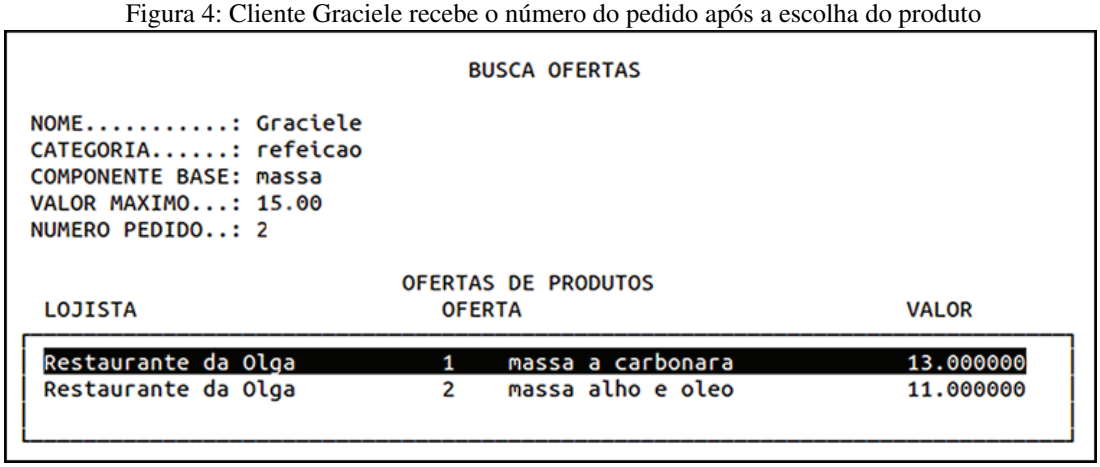


Figura 4: Cliente Graciele recebe o número do pedido após a escolha do produto



código fonte de outras aplicações que são focadas no cenário de educação ubíqua [20] e redes sociais ubíquas [21], além de exemplos abordando especificamente a mobilidade de código e concorrência.

5 Trabalhos relacionados

Os trabalhos relacionados são os modelos de programação e os ambientes de computação que permitem o desenvolvimento de sistemas usando os conceitos suportados pelo UOP.

O *Subject-Oriented Programming* estende os objetos e mensagens da programação orientada a objetos incorporando perspectivas, também conhecidas como *subjects* [22]. Um objeto pode ser visto de múltiplas perspectivas, isto é, um objeto pode responder diferentemente para mensagens enviadas de diferentes perspectivas. Métodos são selecionados não apenas se baseando no nome da mensagem e de seu receptor, mas também em quem está enviando. No entanto, esse modelo não tem suporte à sensibilidade ao contexto, mobilidade de código e concorrência.

Em *Context-Oriented Programming* [23], contextos expressam a ideia de um sistema de relações e assemelham-se às classes e objetos da programação orientada a objetos. Mensagens são enviadas não apenas se baseando no nome da mensagem, em quem está enviando e quem recebe, mas também se baseando no contexto atual da mensagem enviada. Contudo, esse modelo não suporta mobilidade de código e concorrência.

O Holoparadigma é um modelo de desenvolvimento para sistemas sensíveis ao contexto [24]. Uma nova entidade de programação (chamada ente) organiza níveis encapsulados de entes e histórias, e cria uma abstração de contexto. A história é um *blackboard* que armazena os dados compartilhados de um ente, porém, no Holoparadigma não existem abstrações dedicadas à adaptação ao contexto e ao suporte dinâmico de serviços contextualizados.

O *Context Toolkit* [18] é um framework para facilitar o desenvolvimento e a implantação de aplicações sensíveis ao contexto. É implementado como uma biblioteca de código para ser utilizada em outras linguagens.

No entanto, esse ambiente não suporta serviços, mobilidade de código e concorrência.

ISAM [25] integra os conceitos de sensibilidade ao contexto, *grid* e computação móvel. Diferentemente de outras propostas, ISAM foca no desenvolvimento de aplicações ao invés de focar no ambiente e serviços. Em função disso, o projeto define um modelo, uma linguagem e o suporte em tempo de execução para construir e executar aplicações pervasivas, mas essa proposta não suporta serviços definidos pelo usuário e não define uma máquina virtual própria.

O Continuum [26] é uma infraestrutura de software para a computação ubíqua, sensível ao contexto, baseada na arquitetura orientada a serviços (SOA), que facilita o desenvolvimento de aplicações ubíquas. A proposta integra *middleware* e frameworks e apesar de focar nos principais desafios apresentados pela computação ubíqua, não possui linguagem própria e uma plataforma que permita o desenvolvimento de aplicações ubíquas.

One.World [10] provê um framework integrado para o desenvolvimento de aplicações ubíquas. A arquitetura inclui um conjunto de serviços entre os quais se destaca serviços de descoberta, *check pointing*, migração e replicação. Além disso, é definido um modelo de programação no qual a distribuição deve ser explícita. No entanto, não suporta a execução de serviços definidos pelos desenvolvedores.

O Gaia [27] é uma infraestrutura de *middleware* distribuído que coordena serviços de software e dispositivos, contidos em um espaço físico, que se utiliza de uma rede heterogênea. Pode ser visto como um metassistema operacional que coordena entidades de software e dispositivos heterogêneos presentes em um espaço físico. O Gaia possibilita a consulta por serviços, utiliza-se dos recursos existentes para acessar e manipular o contexto atual e provê um framework para desenvolver aplicações centradas no usuário, consciente do espaço físico e para múltiplos dispositivos, entretanto não suporta serviços e mobilidade de código.

Aura [28] é um ambiente para a computação ubíqua que envolve comunicação sem fio, computadores portáteis que podem ser vestidos (*wearable*) e espaços inteligentes. O usuário tem uma "aura pessoal", que, ao entrar em um novo ambiente, encarrega-se de conseguir recursos necessários para execução da sua tarefa. Percebe-se, assim, que o sistema busca a próatividade (antecipa requisições) e o autoajuste (adaptação de uso de recursos e de desempenho). Contudo, nesse ambiente os serviços existentes, não podem ser ampliados, e a adaptação é apenas do ambiente em relação aos recursos disponíveis.

6 Considerações finais

A principal contribuição deste trabalho é a criação do UOP, um modelo orientado ao desenvolvimento de aplicativos ubíquos. A concretização do modelo por meio da definição da UbiL, do UbiC e da UbiVM possibilitou a realização de um experimento que indicou a viabilidade da proposta.

O UOP possibilita o compartilhamento dinâmico de conteúdos e serviços por meio dos contextos, da descoberta de contextos e serviços, do suporte a contextos públicos e privados, provê sensibilidade ao contexto por uso de sensores, permite adaptação ao contexto e mobilidade forte de código, bem como a exploração da concorrência nos aplicativos.

Todavia, o UOP constitui uma proposta inicial que necessita melhorias. A inclusão de contextos privados com memória persistente facilitaria o uso de trilhas [29], o que possibilitaria o registro do histórico das atividades realizadas por entidades em contextos, sendo possível então utilizar esses dados para o aperfeiçoamento da adaptação ao contexto. Além do gerenciamento de trilhas, pretende-se ampliar o estudo de modelos de contexto [30], permitindo o aperfeiçoamento do tratamento desse aspecto na UbiVM. Nesse sentido, além dos contextos públicos e privados, a definição de contextos compartilhados facilitaria o compartilhamento de informações e serviços somente entre um determinado grupo de usuários.

Referências

[1] WEISER, M. The computer for the 21st Century. *Scientific America*, v. 265, p. 94-104, 1991.

[2] COSTA, C. A.; YAMIN, A. C.; GEYER, C. F. Toward a general software infrastructure for ubiquitous computing. *IEEE Pervasive Computing*, v. 7, p. 64-73, 2008.

[3] DEY, A. Understanding and using context. *Personal and ubiquitous computing*, London, v. 5, p. 4-7, 2001.

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

[5] HOAREAU, C.; SATOH, I. Modeling and processing information for context-aware computing: a survey. *New Generation Computing*, v. 27, p. 177-196, 2009.

[6] VAUGHAN-NICHOLS, S. J. Will mobile computing's future be location, location, location?. *Computer IEEE Press*, v. 42, p. 14-17, 2009.

[7] DEY, A. et al. Location-Based Services. *IEEE Pervasive Computing*, vol. 9, p. 11-12, 2010.

[8] U-BLOX site. Disponível em: <http://www.u-blox.com>. Acesso em: 13 dezembro 2013.

[9] UBISENSE site. Disponível em: <http://www.ubisense.net>. Acesso em: 13 dezembro 2013.

[10] GRIMM, R. One.world: experiences with a pervasive computing architecture. *IEEE Pervasive Computing*, v. 3, p. 22-30, 2004.

[11] SAHA, D.; MUKHERJEE, A. Pervasive computing: a paradigm for the 21st century. *Computer*, v. 36, p. 25-31, 2003.

[12] AUSTIN D. et al. Web Services Architecture Requirements. 2004.
Disponível em: <http://www.w3.org/TR/wsa-reqs/>. Acesso em: 13 dez. 2013.

[13] LOPES, J. L. B. et al. Managing adaptation in ubicomp. In: XXXVIII CONFERÊNCIA LATINOAMERICANA DE INFORMÁTICA (CLEI), 2012, Medelin. *Proceedings...* New York: IEEE Press, 2012. p. 1-8.

[14] FUGGETTA, A.; PICCO, G. P.; VIGNA, G. Understanding code mobility. *IEEE Transactions on Software Engineering*, v. 24, p.342-361, 1998.

[15] NASEEN, M. K.; IQBAL, S.; RASHID, K. Implementing strong code mobility. *Information Technology Journal*, p. 188-191. 2004.

[16] ANTLR. Disponível em: <http://www.antlr.org>. Acesso em: 13 dez. 2013.

[17] FRANCO, L. K. et al MUCS: a model for ubiquitous commerce support. *Electronic Commerce Research and Applications*, Amsterdam, v. 10, p. 237-246, 2011.

[18] DEY, A. K.; SALBER, D.; ABOWD, G. D. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware application. *Human-Computer Interaction*, Hillsdale, v. 16, p. 97-166, 2001.

[19] SATYANARAYANAN, M. *Pervasive computing:* vision and challenges. *IEEE Personal Communications*, v. 8, p. 10-17, 2001.

[20] BARBOSA, J. L. V. et al. A ubiquitous learning model focused on learner integration. *International Journal of Learning Technology*, v. 6, p. 62-83, 2011.

[21] ZAUPA, D. et al. Implementing a spontaneous social network for managing ubiquitous interactions. In: XIII Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD-SCC), 2012, Petrópolis. *Anais...* New York: IEEE Press, 2012. p. 163-170.

[22] HARRISON, W.; OSSHER, H. Subject-oriented programming: a critique of pure objects. In: PROCEEDINGS OF THE EIGHTH ANNUAL CONFERENCE ON OBJECT-ORIENTED PROGRAMMING SYSTEMS, LANGUAGES AND APPLICATIONS, OOSPLA '93, 1993, Washington, USA. *Proceedings...* Washington: ACM, 1993, p.411-428.

[23] HIRSCHFELD, R.; COSTANZA, P.; NIERSTRASZ, O. Context-oriented programming. *Journal of Object Technology*, v. 7, p. 125-151, 2008.

[24] BARBOSA J. L. V. et al. Towards a programming model for context-aware applications. *Computer Languages, Systems & Structures*, v. 38, p. 199-213, 2012.

[25] AUGUSTIN, I. Isam, joining context-awareness and mobility to building pervasive applications. *Mobile Computing Handbook*. Auerbach Publications, p.73-94, 2004.

[26] COSTA, C. A. da. *Software infrastructure for ubiquitous computing: a context-aware service-based approach*. Saarbrücken: VDM Verlag, 2009.

[27] ROMÁN, M. et al. Gaia: A middleware infrastructure to enable active spaces. *IEEE Pervasive Computing*, v. 1, p. 74-83, 2002.

[28] GARLAN, D. et al. Project aura: toward distraction-free pervasive computing. *IEEE Pervasive Computing*, v. 1, p. 22-31, 2002.

[29] SILVA, J. M. et al. Content distribution in trail-aware environments. *Journal of the Brazilian Computer Society*, v. 16, p.163-176, 2010.

[30] KNAPPMEYER, M. et al. Survey of context provisioning middleware. *Communications Surveys Tutorials*, v. 15, p. 1492-1519, 2013.