Uma Plataforma de Testes para Aplicações Sensíveis a Contexto

Caroline Rizzi Raymundo[1] **e Patrícia Dockhorn Costa**[1]

[1]Departamento de Informática – Universidade Federal do Espírito Santo (UFES)
Av. Fernando Ferrari 514 – Vitória – ES – Brasil

carol.rizziray@gmail.com, pdcosta@inf.ufes.br

Abstract. *This paper presents a tool, coined CraftContext, capable of leveraging the test phase of context-aware application development. CraftContext offers a 3D environment, which is rich in details and resources, whose access is made by a robust and portable API. Its advantage over the main existing testing tools is its adaptability to different domains.*

Resumo. *Este artigo apresenta uma ferramenta, denominada CraftContext, capaz de auxiliar a fase de testes no desenvolvimento de aplicações sensíveis a contexto. CraftContext oferece um ambiente 3D de simulação de contexto rico em detalhes e variedade de recursos, cujo acesso ocorre por intermédio de uma API robusta e portável. Sua vantagem em relação às principais ferramentas existentes é a sua adaptabilidade a domínios diversos.*

1. Introdução

A crescente popularização dos dispositivos móveis impulsionou significativamente a busca por técnicas de desenvolvimento de aplicações sensíveis a contexto (CA - *Context Aware*). Tais aplicações são capazes de basear-se no contexto de entidades relevantes para o usuário a fim de fornecer autonomamente informações e/ou serviços de interesse deste [Dey 2001]. Um exemplo simples seria uma aplicação que controla o aparelho de ar condicionado de uma casa com base na localização do usuário (e.g., se ele estiver se aproximando da casa, o aparelho é ligado). A aplicação, neste cenário, deve analisar o contexto do usuário e, sem que este precise intervir, controlar diretamente o aparelho.

Para capturar dados de contexto essas aplicações geralmente utilizam-se de sensores, como aqueles presentes nos dispositivos móveis modernos (GPS, acelerômetro, luminosidade, etc.). Como consequência, um problema comumente enfrentado por desenvolvedores de aplicações CA é a fase de testes. Utilizar fontes de contexto físicas (ou seja, sensores) de forma controlada e reproduzível é uma tarefa trabalhosa e muitas vezes impraticável [Shah et al. 2010]. Por exemplo, considere uma aplicação que monitora a velocidade de um veículo e o ambiente ao seu redor, detectando, por intermédio de mapas, situações de risco (como curvas acentuadas ou trechos de tráfego intenso de pedestres). Ao detectar que o veículo está em alta velocidade numa área de risco eminente, a aplicação envia um alerta ao motorista. Para testar tal aplicação através do uso de sensores reais o engenheiro de teste precisaria colocar-se em uma situação de risco. Uma das formas encontradas para contornar esse problema é simular virtualmente o contexto do usuário.

Ferramentas de simulação de contexto, como em [Bylund and Espinoza 2002], [Broens and van Halteren 2006] e [Martin and Nurmi 2006], têm por objetivo gerar os dados de contexto e exercer o papel dos sensores, enviando os dados gerados para a aplicação CA. Essas ferramentas costumam fornecer um ambiente que simula o mundo real e dependem da existência e interação de agentes para, com base em suas ações no mundo virtual, gerar contexto. Usualmente de propósito geral, esses simuladores costumam implementar apenas os sensores mais comuns, facilmente encontrados em *smartphones* e outros dispositivos móveis, como posição, velocidade, temperatura, luminosidade, etc. Essa limitação se torna perceptível e problemática quando se faz necessário o teste de aplicações CA de um domínio muito específico, cujas necessidades vão além dos recursos dos sensores mais comuns, citados anteriormente. Alguns exemplos de sensores pouco usuais são os de sinais vitais (batimentos cardíacos, ritmo respiratório, etc.), de porta aberta, de peso, de despressurização, etc.

O presente trabalho propõe uma ferramenta de simulação de contexto que oferece um extenso conjunto de sensores simulados. Tal ferramenta, denominada *CraftContext*[1], utiliza o jogo *Minecraft* [Mojang 2012a] para simular o mundo real e gerar dados de contexto. Graças à grande variedade de eventos e recursos existentes em *Minecraft*, diversas situações e cenários do mundo real podem ser simulados ou mapeados para alguma funcionalidade particular do jogo. Por exemplo, um sensor interno a uma máquina de refrigerantes com o objetivo de alertar quando esta estiver vazia (um exemplo de domínio bem específico) poderia ser simulado no jogo através de um baú. Quando sem itens, este baú ativa um evento informando sobre sua atual condição: vazio. A ideia é que o desenvolvedor da aplicação CA possa usar de forma criativa a extensa gama de possibilidades do jogo para simular quaisquer sensores que ele precise nos testes de sua aplicação.

Além de oferecer apoio à detecção de eventos de contexto, *CraftContext* ainda possibilita à aplicação CA cliente o acesso à chamada de funções para alterar o mundo de *Minecraft*, recuperar informações sobre este ou mesmo enviar mensagens aos jogadores. Os comandos e requisições enviados diretamente pelos jogadores também podem ser detectados.

Na próxima seção são analisados alguns trabalhos que buscam oferecer soluções semelhantes àquela aqui descrita. A seguir é apresentada a ferramenta aqui proposta, *CraftContext*, bem como o jogo *Minecraft*, base para a sua construção. Na seção 4 é discutido e implementado um cenário, a fim de demonstrar a utilização da ferramenta. A última seção, por fim, traz as considerações finais sobre o trabalho.

2. Trabalhos Relacionados

Existem diversos desafios relacionados à fase de testes de aplicações sensíveis a contexto. [Sama et al. 2008] aponta uma importante característica inerente às aplicações CA que contribui significativamente para a complexidade da etapa de validação. Segundo [Sama et al. 2008], contexto pode ser classificado como físico (contexto real), sensorial (dados provenientes dos sensores), inferido (premissas inferidas a partir do contexto sensorial) e presumido (conclusão com base nos dados inferidos). Qualquer pequena falha no fluxo de dados entre os diferentes níveis de contexto pode causar inconsistências momentâneas no sistema, levando-o a dar respostas incompatíveis com a situação real.

[1] https://github.com/carolrizzi/CraftContext

Para evitar que esta e outras possíveis complicações passem despercebidas na etapa de validação, é importante o uso de um ambiente de teste robusto que dê suporte à avaliação adequada de tais quesitos.

Conforme comentado anteriormente, utilizar fontes de contexto reais (ou seja, dados de sensores) para efetuar testes é muito difícil. Essa dificuldade se deve, principalmente, à falta de controle sobre os eventos do mundo real, cujos dados geralmente variam com frequência, tornando escassas as chances de reproduzir um mesmo cenário de teste. Além disso, ainda há outras questões envolvidas, como por exemplo, questões financeiras (compra de dispositivos caros ou em grande quantidade), situações de risco (na qual o engenheiro de teste precisa colocar-se em uma situação perigosa), logística (necessário mover-se por grandes distâncias, aguardar um longo período de tempo, etc.), entre outros. Várias pesquisas e trabalhos foram realizados a fim de apresentar uma solução para esse problema.

[Dey et al. 2001] propõe um conjunto de ferramentas que atuam como intermediadoras entre os sensores e a aplicação sensível a contexto. Esse conjunto ferramental é composto por *widgets* de contexto, que são componentes de software hospedados em uma arquitetura distribuída. Cada componente se comunica com um sensor específico e tem como principal objetivo abstrair da aplicação CA detalhes sobre a aquisição das informações contextuais. A ferramenta proposta em [Dey et al. 2001] é um grande avanço e em muito contribui para amenizar a complexidade na comunicação entre aplicação e sensores. Todavia, por utilizar sensores reais na realização dos testes, não soluciona a falta de um ambiente controlado no qual os testes possam ser facilmente reproduzidos.

Para solucionar esse dilema, outros trabalhos propuseram a simulação virtual dos sensores. Desta forma, os dados de contexto passariam a ser obtidos por meio de programas simuladores de contexto, configurados para fornecer informações de acordo com o interesse do engenheiro de testes. Este, por sua vez, não estaria mais suscetível à limitações físicas ou financeiras.

Em [Martin and Nurmi 2006] é proposta uma ferramenta que simula um ambiente virtual semirreal. Esse ambiente é construído com base em modelos que seguem um padrão bem definido. Para usar a ferramenta o desenvolvedor deve fornecer três tipos de modelos: dos agentes, do mundo e do contexto, que devem ser previamente elaborados. Por exemplo, os agentes e seus comportamentos devem ser previamente programados, bem como o desenho dos mapas que definem o mundo. Não há, no entanto, uma ferramenta específica para esta finalidade, acabando por delegar ao engenheiro de testes uma tarefa trabalhosa.

Já a ferramenta proposta no trabalho de [Broens and van Halteren 2006], denominada *SimuContext*, exige apenas a prévia programação dos valores de contexto e suas entidades relacionadas, havendo para isso uma interface apropriada. As possibilidades de contexto são amplas, uma vez que o *SimuContext* não restringe tipos ou formatos de contexto, deixando isso a cargo do programador. Todavia não há suporte à simulação de um mundo virtual, reduzindo a ferramenta a um gerador de dados de contexto pré-programados.

Uma ferramenta interessante, e em grande parte semelhante àquela proposta no corrente trabalho, é apresentada em [Bylund and Espinoza 2002]. Tal ferramenta,

chamada *QuakeSim*, baseia-se no jogo *Quake III Arena* para simular o mundo real. Os jagadores podem se conectar em um mesmo servidor do jogo e interagir entre si, produzindo dados de contexto. Estes dados são capturados pelo *QuakeSim*, que os envia às aplicações CA clientes, exercendo o papel dos sensores. A riqueza do jogo, entretanto, é explorada de forma limitada, tal que apenas os sensores mais comuns (posição, temperatura, velocidade, etc) são simulados pela ferramenta.

A ferramenta aqui proposta, *CraftContext*, possui como diferencial a ampla exploração dos recursos do jogo em que se baseia (*Minecraft*), possibilitando a simulação de uma grande variedade de sensores. Além disso, o estilo de jogo em primeira pessoa permite que o engenheiro de teste se coloque no lugar do usuário, conferindo uma sensação mais real de simulação. Por fim, uma importante característica do *CraftContext* é sua quase completa independência de hardware, plataforma e linguagem de programação, incrementando sua abrangência.

3. CraftContext

A ferramenta aqui proposta, *CraftContext*, utiliza-se do jogo *Minecraft* para simular o mundo que fornecerá os dados de contexto. Nesta seção será apresentado o conceito de contexto, base para a elaboração da ferramenta, as principais características do jogo e os detalhes arquiteturais do *CraftContext*.

3.1. Contexto

O *CraftContext* é uma ferramenta elaborada com o intuito de simular uma fonte de contexto, ou seja, simular um ambiente capaz de fornecer dados de contexto. Mas o que é contexto? Na literatura, contexto é um conceito que aparece sob diversas definições, sendo uma das mais conhecidas e utilizadas a de [Dey 2001]. Em seu trabalho, Dey define contexto como qualquer informação que pode ser utilizada para caracterizar a situação de uma entidade. Uma entidade pode ser uma pessoa, um lugar ou um objeto que seja considerado relevante para a interação entre um usuário e uma aplicação, incluindo o próprio usuário e a própria aplicação.

CraftContext segue um conceito de contexto semelhante, no qual o papel das entidades descritas por Dey são desempenhados pelos jogadores, ambiente, objetos e demais seres presentes no mundo de *Minecraft*. Contexto em *CraftContext* é, portanto, tudo aquilo que ocorre em torno do jogador (inclusive suas ações) e que tenha relevância para ele de alguma forma, caracterizando sua atual situação.

3.2. O Jogo Minecraft

Minecraft é um jogo eletrônico em primeira pessoa, desenvolvido pela empresa Mojang AB [Mojang 2012b], que simula um mundo virtual 3D composto basicamente por blocos. Todos os elementos naturais são representados por cubos de um metro quadrado, formando paisagens que lembram vagamente o famoso brinquedo *LEGO*. Esses blocos podem ser removidos e recolocados em quase todos os lugares, sendo inclusive utilizados como matéria prima para construções.

Seu estilo de jogo *sandbox* concede ao jogador total liberdade de ação e movimento dentro do mundo, sem a obrigatoriedade de alcançar um objetivo específico. *Minecraft* ainda permite que diversos jogadores interajam entre si no mesmo ambiente

(jogo *multiplayer*), enriquecendo a gama de possibilidades de contexto. O jogo possui um vasto território, equivalente a oito vezes a extensão da superfície terrestre e conta com diversos ecossistemas encontrados no mundo real, como florestas, desertos, rios e oceanos.

Minecraft foi escolhido como base da ferramenta principalmente por sua grande variedade de ambientes, recursos e possibilidades, muitos dos quais representam fielmente o mundo real. Além disso, a total liberdade de ação e movimento do jogador é outro fator importante, encontrado em poucos jogos de RPG (*Role Playing Game*). Por fim, a facilidade de desenvolver expansões personalizadas para o jogo foi um fator decisivo na escolha de *Minecraft* como alicerce do *CraftContext*.

3.3. Arquitetura do CraftContext

Para capturar os eventos ocorridos dentro do mundo de *Minecraft* faz-se necessário o desenvolvimento de um plugin de extensão do jogo. Entretanto, o servidor oficial de *Minecraft* não oferece suporte à criação de plugins. Várias versões modificadas do servidor original foram desenvolvidas com a finalidade de suprir essa falta, sendo *Bukkit* [Bukkit Team 2012] o mais conhecido dentre estes. O corrente trabalho utiliza o servidor *Bukkit*, versão 1.1-R4, para o desenvolvimento do plugin *CraftContext*.

CraftContext é desenvolvido inteiramente em Java e utiliza-se da biblioteca fornecida pelo *Bukkit* para implementar as funcionalidades necessárias ao seu propósito. Toda a parte de comunicação remota entre a aplicação cliente e o plugin é realizada com o auxílio de JacORB [JacORB 2012], uma implementação Java *open-source* do padrão CORBA (*Common Object Request Broker Architecture*) [OMG 2012]. Uma importante funcionalidade de CORBA para o corrente trabalho é tornar transparente ao programador a comunicação remota entre as aplicações envolvidas, de forma que as invocações remotas são realizadas como se fossem locais. CORBA provê ainda a independência de recursos em vários níveis, fornecendo suporte à quase todos os tipos de hardware, sistema operacional e linguagem de programação. Com isso, a aplicação cliente não fica limitada aos tipos de recurso utilizados pelo *CraftContext*. Esta característica foi fundamental na escolha de CORBA em detrimento de outras potenciais tecnologias, como Java RMI, que não fornece suporte à independêcia de linguagem, tendo Java como única opção. Além disso, o suporte a serviços de chamadas remotas oferecido por Java RMI é bem simples e limitado se comparado àquele fornecido por CORBA, o qual inclui um robusto conjunto de opões de comunicação através de canais de eventos, recurso que se mostrou fundamental na elaboração da ferramenta proposta.

Três tipos de comunicação são estabelecidos entre a aplicação cliente e o *CraftContext*, os quais foram implementados com o auxílio de JacORB, sendo eles: *message passing*, *request-response* e *publish-subscribe* [Dockhorn Costa 2007]. Na Figura 1 é possível analisar de forma mais detalhada como ocorre a comunicação entre os componentes da arquitetura proposta. O plugin *CraftContext* funciona como um módulo interno ao servidor *Bukkit*. Para se conectar ao servidor, o jogador precisa de um cliente *Minecraft* e uma conta de acesso. Em um mesmo servidor é possível conectar simultaneamente vários clientes *Minecraft*, situação na qual vários jogadores dividem o mesmo ambiente e podem interagir entre si. As ocorrências do jogo são capturadas pelo plugin, que, seguindo o modelo *publish-subscribe*, as publica em canais de evento específicos

(também implementados com JacORB), de acordo com o tipo de evento detectado. Esses canais funcionam como intermediadores de eventos, abstraindo do cliente a localização e condição do servidor que fornece o evento. Abstração similar ocorre ao servidor, que desconhece quantos e quais clientes estão registrados para consumir seus eventos.

Como a aplicação CA está preparada para receber dados de sensores reais e não da ferramenta de teste, faz-se necessário implementar um adaptador, que irá atuar como interface entre a aplicação sensível a contexto e o *CraftContext*. O objetivo desse adaptador é adequar a aplicação CA à interface do *CraftContext*, a fim de que o teste seja realizado sem a necessidade de alterar o código final da aplicação testada. Isso permite, por exemplo, que o *CraftContext* seja utilizado sobre uma aplicação já existente e concluída.

O adaptador deve, então, se registrar nos canais que fornecem eventos do interesse de sua aplicação CA. Através dos canais o plugin fornece informações sobre eventos ocorridos no mundo virtual e sobre requisições enviadas pelos jogadores. Além dos canais de eventos, as aplicações cliente (ou seus adaptadores) podem ainda requisitar serviços diretamente no plugin, tais como alterações do mundo, informações sobre seu estado atual (*request-response*) e envio de mensagens para os jogadores (*message passing*).

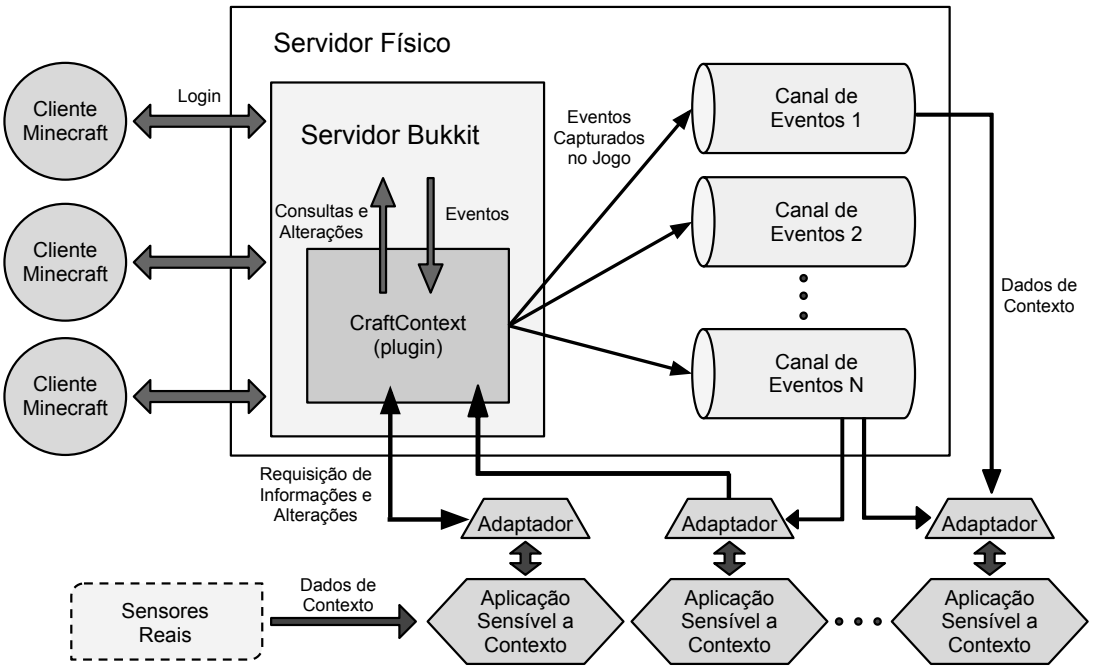

Figura 1. Comunicação entre os Componentes da Arquitetura

A implementação de um cenário facilitará o entendimento sobre o funcionamento da ferramenta de simulação de contexto. Na próxima seção será discutido um estudo de caso sobre o qual será implementada uma aplicação sensível a contexto, cujos testes serão executados com o apoio do *CraftContext*.

4. Estudo de caso

Nesta seção será apresentado um cenário sobre o qual é implementada uma aplicação CA, bem como seu adaptador, que servirá de interface entre a aplicação e o *CraftContext*. Por

fim são discutidos os resultados da fase de testes.

4.1. Cenário

Para o cenário proposto, considere uma aplicação CA para *smartphones* que é capaz de se comunicar via *bluetooth* com um dispositivo que supervisiona o nível de glicose no sangue de um paciente. Ao detectar que a taxa de glicose do paciente está abaixo do normal, caracterizando hipoglicemia leve, a aplicação envia um alerta ao usuário, sugerindo-o algumas ações para a reversão do quadro. Se a taxa de glicose continuar caindo, elevando o nível da hipoglicemia para moderada, a aplicação deve localizar o paciente através de informações de GPS e analisar qual o posto médico mais próximo no momento. Uma mensagem então é enviada ao paciente, informando sobre sua atual condição de saúde, sugerindo-o direcionar-se ao posto médico mais próximo.

Caso a taxa de glicose do paciente alcance níveis críticos (hipoglicemia severa), caracterizando risco de desmaio, a aplicação deve localizar dentre os amigos do paciente aqueles que estejam mais próximos no momento. A estes é enviada uma mensagem de alerta, informando a condição urgente do paciente. Quando o nível de glicose do paciente estabilizar novamente a aplicação deve avisá-lo sobre o fim do estado de alerta.

4.2. Aplicação Sensível a Contexto

A aplicação aqui proposta foi desenvolvida sobre o paradigma de programação orientado a regras. Como apoio ferramental foi utilizado o *JBoss Drools*, uma plataforma baseada na linguagem Java que une regras, processamento de eventos complexos (CEP) e gerência de processos em uma só ferramenta [Bali 2009]. *Drools* possui uma poderosa máquina de execução de regras e uma linguagem própria para a especificação destas, o DRL (*Drools Rule Language*).

A aplicação primeiramente requisita as informações necessárias para registrar o paciente que será monitorado. Também são fornecidos comandos adicionais, que podem ser utilizados a qualquer momento durante a execução da aplicação. Tais comandos incluem o registro de amigos do paciente, que possam ajuda-lo em uma situação de emergência, e o registro de hospitais próximos aos caminhos comumente utilizados pelo paciente. De posse desses dados a aplicação pode então iniciar seu monitoramento.

Um sensor, capaz de medir a taxa de glicose no sangue do paciente em intervalos fixos de tempo, fornece dados para a aplicação continuamente. Esses dados são lidos pela aplicação sensível a contexto e inseridos na memória de trabalho da máquina de regras. Os dados inseridos (também conhecidos como fatos) podem ocasionar a execução de algumas regras, as quais podem ser responsáveis pela inferência de situações ou pela tomada de ações. Em *Drools* as regras são caracterizadas pela por uma estrutura dividida em quatro partes: (1) o nome da regra, (2) atributos opcionais da regra, (3) as condições sobre as quais a regra será ativada e (4) as inferências e ações a serem tomadas caso as condições da regra sejam satisfeitas.

Segue, na Figura 2, uma das regras de inferência implementadas na aplicação proposta. Tal regra define que, se houver uma atualização na taxa de glicose do paciente caracterizando hipoglicemia leve e ainda não existir o evento *HipoglicemiaLeve* (responsável por sinalizar a atual situação do paciente), então tal evento deve ser inserido na memória de trabalho do programa.

```
rule "Situacao: Hipoglicemia Leve"
    when
        LeituraGlicose($taxa : taxa <= Hipoglicemia.MAX_LEVE && > Hipoglicemia.MAX_MODERADA)
        not (exists HipoglicemiaLeve())
    then
        insert(new HipoglicemiaLeve($taxa));
end
```

Figura 2. Exemplo de regra de inferência.

4.3. Adaptador

Para utilizar o *CraftContext* no teste de uma aplicação CA é necessário adaptar os dados provenientes da ferramenta ao formato que a aplicação espera receber. A aplicação proposta nesta seção aguarda a entrada de dados através do canal de comunicação com o monitor do nível de glicose e também realiza consultas ao dispositivo GPS. Tanto o dispositivo de monitoramento quanto o GPS possuem uma API própria, a qual torna possível a comunicação com a aplicação.

A forma mais simples de construir uma interface para atuar entre a aplicação CA e o *CraftContext* é substituir as bibliotecas dessas APIs por outras de mesmo nome, porém de funcionalidade distinta. Por exemplo, suponha que a API do dispositivo GPS forneça um método *request-response* denominado *getCoordinates()*, o qual retorna informações sobre a latitude, longitude e altitude do paciente no momento atual. O adaptador deve, então, possuir um método de mesmo nome, *getCoordinates()*, que fornece os mesmos dados, porém os recupera do jogo e não de sensores reais.

Os dados recuperados do jogo devem ser convertidos para um formato que a aplicação seja capaz de processar. Por exemplo, a aplicação trabalha com uma estrutura de dados denominada *Coordinate*, que não existe no *CraftContext*. Em contrapartida, a ferramenta trabalha com uma estrutura de dados denominada *Position*, que corresponde à estrutura *Coordinate* utilizada pela aplicação. O papel do adaptador, neste caso, é receber do jogo um dado *Position*, transformá-lo em um dado do tipo *Coordinate* e enviá-lo à aplicação sensível a contexto. A Figura 3 ilustra os exemplos dados.

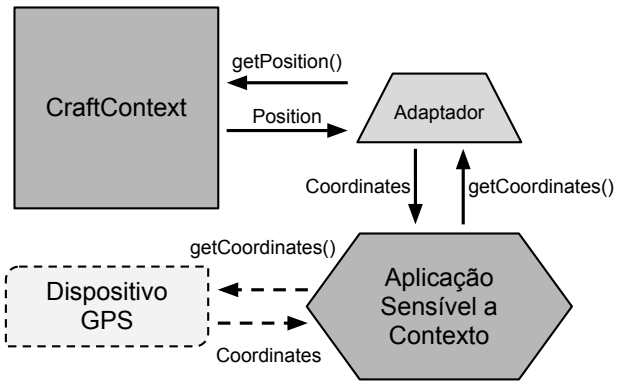

Figura 3. Exemplo de funcionamento do adaptador em uma interação do tipo *request-response*.

É importante ressaltar que ainda não existe uma metodologia definida para a implementação do adaptador e a mesma encontra-se em desenvolvimento no momento.

4.4. Testes

Foi aplicado um teste informal sobre a aplicação do cenário proposto que objetivou analisar principalmente a real contribuição do *CraftContext* enquanto ferramenta de apoio à fase de testes. O teste contou com o auxílio de vários jogadores distintos, conectados no mesmo servidor *Bukkit*, dentre os quais somente um representou o paciente. Cada jogador é identificado por um id, que é o *login* de sua conta *Minecraft*. Diversos estabelecimentos foram construídos dentro do mundo do jogo e a cada um foi atribuído um nome, um tipo (estadia, saúde, comércio, etc.) e uma localização. As variáveis de teste (hospitais registrados, amigos cadastrados, posição do paciente e níveis de hipoglicemia) foram rearranjadas de diversas formas a fim de analisar a reação da aplicação em diversas variações de contexto.

A aplicação respondeu conforme esperado em todas as instâncias de teste aplicadas. Depois de elaborado, o teste completo foi executado em pouco mais de uma hora. É interessante ressaltar que se fossem utilizados sensores reais de localização e de taxa de glicose, bem como o cadastro de hospitais reais, talvez fosse necessário semanas para a conclusão do teste. O uso de ferramentas de simulação de contexto poupa não somente tempo, como também recursos, pois dispensa a aquisição de sensores reais. Outro fator importante é que, além da validação final, testes também precisam ser executados durante o desenvolvimento do sistema, o que ocorre com frequência. Ir a campo para executar um teste real é muito demorado e inviável durante o processo e desenvolvimento de uma aplicação. Dado os resultados obtidos, pode-se concluir que o *CraftContext* desempenhou um relevante papel na superação desses obstáculos.

5. Considerações Finais

O presente trabalho abordou o recorrente problema da fase de testes de aplicações sensíveis a contexto. Tal problema provém da dificuldade na utilização de sensores reais para a captura de contexto de forma controlada e reproduzível, o que pode resultar em testes demorados e pouco confiáveis. As soluções até então apresentadas geralmente possuem aplicabilidade limitada, abrangendo um escopo restrito de aplicações CA.

O corrente artigo propôs uma nova ferramenta, capaz de expandir expressivamente o conjunto de domínios passíveis de solução. Seu diferencial está na base de sua simulação de contexto, o jogo *Minecraft*. Para simular o mundo real, *CraftContext* conta com um jogo rico em detalhes e diversidade de recursos, possibilitando-o fornecer uma ampla gama de dados de contexto. Com isso, aplicações de domínios muito específicos, cuja etapa de validação requisita o uso de sensores pouco usuais, podem também usufruir de uma ferramenta de simulação de fonte de contexto, a fim de simplificar e agilizar seus testes.

A aplicabilidade da ferramenta foi analisada na seção 4. Constatou-se, através de testes informais, que a agilidade na fase de validação foi muito superior com o apoio da ferramenta. A aplicação do cenário proposto na seção 4 foi elaborada com o propósito de exemplificar um domínio bem específico, cujos sensores necessários na etapa de validação fogem ao usual, ou seja, não estão disponíveis em dispositivos populares como *smartphones*, a exemplo do medidor de glicose. Os testes foram integralmente satisfeitos, não havendo qualquer necessidade de contribuição externa à ferramenta para suprir as necessidades dos cenários propostos.

Como trabalhos futuros, há dois interessantes pontos a serem abordados. Contexto é um dado que possui uma série de metainformações relacionadas à sua qualidade, tais como precisão, confiabilidade, latência, entre outros [Broens and van Halteren 2006]. Tem-se, como perspectiva de trabalho futuro, adicionar ao *CraftContext* a especificação de tais metadados em relação às variáveis de teste. Outro interessante recurso a ser adicionado é a combinação de sensores reais e simulados, possibilitando usar a ferramenta para simular somente os sensores que a equipe de testes não possui acesso.

Referências

Bali, M. (2009). *Drools Jboss Rules 5.0 Developer's Guide*. From Technologies to Solutions. Packt Publishing, 1st edition.

Broens, T. and van Halteren, A. (2006). Simucontext: Simply simulate context. In *Proceedings of the International Conference on Autonomic and Autonomous Systems*, pages 1–6, Los Alamitos, California. IEEE Computer Society.

Bukkit Team (2012). Bukkit Forums. `http://bukkit.org`.

Bylund, M. and Espinoza, F. (2002). Testing and demonstrating context-aware services with Quake III Arena. *Communications of the ACM*, 45(1):46–48.

Dey, A. K. (2001). Understanding and Using Context. *Personal and Ubiquitous Computing*, 5(1):4–7.

Dey, A. K., Abowd, G. D., and Salber, D. (2001). A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16(2):97–166.

Dockhorn Costa, P. (2007). *Architectural support for context-aware applications: from context models to services platforms*. PhD thesis, Enschede.

JacORB (2012). JacORB. `http://www.jacorb.org/`.

Martin, M. and Nurmi, P. (2006). A Generic Large Scale Simulator for Ubiquitous Computing. *Annual International Conference on Mobile and Ubiquitous Systems*, 0:1–3.

Mojang (2012a). Minecraft. `http://www.minecraft.net`.

Mojang (2012b). Mojang - Creators of Minecraft. `http://www.mojang.com`.

OMG (2012). OMG's CORBA Website. `http://www.corba.org`.

Sama, M., Rosenblum, D. S., Wang, Z., and Elbaum, S. (2008). Multi-layer faults in the architectures of mobile, context-aware adaptive applications: a position paper. In *Proceedings of the 1st international workshop on Software architectures and mobility*, SAM '08, pages 47–49, New York, NY, USA. ACM.

Shah, S., Ilyas, M., and Mouftah, H. (2010). *Pervasive Communications Handbook*. Taylor and Francis, 1st edition.