Comunicação Orientada a Eventos na Arquitetura de Software Continuum (position paper)

Rodolfo Stoffel Antunes[1], Cristiano André da Costa[1], Cláudio Fernando Resin Geyer[2], Adenauer Corrêa Yamin[3]

[1]UNISINOS – Universidade do Vale do Rio dos Sinos
São Leopoldo, RS, Brasil

[2]UFRGS – Unifersidade Federal do Rio Grande do Sul
Porto Alegre, RS, Brasil

[3]UCPel – Universidade Católica de Pelotas
Pelotas, RS, Brasil

rodolfo.s.antunes@gmail.com, cac@unisinos.br, geyer@inf.ufrgs.br, adenauer@ucpel.tche.br

Resumo. *Aplicações para a computação ubíqua tornam-se complexas, pois devem tratar diversas questões de implementação. A infra-estrutura de software Continuum tem por objetivo simplificar o desenvolvimento destas aplicações, através da integração de um framework e um middleware específicos para a computação ubíqua. Este artigo apresenta a pesquisa desenvolvida durante o projeto do serviço communicator, presente no middleware do Continuum, bem como o modelo desenvolvido a fim de guiar a implementação deste serviço.*

Abstract. *Ubiquitous computing applications become complex, because they must address various implementation issues. The Continuum software infrastructure is aimed at simplifying the development of these applications, through the integration of a framework and a middleware specific for the ubiquitous computing. This paper presents the research conducted during the design of the service communicator, present in the Continuum middleware, as well as the model created to guide the implementation of the service.*

1. Introdução

Mark Weiser [Weiser 1991] afirma que a tecnologia da informação deve ser mais integrada ao cotidiano, tornando-se transparente e auxiliando os usuários a lidar com a sobrecarga diária de informações. Pesquisas na computação ubíqua englobam diversas áreas, dentre elas sistemas distribuídos e computação móvel [Satyanarayanan 2001]. [Costa et al. 2008] apresenta um estudo sobre os desafios da computação ubíqua, que é utilizado para definir um modelo de infra-estrutura de software para a computação ubíqua. Este modelo tem como principais elementos um *framework*, para auxiliar no projeto de aplicações, e um *middleware*, responsável pela execução destas aplicações.

Baseado neste modelo, o Continuum [Costa 2008] é uma infra-estrutura de software baseada em um *framework* e um *middleware*, que simplificam o tratamento das questões impostas pela computação ubíqua. O *middleware* do Continuum é baseado em uma arquitetura orientada a serviços (SOA). Uma SOA é composta por um conjunto de

serviços base, que podem ser utilizados para a criação de novos serviços e aplicações através de composição e gerenciamento [Papazoglou and Georgakopoulos 2003]. Alguns dos serviços que compõe o *middleware* do Continuum, apresentados em [Costa 2008], possuem apenas um modelo da interface que deveriam possuir. Um destes serviços é o *communicator*, que provê um sistema simples de comunicação orientada a eventos, seguindo os princípios do modelo publicar-assinar (*publish-subscribe*) [Eugster et al. 2003], para uso nas aplicações do Continuum.

O modelo publicar-assinar permite que um objeto seja notificado sobre eventos ocorridos em outros objetos, podendo executar operações como resultado da mudança no estado de um objeto de interesse [Coulouris et al. 2001]. Neste modelo, um nodo provedor publica os eventos disponíveis para observação, enquanto nodos clientes criam assinaturas para o recebimento dos eventos de interesse. A comunicação orientada a eventos permite que provedores e assinantes sejam desacoplados nas dimensões de espaço, tempo e sincronização [Eugster et al. 2003]. Provedores e assinantes se comunicam através de um serviço de notificações, e portanto não mantém contato direto. Também não é necessário que as partes que se comunicam estejam ativas simultaneamente. As operações utilizadas, por fim, são métodos não-bloqueantes, permitindo que a comunicação seja realizada assincronamente. A anonimidade e assincronismo inerentes deste modelo de comunicação o tornam ideal para sistemas distribuídos [Huang and Garcia-Molina 2004].

Sistemas publicar-assinar podem utilizar três metodologias para o gerenciamento de assinaturas [Eugster et al. 2003]. Em um sistema baseado em tipos, as assinaturas são especificadas a partir dos tipos de dados contidos nas notificações. Um sistema de tópicos define assinaturas a partir de tópicos pré-definidos no sistema. Um sistema baseado em conteúdo, por fim, define assinaturas com base nos atributos presentes nas notificações, através de uma linguagem de pesquisa.

Este artigo apresenta a pesquisa realizada para determinar os requisitos e metodologia a serem empregados no projeto do *communicator*, além do modelo criado para guiar sua implementação. A seção 2 apresenta características de outros sistemas que utilizam a comunicação por eventos. A seção 3 apresenta, em detalhes, o modelo proposto para a implementação do serviço no *middleware* do Continuum. Finalmente, na seção 4, são apresentadas considerações finais relativas ao trabalho desenvolvido e sua continuidade.

2. Trabalhos Relacionados

O serviço *communicator* implementa um sistema simples de comunicação por eventos para o Continuum. Para definir os requisitos do serviço, foram estudados outros sistemas publicar-assinar.

O Hermes [Pietzuch and Bacon 2002] é um sistema publicar-assinar baseado em um *middleware* formado por uma rede de *overlay*, que interliga os nodos responsáveis por executar os mecanismos para a comunicação entre os clientes. A rede de *overlay* permite que o hermes se adapte a mudanças na topologia do *middleware*, além de tornar eficiente o roteamento de notificações. O Hermes utiliza assinaturas baseadas em tipos. Cada tipo de dado dos eventos possui um nodo específico do *overlay* para a gerencia de assinaturas, que é encontrado através de uma função *hash* que usa o tipo de dado como argumento. O mecanismo de tipos empregado pelo Hermes traz diversas vantagens para o *middleware*, mas o torna dependente de uma biblioteca de tipos, o que limita sua heterogeneidade.

O Echo [Eisenhauer et al. 2006] é um *middleware* publicar-assinar para comunicação em alta performance, que utiliza canais de eventos para a propagação de mensagens. Estes canais utilizam um mecanismo de assinatura baseado em tópicos, ou tipos, e são entidades descentralizadas, mantidas pelos próprios clientes do serviço, que propagam notificações diretamente entre si. As mensagens do Echo utilizam uma biblioteca de tipos própria, que permite a descrição de tipos similares à uma linguagem estruturada. O *middleware* empregado pelo Echo requer o poder computacional dos clientes para sua execução, trazendo problemas para dispositivos limitados. A conectividade intermitente presente em ambientes ubíquos também pode influenciar negativamente no *middleware*.

O *Java Messaging Services* (JMS) [Hapner et al. 2002] é um conjunto de interfaces que permite a programas Java o uso de sistemas de comunicação por eventos, dentre as quais há uma interface para sistemas publicar-assinar. O JMS assume a existência de um servidor central, com acesso a um meio de armazenamento estável, para a gerência das comunicações entre clientes. As assinaturas do JMS utilizam o modelo de tópicos, e podem ser de dois tipos: duráveis, quando a entrega da notificação é garantida, mesmo que haja falhas de conexão; e não-duráveis, quando não há tal garantia. O JMS ainda permite que as notificações sejam armazenadas em um meio estável, evitando que falhas provoquem a perda de mensagens não entregues. A principal desvantagem do JMS é a forte associação do serviço à linguagem Java, limitando sua heterogeneidade. A escalabilidade do serviço também pode ser prejudicada, devido ao servidor central de gerenciamento.

3. Modelo de Comunicação

Em [Costa 2008] é proposto um modelo de abstração que permite a representação, no contexto de aplicações ubíquas, de entidades relevantes do mundo real. No modelo, uma *CoDimension* engloba todas as entidades que podem estar contidas no ambiente de uma aplicação, sendo composta por um conjunto de células, denominadas *CoCells*, que representam os lugares do ambiente modelado. O modelo também permite a representação de objetos, denominados *CoNodes*, os quais englobam dispositivos eletrônicos presentes nas células, como computadores, sensores, ou dispositivos móveis. Cada célula possui um nodo especial, denominado *CoBase*, responsável pela execução de serviços básicos para seu gerenciamento. Cada CoBase presente no ambiente possui uma instância do *communicator* em execução, permitindo que os dispositivos presentes em uma célula possam se comunicar através da publicação e assinatura de eventos. Dispositivos de diferentes células também podem se comunicar através do serviço. Neste caso, os eventos são roteados entre as instâncias do serviço utilizadas por estes dispositivos.

A Figura 1 ilustra as relações existentes entre dispositivos que utilizam o *communicator*. As linhas tracejadas representam as relações de comunicação estabelecidas entre um cliente e um servidor. Tais relações são dinâmicas, e se estabelecem quando os eventos de um provedor se enquadram na assinatura de um cliente. Linhas contínuas apresentam os canais de comunicação de fato estabelecidos pelo serviço. O fluxo de eventos em uma célula é controlado pela instância do *communicator* presente no CoBase. Quando dispositivos de diferentes células se comunicam, o fluxo de mensagens é roteado através das instâncias do *communicator* dessas células.

3.1. Gerência de Assinaturas

Um dispositivo que irá publicar eventos através do *communicator* deve criar um registro inicial no serviço. Este registro é composto por um conjunto de atributos que descre-

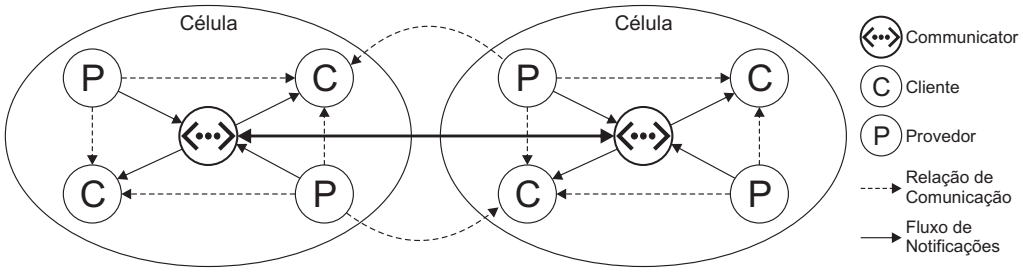

Figura 1. Relações de comunicação entre dispositivos que utilizam o *communicator*

vem propriedades gerais dos eventos que serão publicados pelo dispositivo. Os atributos presentes no registro variam de acordo com as aplicações que utilizam o serviço para se comunicar, e portanto o número destes atributos será variável, e não haverá atributos obrigatórios. Há a possibilidade de que um dispositivo possua mais de um registro no *communicator*, caso ele publique eventos que possuam características variadas, que devam ser expressadas através dos atributos gerais no registro.

Um dispositivo que deseja receber os eventos publicados no serviço, por sua vez, deve primeiramente criar uma assinatura. Uma assinatura será composta por dois filtros, cada um formado por um conjunto de atributos e valores. O primeiro conjunto, denominado **filtro de assinatura**, é comparado com os atributos em cada registro de provedor cadastrado no serviço. Nos casos em que os atributos do registro estão de acordo com o filtro de assinatura, é criada uma associação entre esta assinatura e o registro do provedor. Os eventos publicados por um provedor, então, são apenas processados em relação às assinaturas associadas ao seu registro.

O segundo filtro definido é denominado **filtro de eventos**. Os atributos definidos neste filtro são utilizados para realizar a seleção de quais notificações, dentre as recebidas, serão entregues ao cliente. Ao receber uma notificação de um provedor, o serviço verifica quais assinaturas estão associadas com o provedor que enviou a notificação. Para cada assinatura encontrada, então, os atributos da notificação são comparados com o filtro de eventos da assinatura. Para os casos em que essa comparação é positiva, a notificação é associada com a assinatura, para que ela seja posteriormente enviada ao respectivo cliente.

A Figura 2 ilustra a atuação dos filtros de assinatura e de eventos contidos em uma assinatura de um cliente. Na ilustração, os provedores P_1 e P_3 foram selecionados segundo o filtro de assinatura, e portanto apenas as notificações destes provedores são consideradas para a assinatura. As notificações geradas por P_1 e P_3 são então submetidas ao filtro de eventos da assinatura, que seleciona com base nos atributos das notificações aquelas que serão entregues ao cliente. Na ilustração, o filtro de eventos selecionou, dentre as notificações recebidas, N_1P_1, N_2P_1, e N_1P_3. As notificações selecionadas são, por fim, armazenadas pelo serviço até que elas sejam entregues ao cliente.

3.2. Processamento de Eventos

Após a criação do registro inicial, um provedor pode enviar para o *communicator* mensagens de notificação, indicando a ocorrência do evento registrado. Ao receber uma notificação, o serviço a processa de acordo com as assinaturas cadatradas pelos clientes, associando a notificação com as assinaturas cujos filtros definidos se enquadram com os atributos presentes na notificação e no registro do provedor. O serviço armazena a men-

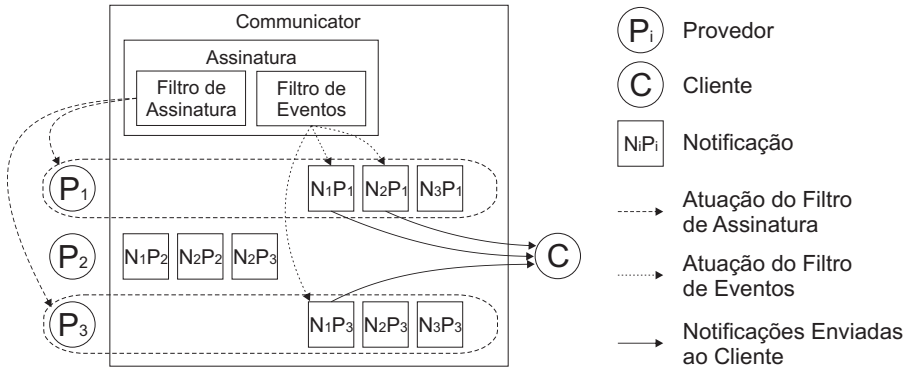

Figura 2. Atuação dos filtros de assinatura e de eventos

sagem até ela seja entregue para todos os clientes que tenham uma assinatura associada com a notificação. Após recuperada por todos os clientes, a notificação é descartada.

Uma mensagem de notificação é composta por um conjunto de atributos, utilizados pelo provedor para descrever detalhes do evento ocorrido. O provedor pode utilizar quantos atributos forem necessários para descrever o evento. Dentre os atributos, porém, deve haver um conjunto básico, utilizado pelo *communicator* para o controle das notificações. Estes atributos são: o identificador do registro de provedor no *communicator*, que é recebido pelo provedor durante seu cadastro no serviço; o número de série da notificação, representado por um valor incrementado a cada notificação enviada ao *communicator* pelo provedor; e por um *timestamp* contendo o momento em que o evento ocorreu.

3.3. Distribuição

Durante a inicialização do *communicator*, o serviço de descoberta do Continuum é utilizado para encontrar as outras instâncias do *communicator* no ambiente. O serviço entra em contato, então, com cada instância descoberta, registrando-se como uma instância do serviço, de forma que todas instâncias possam trocar mensagens diretamente entre si.

Cada novo registro de provedor criado em uma instância do *communicator* é propagado para todas outras instâncias, de forma que os provedores de eventos sejam conhecidos em todo o ambiente. Assinaturas podem ser, então, associadas com provedores de outras instâncias do serviço. Quando tal associação ocorre, a instância onde a assinatura foi criada submete uma cópia desta para a instância onde o provedor se cadastrou. A instância do provedor, neste caso, realiza o processamento do filtro de eventos para cada notificação relacionada com a assinatura, e envia para a instância do cliente cada notificação válida para a assinatura. A instância do cliente, por fim, armazena as notificações para que sejam posteriormente entregues.

3.4. Armazenamento

Cada instância do *communicator* utiliza um banco de dados relacional para armazenar as informações utilizadas durante sua execução. Caso uma falha ocorra, as informações presentes no banco de dados são utilizadas para restaurar o serviço. O banco de dados utilizado é composto por três tabelas principais. A primeira tabela é utilizada para armazenar os atributos cadastrados nos registros de provedores. A segunda tabela contém as assinaturas de clientes, armazenando os filtros nelas definidos. A terceira tabela, por fim,

armazena os eventos publicados através do serviço, e que ainda não foram recuperados por todos os clientes que devem recebê-los.

4. Considerações Finais

Os avanços tecnológicos e a popularização dos dispositivos móveis torna a computação ubíqua cada vez mais próxima da realidade. A complexidade envolvida no desenvolvimento de uma aplicação para tal ambiente, porém, é cada vez mais desafiadora, devido ao número de questões que devem ser consideradas. Integrando um *framework* e um *middleware* específicos para a computação ubíqua, a infra-estrutura de software Continuum apresenta diversas vantagens para o desenvolvimento destas aplicações.

A partir do modelo apresentado neste artigo, será desenvolvido um protótipo do *communicator*, implementado na linguagem Java. A interface do serviço utilizará os padrões SOAP e WSDL para comunicação. As notificações e assinaturas serão descritas através de padrões baseados na linguagem XML, de forma que seja possível avaliar o desempenho do padrão XPath, utilizado para pesquisas em documentos XML, como ferramenta para a filtragem das notificações.

Referências

Costa, C. (2008). *Continuum: A Context-aware Service-based Software Infrastructure for Ubiquitous Computing*. PhD thesis, UFRGS.

Costa, C., Yamin, A., and Geyer, C. (2008). Towards a general software infrastructure for ubiquitous computing. *IEEE Pervasive Computing*, 7(1):64–73.

Coulouris, G., Dollimore, J., and Kindberg, T. (2001). *Distributed Systems: Concepts and Design*. Addison-Wesley, Harlow.

Eisenhauer, G., Schwan, K., and Bustamante, F. (2006). Publish-subscribe for high-performance computing. *IEEE Internet Computing*, 10(1):40–47.

Eugster, P., Felber, P., Guerraoui, R., and Kermarrec, A. (2003). The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131.

Hapner, M., Burridge, R., Sharma, R., Fialli, J., and Stout, K. (2002). Java message service. Disponível em <http://java.sun.com/products/jms/docs.html>. Acesso em Out. 2008.

Huang, Y. and Garcia-Molina, H. (2004). Publish/subscribe in a mobile environment. *Wireless Networks*, 10(6):643–652.

Papazoglou, M. and Georgakopoulos, D. (2003). Introduction: Service-oriented computing. *Communications of the ACM*, 46(10):24–28.

Pietzuch, P. and Bacon, J. (2002). Hermes: A distributed event-based middleware architecture. In *22nd International Conference on Distributed Computing Systems Workshops, 2002, Vienna*, pages 611–618, Washington. IEEE.

Satyanarayanan, M. (2001). Pervasive computing: Vision and challenges. *IEEE Personal Communications*, 8(4):10–17.

Weiser, M. (1991). The computer for the 21st century. *Scientific American*, 265(3):94–104.