Gerenciamento Dinâmico de Modelos em Computação Sensível ao Contexto

Rodrigo Hernandez Soares[1], **Ricardo Couto Antunes da Rocha**[1]

[1] Instituto de Informática – Universidade Federal de Goiás (UFG)

{rodrigosoares,ricardo}@inf.ufg.br

Resumo. *Este artigo apresenta uma arquitetura para gerenciamento dinâmico de modelos em computação sensível ao contexto, baseada em CEP (Complex Event Processing) e em OSGi . A arquitetura proposta permite que modelos de contexto sejam modificados, sem que as aplicações precisem sofrer interrupções ou deixem de perceber os novos modelos ou regras. O artigo apresenta ainda a implementação de um cenário de chat sensível ao contexto em ambiente distribuído, a qual que permite validar a arquitetura proposta.*

Abstract. *This paper presents an architecture based on CEP and OSGi, that allows dynamic management of context models for context-aware computing. The proposed architecture aims to keep updates in context models transparent to applications and to avoid disruptions an application and in the event engine. In order to validade this work, the paper presents an implementation of context-aware chat application in a distributed environment.*

1. Introdução

O principal objetivo da computação sensível ao contexto é permitir que aplicações reajam a ocorrências de determinadas situações. Para que isso seja possível, as informações contextuais relacionadas às situações juntamente com as condições que caracterizam a ocorrência das mesmas precisam ser previamente especificadas. Normalmente, essa especificação é feita através de modelos de contexto.

Este trabalho trata do gerenciamento dinâmico de modelos de contexto, que envolve a atualização, em tempo de execução, de certos elementos que compõem um modelo, com a finalidade de adaptá-lo a situações de caráter dinâmico e imprevisível. As discussões desse artigo não se limitam a uma abordagem particular de especificação de modelos e nem propõe uma nova abordagem. O principal objetivo deste trabalho é especificar e implementar uma arquitetura para gerenciamento dinâmico de modelos baseada em Esper[1].

O Esper é uma tecnologia que implementa os fundamentos de CEP (*Complex Event Processing*), que é a análise de eventos em tempo real com o objetivo de permitir respostas imediatas a mudanças contextuais. A escolha de CEP, e em particular de Esper, foi motivada pelo fato de essa tecnologia oferecer um dos mais ricos conjuntos de construções se comparada com outras soluções similares, conforme conclui o trabalho [Margara and Cugola 2012]. A arquitetura proposta e os fundamentos utilizados na

[1] http://esper.codehaus.org/

sua estruturação são a principal contribuição deste trabalho. Uma outra contribuição é a modularização de modelos através da plataforma de serviços OSGi para aplicativos *Java*.

O restante do artigo é organizado como a seguir. A Seção 2 apresenta um cenário motivacional para manutenção dinâmica de modelos de contexto. A Seção 3 discute o modelo de sistema relativo à arquitetura proposta e discute os impactos que atualizações dinâmicas em modelos contextuais podem causar nos sistemas que gerenciam os mesmos e em aplicações sensíveis ao contexto. A Seção 4 apresenta e discute a arquitetura proposta. A Seção 5 apresenta um estudo de caso em gerenciamento dinâmico de modelos contextuais. Por fim, a Seção 7 apresenta conclusões e discute os trabalhos futuros.

2. Cenário de Manutenção Dinâmica de Modelos de Contexto

Modelos de contexto são representações abstratas das condições e situações que são relevantes para o universo de discurso de uma aplicação. A literatura em computação sensível ao contexto usualmente explora casos em que a modelagem de contexto é estabelecida estaticamente. Nestes casos, as aplicações são desenvolvidas baseadas em um conhecimento prévio do modelo de contexto, que não pode ser alterado ao longo da execução das aplicações, sob risco de sofrer interrupções ou inconsistências. Uma mesma informação contextual pode ser inferida de maneiras distintas, dependendo do próprio contexto do usuário, da execução da aplicação ou do ambiente provedor da informação contextual (sensores, agentes de inferência e máquinas baseadas em eventos).

Por exemplo, considere o contexto "ocupado", que define a disponibilidade de um usuário, utilizado por uma aplicação de troca de mensagens instantâneas como ConChat [Ranganathan et al. 2002] para, tanto atualizar um indicador visual para o usuário sobre o estado de cada um dos seus contatos, como para permitir o início de uma comunicação. A aplicação é notificada sempre que o estado de um usuário transitar entre "ocupado" e "disponível". Neste cenário, o contexto "ocupado" pode ser inferido de diversas maneiras, como informado explicitamente por uma interação com o usuário, baseando-se na sua localização, nas atividades agendadas para aquele período, no estado do local onde ele se encontra (e.g. *está ocorrendo uma reunião*), no estado de uso de seu telefone, só para citar algumas possibilidades. No caso da inferência baseada em localização, pode-se considerar, por exemplo, que se um usuário se encontra em seu escritório, então a semântica desse local é *local de trabalho*, e o sistema pode utilizar as tarefas programadas na agenda para definir o estado de disponibilidade. Por outro lado, se o usuário em questão está em um hospital, a semântica de *local de trabalho* não deve mais ser aplicada, pois ele assume um papel diferente e próprio daquele local.

Uma arquitetura para computação sensível ao contexto deve permitir que uma mesma aplicação possa, em contextos diferentes, perceber o estado de "ocupado" de uma pessoa de acordo com os diferentes significados que o estado possa assumir. Embora tais semânticas ainda precisem ser modeladas, a escolha e a avaliação da regra de inferência que deve ser aplicada em cada caso ainda é uma tarefa definida estaticamente nas arquiteturas atuais. Esta natureza dinâmica dos modelos sugere a adoção de arquiteturas para sistemas sensíveis ao contexto que suportem modelos dinâmicos sem interferir na consistência do funcionamento das aplicações pré-existentes.

3. Modelo de Sistema para Ambientes de Computação Sensível ao Contexto

Arquiteturas para computação sensível ao contexto baseiam o gerenciamento de contexto em sistemas baseados em eventos (SBE), tais como sistemas *publish/subscribe*. Algumas soluções preferem desenvolver o seu próprio sistema baseado em eventos para, entre outros motivos, evitar limitarem a descrição das situações contextuais à linguagem de assinatura de eventos suportada pelo SBE de propósito geral. Em ambos os casos, a abordagem de modelagem de contexto adotada por uma arquitetura e as consequentes limitações na manutenção dos modelos em tempo de execução, são fortemente ligadas à escolha do SBE na qual a arquitetura é baseada. Este artigo trata o problema de manutenção dinâmica de modelos de contexto na perspectiva de tais sistemas, sobretudo com relação ao funcionamento das suas máquinas de processamento de eventos. Para tal, este trabalho adota o modelo de sistema proposto em [Margara and Cugola 2012], ilustrado na Figura 1, no qual os sistemas baseados em eventos são chamados de *Sistemas de Processamento de Fluxos de Informações*, ou IFPS, que integram geradores de eventos (*Sources*) e consumidores de eventos (*Sinks*) por meio da execução de regras que realizam a interpretação e inferência de um evento a partir de outros.

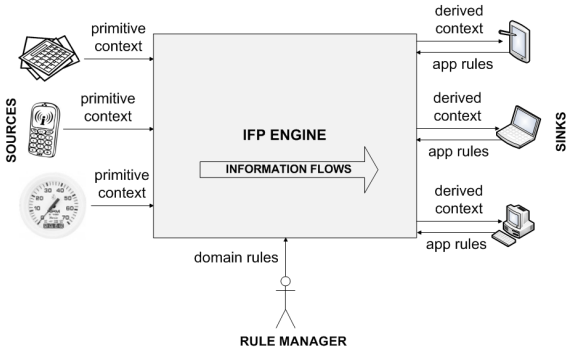

Figura 1. Modelo de funcionamento do sistema proposto.

Os *Sources* são responsáveis por criar os fluxos de informações que entram no componente *IFP Engine*. No universo da computação sensível ao contexto, eles podem ser vistos como provedores de informações contextuais primitivas, que são informações obtidas diretamente, sem a necessidade de processos de inferência intermediários. Exemplos típicos de *Sources* são sensores. Os *Sinks* são os elementos interessados nos eventos contextuais produzidos por uma *IFP Engine*, e representam as aplicações sensíveis ao contexto.

A *IFP Engine* é responsável por processar as informações geradas pelos *Sources* de acordo com um conjunto de regras, com o objetivo de produzir informações contextuais derivadas, ou seja, informações inferidas pelas regras de processamento. As aplicações sensíveis ao contexto registram interesse na ocorrência de eventos contextuais derivados. As regras podem ser classificadas em *Regras de Domínio* e *Regras de Aplicação*. Enquanto as *Regras de Aplicação* especificam os eventos nos quais os *Sinks* estão diretamente interessados, as *Regras de Domínio* determinam as condições contextuais associadas a uma situação de interesse, de maneira a produzir eventos de mais alto nível a partir de outros eventos. Uma regra é especificada em uma linguagem própria da *IFP Engine* adotada no IFPS.

Os *Rule Managers* são pessoas especialistas no domínio de operação de um IFPS, responsáveis por modelar as regras, informações de contexto e entidades do cenário sensível ao contexto associado ao domínio do qual são especialistas. Quando aplicada a computação sensível ao contexto, um IFPS delimita um modelo de contexto como sendo as descrições das regras e tipos de eventos, mantidos e manipulados por uma *IFP Engine*.

3.1. Componentes de Modelo

Componente de modelo são os conceitos fundamentais a partir do qual um modelo é construído. Cada abordagem de modelagem de contexto especifica seus próprios componentes de modelo. Por exemplo, modelagem de contexto baseada em ontologias define *classes* (conceitos) e *relações entre classes* como alguns dos componentes de modelo. Neste trabalho e em consonância com o modelo de sistema adotado, definimos quatro componentes de modelo: **eventos contextuais**, **entidades**, **regras** e **operadores**. Eventos contextuais representam alterações em situações contextuais de interesse de uma ou mais aplicações ou serviços. Entidades representam os objetos que compõem o cenário de computação sensível ao contexto e cujo estado é compreendido como informação contextual. Regras representam condições, expressões e consultas, que permitem tanto a aplicações como ao restante do sistema indicar situações ou condições que podem produzir novos eventos contextuais e/ou associá-los a entidades. Como este trabalho adotou CEP/Esper como estudo de caso para implementação da arquitetura para gerenciamento dinâmico de modelos, as regras serão descritas na linguagem EPL, que é implementada pelo Esper. Operadores representam primitivas de manipulação de eventos contextuais. Por exemplo, um contexto de localização geográfica pode oferecer o operador *distância*. Operadores existem independentemente de aplicações e são utilizados na modelagem de regras. As diferentes abordagens de modelagem de contexto permitem a especificação parcial ou total desses elementos.

4. Uma arquitetura para gerenciamento dinâmico de modelos

A arquitetura proposta nessa seção para gerenciamento dinâmico de modelos de contexto é baseada em *Esper* e foi projetada para execução na plataforma de serviços OSGi. A arquitetura deve implementar mecanismos que permitam a criação e atualização dinâmica de modelos contextuais. Além disso, os componentes incorporados dinamicamente a um modelo de contexto devem ser reconhecidos por aplicações que utilizam os mesmos. Outro requisito é que atualizações em modelos de contexto não podem interromper o processamento de regras, mesmo que as definições dos componentes a serem atualizados estejam sendo utilizadas no contexto de execução das regras. Por fim, os modelos de contexto devem ser desacoplados dos componentes que implementam os mecanismos de inferência e notificação, pois os modelos são compartilhados por esses componentes. Além disso, o desacoplamento facilita o gerenciamento dinâmico de modelos contextuais, pois dessa forma, atualizações são realizadas em partes específicas do sistema.

4.1. Arquitetura

A Figura 2 ilustra a arquitetura proposta para o componente *IFP Engine*, que é formado por quatro camadas: *gerenciamento de eventos de entrada*, *gerenciamento de eventos de saída*, *camadas de processamento de eventos* e *camada de gerenciamento de modelos de contexto*.

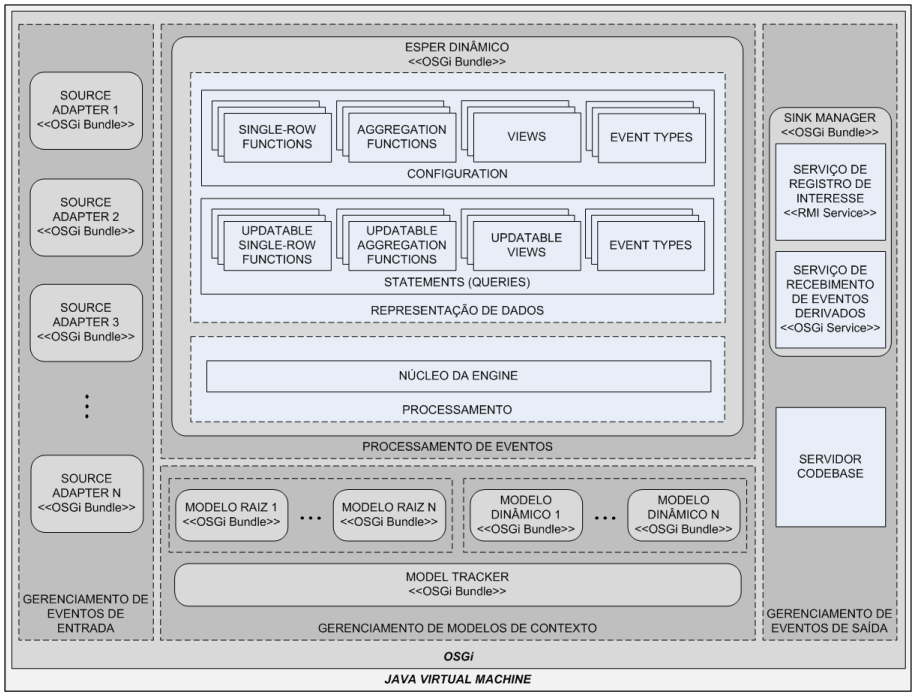

Figura 2. Arquitetura da *IFP Engine* proposta.

A camada de *Gerenciamento de Modelos de Contexto* é responsável por manter os modelos de contexto e controlar o seu ciclo de vida. Modelos de contexto são conjuntos de componentes de modelo que descrevem conceitos e regras associados a um cenário de computação sensível ao contexto. Um modelo de contexto é implementado como *bundle* (módulo) OSGi e pode especificar dependência por conceitos declarados em outros modelos de contexto. Cada modelo de contexto pode ser independentemente instalado, atualizado ou removido, desde que as restrições resultantes das dependências com outros modelos sejam respeitadas. Por exemplo, um modelo A dependente de um modelo B só pode ser instalado no ambiente caso o modelo B também esteja instalado. As dependências entre modelos traduzem relações de especialização entre conceitos especificados nos modelos.

O ciclo de vida de um modelo de contexto possui quatro estados: *Carregado*, *Ativo*, *Inativo* e *Suspenso*. Quando um modelo está no estado Carregado, suas dependências foram resolvidas e ele pode ser utilizado pelas aplicações e pela engine IFP, o que o leva para o estado Ativo. Quando um modelo está neste estado, qualquer alteração no modelo tem o potencial de interromper o funcionamento das aplicações sensíveis ao contexto que são baseadas nele. Quando uma alteração ocorre, o modelo passa para o estado intermediário Suspenso e, todas as operações de uso do modelo, como instanciar um tipo de contexto ou executar uma regra, são suspensas. Uma vez finalizada a atualização, um modelo pode passar para o estado Ativo ou para o estado Inativo, quando a atualização colocou o modelo em um estado inconsistente. Cada ciclo de vida de um modelo de contexto é mapeado para um ou mais ciclo de vida de bundle gerenciado pelo OSGi, o qual é responsável por resolver as dependências entre modelos e carregar as realizações dos componentes de modelo na máquina virtual Java.

A camada de *Processamento de Eventos* é composta por uma versão modificada

do Esper, chamada de Esper Dinâmico, que oferece a possibilidade de atualizar, em tempo de execução, as implementações dos operadores de semântica variável. Nesse contexto, atualizar em tempo de execução significa alterar a implementação de um operador que pode estar sendo utilizado em uma ou várias consultas.

Cada elemento de um modelo de contexto é mapeado em tempo de execução na correspondente construção na máquina Esper. Na implementação atual, os modelos de contexto permitem a implementação de tipos de eventos, regras EPL, e três diferentes construções Esper que permitem a implementação de operadores: funções single-row, funções agregadas e views, conforme apresentado na Figura 2. O *Model Tracker* é um componente da camada de gerenciamento de modelos responsável por fazer o mapeamento entre modelos de contexto e construções da engine Esper. Este componente acompanha mudanças nos estados do ciclo de vida de um modelo. Quando um modelo de contexto é instalado ou modificado, o *Model Tracker* então realiza a modificação equivalente no ambiente em tempo de execução do Esper. Os detalhes de implementação dessa arquitetura podem ser encontrados em [Soares 2012].

5. Estudo de Caso

Esta seção apresenta um estudo de caso de implementação de um cenário de *chat* sensível ao contexto, discutido na Seção 2, como forma de demonstrar como a arquitetura proposta permite a manutenção de modelos de contexto dinâmicos. A aplicação de *chat* é um cliente que obtém o estado de disponibilidade dos *buddies* a partir das informações contextuais providas pelo middleware, que permite ao usuário identificar o momento mais adequado para iniciar uma comunicação. A aplicação foi projetada para utilizar dois possíveis estados de disponibilidade, *Ocupado* ou *Disponível*, determinados a partir de regras independentes da aplicação.

A aplicação será utilizada em uma universidade para reduzir as interrupções indesejadas que influenciam a produtividade de alunos e professores, e aplicada em dois cenários particulares, sendo que o segundo cenário exige modificações no modelo de contexto. Essas modificações ficam transparentes para a aplicação, que mantém as mesmas consultas por eventos contextuais na *IFP Engine*. Por falta de espaço, esta seção apresentará apenas versões simplificadas dos cenários, sobretudo com relação às regras EPL utilizadas em cada modelo. O trabalho [Soares 2012] apresenta maiores detalhes acerca da modelagem e implementação dos cenários aqui discutidos.

5.1. Cenário e Modelo de Contexto Inicial

A Figura 3 ilustra o cenário inicial de uso do *chat*. Os eventos contextuais disseminados pelos *Sources* e consumidos pelos *Sinks* são identificados em termos do evento de alto nível e os eventos específicos recebidos de fato durante a execução do cenário. Desta maneira, de acordo com a figura, a aplicação de *chat* interpreta o evento recebido como do tipo $Ocupado(Pessoa, Local)$, enquanto que o evento disseminado pela *IFP Engine* é $EmAula$, que é uma especialização do evento anterior.

O cenário utiliza os eventos de $Localizacao$ e $Atividade$ para inferir a disponibilidade dos usuários. A localização é obtida através de leitores de *smartcards* instalados na entrada de cada sala de aula do prédio do departamento. Uma leitura de localização pode ser de dois tipos: $LocalizacaoEntrada$, que indica que uma pessoa entrou em uma

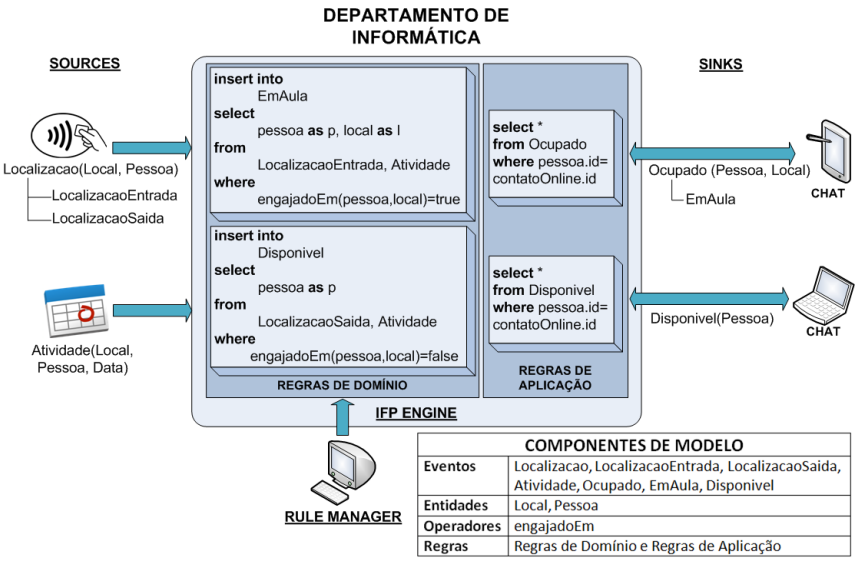

Figura 3. Cenário de uso da aplicação de *chat* do departamento de informática.

sala de aula, ou $LocalizacaoSaida$, que indica que uma pessoa saiu de uma sala de aula. A atividade é obtida a partir de um sistema de calendário do departamento que disponibiliza horários e locais de aulas associados a alunos matriculados e professores designados para as aulas em questão. As regras de domínio que determinam a inferência dos eventos $EmAula$, especialização de $Ocupado$, e $Disponivel$ utilizam os eventos contextuais mencionados e a operação $engajadoEm$, conforme ilustrado na figura. A operação $engajadoEm$ determina se uma pessoa está, de fato, engajada em uma atividade e, para isso, leva em conta outros fatores particulares ao evento em questão. Por exemplo, para a atividade $EmAula$, a operação pode levar em conta como a pessoa está fazendo uso dos dispositivos, qual é o seu papel em sala de aula e outras informações coletadas do ambiente (não ilustradas na figura). O principal aspecto a se notar é que $engajadoEm$ é uma operação de semântica variável, dependente da atividade executada pela pessoa e que, por isso, pode ser implementada de maneira distinta em cada um dos possíveis sub-tipos de $Atividade$.

5.2. Novo Cenário e Modelo de Contexto

Considere agora que o departamento de informática sedie um evento de pesquisa, para o qual alguns professores assumiram responsabilidades como coordenar palestras e sessões de posteres. Como este cenário é particular e temporável, é natural que o modelo de contexto anterior não descreva adequadamente as novas condições de (in)disponibilidade de professores associadas ao evento. Entretanto, para que a mesma aplicação de *chat* continue sendo relevante e consistente neste novo cenário, o modelo de contexto deverá ser modificado, passando a conter regras adicionais que especifiquem a participação nas atividades do evento.

A Figura 4 ilustra as alterações realizadas no modelo de contexto decorrentes do novo cenário. Todas as regras do cenário anterior foram mantidas, e três novas regras foram acrescentadas para inferência das situações $CoordenandoPalestra$, $CoordenadorSessaoPoster$ e $Disponivel$.

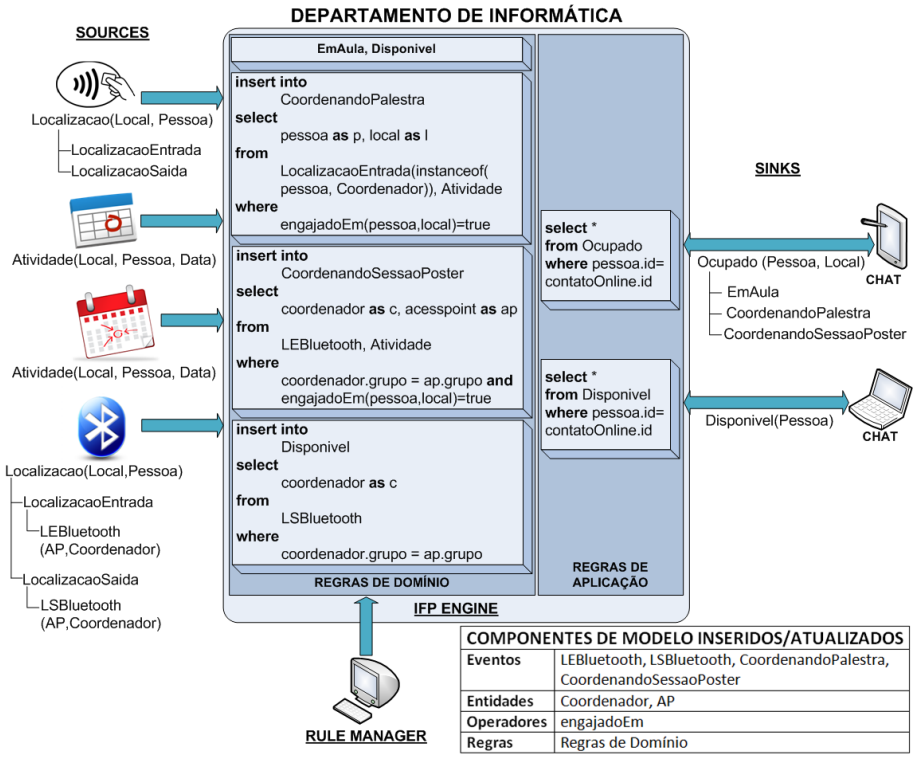

Figura 4. Novo cenário com modelo de contexto atualizado.

O novo cenário adiciona aos *Sources* um provedor de atividades do evento de pesquisa, de acordo com o cronograma do evento, e novos provedores de localização que determinam a entrada ($LEBluetooth$) ou saída ($LSBluetooth$) de locais onde ocorrem atividades do evento. Em ambos os casos, os eventos publicados são especializações dos eventos $Localizacao$ e $Atividade$ já previstos no cenário e modelo anterior.

O novo cenário também adota novas regras de inferência de eventos de indisponibilidade, modelada nos eventos $CoordenandoPalestra$ e $CoordenandoSessaoPoster$, que indicam a indisponibilidade de um professor por estar, respectivamente, coordenando uma palestra ou coordenando uma sessão de poster. Nestas duas situações, um professor passa a ser representado pela entidade $Coordenador$.

As três regras introduzidas no cenário, conforme ilustrado na Figura 4, utilizam ainda o operador $engajadoEm$, cuja implementação é particular às atividades de $CoordenandoPalestra$ e $CoordenadorSessaoPoster$. Por exemplo, um professor fica engajado na coordenação de uma palestra antes mesmo do início planejado, para garantir que ela iniciará no tempo planejado. Por outro lado, ele deixa de ficar enjagado se, por algum motivo, a palestra termina antes do previsto, ou se sua atividade de coordenação é limitada aos instantes iniciais de apresentação do palestrante e nos momentos finais guardados para perguntas. Esta semântica de operador é particular ao tipo de atividade que está ocorrendo e, por isso, não pode ser assumida para outras atividades.

5.3. Discussão

O estudo de caso apresentado evidencia algumas consequências provocadas por atualizações sobre modelos. A primeira consequência do cenário é a necessidade de

reconhecimento de especializações de entidades e eventos em aplicações. Embora tal reconhecimento seja comum em linguagens que permitem polimorfismo de tipos, no ambiente distribuído é necessário o reconhecimento do modelo dos novos componentes de modelo (entidades e eventos) para que o sistema se comporte consistentemente em tempo de execução. No cenário, isso ocorre quando eventos $EmAula$ e $CoordenandoPalestra$ são disseminados para aplicação de *chat* que está preparada para lidar com eventos $Ocupado$, só para citar uma das situações geradas pelo cenário. A arquitetura lida com esses problemas utilizando um protocolo de distribuição baseado em Java RMI em conjunto com um servidor de codebase para carga dinâmica dos novos componentes de modelos.

A segunda consequência do cenário é a necessidade de atualização dinâmica da operação $engajadoEm$. A modelagem adotada no cenário, com o uso da operação $engajadoEm$, permite uma construção incremental e adaptativa das condições que levam a uma situação de indisponibilidade. Em geral, tais situações não podem ser adequadamente previstas e dependem de uma série de condições em tempo de execução.

Por fim, o cenário *pode* gerar uma situação de inconsistência de eventos entre o modelo inicial e o atualizado: uma a regra pode inferir que um professor está ocupado em sala (modelo inicial), quando o professor está, na verdade, envolvido no evento. As operações $engajadoEm$ auxiliam a eliminar tais conflitos, uma vez que elas presumivelmente não irão produzir resultados ambíguos. Entretanto, isso depende de uma modelagem cuidadosa da operação por parte do *Rule Manager*. Por este motivo, uma arquitetura deveria oferecer mecanismos para lidar com tais conflitos e inconsistências, o que não é objetivo deste trabalho.

6. Trabalhos Relacionados

A atualização dinâmica de modelos é um problema que envolve tanto as máquinas de gerenciamento de contexto, como as abordagens de modelagem de contexto. De fato, quanto mais simples for a abordagem de modelagem de contexto, mais simples também é a solução do problema atacado. Por exemplo, abordagens que utilizam modelos mais simples como par-valor, permitem naturalmente a atualização dinâmica dos modelos, mas em contra-partida dificultam o desenvolvimento das aplicações complexas, assim como a modelagem de situações complexas. Neste trabalho, escolhemos a implementação Esper de CEP como estudo de caso, com o objetivo de selecionar uma abordagem que oferecesse um mínimo de expressividade dos modelos e uma rica linguagem para construção de consultas.

Abordagens de modelagem baseadas em ontologias oferecem a construção de modelos dinâmicos. Entretanto, tais abordagens não oferecem um mecanismo de disseminação de eventos contextuais, que deve ser implementado como um componente externo ao gerenciador de ontologias. Como resultado, embora os modelos possam ser dinâmicos, o dinamismo não é capturado nas consultas feitas pelas aplicações, que não podem lidar transparentemente com essas atualizações.

Há dois trabalhos que, de fato, lidam com o problema de atualização dinâmica de modelos: [da Rocha 2009] e CA3M [Taconet et al. 2009]. O trabalho [da Rocha 2009] oferece soluções para um ambiente distribuído, no qual a mudança nos modelos é um efeito da mobilidade entre domínios, enquanto que este trabalho foca no efeito das atualizações nas máquinas de eventos, quando consideradas isoladamente. Os protocolos

inter-domínios implementados naquele trabalho podem ser incorporados neste trabalho, assim como a arquitetura interna do sistema de gerenciamento de contexto poderia ser incorporada naquele. Neste sentido, os dois trabalhos são complementares. CA3M, por sua vez, tem como objetivo permitir a incorporação dinâmica de novos tipos de *entidades* e *eventos contextuais*, que são chamados de *observables*. Instâncias de *observables* são produzidas por sistemas chamados *collectors*, e uma mesma instância do *middleware* proposto no trabalho mencionado pode se comunicar com *collectors* com diferentes APIs e que publicam diferentes tipos de *observables*. Uma vantagem deste trabalho com relação a CA3M são as atualizações dinâmicas dos operadores utilizados em uma regra e a criação dinâmica de regras, as quais são definidas em tempo de compilação no CA3M. Além disso, a modularização dos modelos de contexto implementada nesse trabalho permite que diferentes tipos de aplicação sejam criadas com base na infraestrutura proposta. Em CA3M, o uso de um modelo de contexto atômico limita o uso do *middleware* a um tipo específico de aplicação.

7. Conclusões

A principal contribuição deste trabalho é a arquitetura proposta para gerenciamento dinâmico de modelos de contexto, sendo que a metodologia utilizada para classificação de componentes de modelo pode ser aproveitada no projeto de arquiteturas baseadas em outros tipos de *IFP Engines*. Outra contribuição importante é a abordagem utilizada para modularização de modelos de contexto, que facilita o gerenciamento dinâmico através do descoplamento de modelos dos demais componentes que integram a arquitetura.

Como trabalhos futuros, pretendemos expandir o conjunto de operações de atualização permitidas sobre modelos de contexto, que nesse trabalho foram limitadas a inclusão de novos eventos e entidades na hierarquia de componentes e mudança de implementação de operadores através de *proxies*. Pretendemos também implementar protocolos de comunicação inter-domínios similares aos discutidos em [da Rocha 2009], com o intuito de considerar os efeitos que a mobilidade pode causar na gerência dinâmica de modelos de contexto.

Referências

da Rocha, R. C. A. (2009). *Context Management for Distributed and Dynamic Context-Aware Computing*. PhD thesis, Departamento de Informatica, Pontificia Universidade Catolica do Rio de Janeiro, Rio de Janeiro.

Margara, A. and Cugola, G. (2012). Processing flows of information: from data stream to complex event processing. In *ACM Computing Surveys*. (to appear).

Ranganathan, A., Campbell, R. H., Ravi, A., and Mahajan, A. (2002). ConChat: A context-aware chat program. *IEEE Pervasive Computing*, 1:51–57.

Soares, R. H. (2012). Gerenciamento dinâmico de modelos de contexto: Estudo de caso baseado em esper. Master's thesis, Instituto de Informática, Universidade Federal de Goiás.

Taconet, C., Kazi-Aoul, Z., Zaier, M., and Conan, D. (2009). Ca3m: A runtime model and a middleware for dynamic context management. In *Proc. The 11th International Symposium on Distributed Objects, Middleware, and Applications (DOA'09)*.