Uma infra-estrutura reflexiva para aplicações dependentes de contexto

Carlos Eduardo G. Tosin e Luiz Lima Jr.

***Resumo* – Aplicações dependentes de contexto utilizam informações do ambiente que as cerca para adaptarem o seu comportamento. Este trabalho propõe uma infra-estrutura reflexiva, com baixo acoplamento e orientada a eventos chamada CxFramework para auxiliar no desenvolvimento desta categoria complexa de aplicações. A modelagem das informações de contexto é feita por meio de ontologias, que permitem a inferência de informações de contexto de mais alto nível. A aplicação do modelo em serviços de TV digital móvel visando melhor interatividade é discutida e aspectos de implementação do protótipo são detalhados.**

***Palavras-Chave* – Dependência de contexto, computação ubíqua, infra-estrutura, interatividade, TV digital móvel**

***Abstract* – Context-aware applications use information from their surrounding environment to adapt their behavior. This paper proposes a reflexive, loose-coupled, event-based infrastructure named CxFramework to aid the development of such complex applications. Context information is modeled using ontologies, from which higher-level context information can be inferred. The applicability of the model in mobile digital TV, aiming better interactivity, is discussed and the implementation aspects of the prototype are detailed.**

***Keywords* – Context awareness, ubiquitous computing, infrastructure, mobile digital TV**

I. INTRODUÇÃO

A capacidade de adaptação de uma aplicação ao ambiente no qual está inserida é de grande importância para a construção de sistemas flexíveis e dinâmicos. Esta capacidade, conhecida como *dependência de contexto* (ou *sensibilidade a contexto*), torna-se praticamente indispensável quando a mobilidade dos sistemas é levada em conta. Enquanto seres humanos têm uma capacidade natural de se adaptar aos contextos dentro dos quais estão e de utilizá-los enquanto se comunicam, é necessário definir, estruturar e organizar as informações de contextos para enriquecer e otimizar as interações homem-máquina e entre computadores.

A construção de aplicações dependentes de contexto requer, portanto, que o seu ambiente de inserção seja capaz de fornecer informações de forma estruturada, de forma relevante à aplicação (inferindo eventos de mais alto nível) e de maneira desacoplada (permitindo a autonomia da aplicação). Além disso, a presença das aplicações (e usuários) em determinado contexto acabam por alterar o próprio contexto, o que impõe a necessidade de uma infra-estrutura adaptável a esta presença e que retroalimente as próprias aplicações com novas informações e eventos. Esta característica conhecida como *reflexividade* permite a construção de categorias de aplicações, particularmente relevantes em cenários interativos, como é o caso do modelo de TV digital móvel discutido na Seção IV.

O objetivo deste trabalho é propor uma arquitetura reflexiva, que também seja genérica e fortemente desacoplada, para a integração de informações de contexto em uma plataforma distribuída através da definição de um modelo de suporte baseado em um serviço de inferência e fazer com que estas informações sejam disponibilizadas às aplicações sensíveis a contextos. Este processo visa simplificar o desenvolvimento de aplicações, transferindo grande parte da complexidade de detecção de mudanças de contexto à infra-estrutura genérica, que chamamos de CxFramework. O protótipo do CxFramework foi implementado utilizando CORBA *(Common Object Request Broker*) como plataforma distribuída devido a sua grande adoção, estabilidade, maturidade, por lidar satisfatoriamente com a heterogeneidade (típica de ambientes móveis) e possuir um conjunto de recursos (serviços) adequados. Desta forma, o CxFramework integra informações de contexto à plataforma CORBA, muito embora o modelo arquitetural definido possa ser incorporado virtualmente em qualquer *middleware*, com maior ou menor dificuldade. O CxFramework é uma infra-estrutura que, além da reflexividade permite a criação de *federações de contexto*, possibilitando a interferência controlada entre contextos distintos ou sobrepostos.

O restante do artigo está assim organizado. Na Seção II são abordados temas que fundamentam a arquitetura proposta, a saber: contexto e dependência de contexto, ontologias e a plataforma CORBA. Na Seção III, a arquitetura proposta é apresentada inicialmente sob um aspecto conceitual e depois sob a perspectiva de implementação. Nesta seção, são também discutidos aspectos de reflexividade, federação de contextos e coleta de lixo, esta última importante para a escalabilidade do sistema. A Seção IV mostra como o CxFramework pode ser integrado ao ambiente da TV digital interativa. Na Seção V, alguns trabalhos alinhados com a nossa proposta são citados e comentados. A Seção VI discute conclusões do trabalho e atividades futuras possíveis.

II. CONCEITOS FUNDAMENTAIS

Esta Seção introduz alguns temas necessários para compreender a arquitetura proposta e a sua implementação.

A. Contexto e Dependência de Contexto

Segundo [1], um contexto é qualquer informação que pode ser utilizada para caracterizar a situação de uma entidade, a qual pode ser uma pessoa, lugar ou objeto considerado relevante na interação entre um usuário e uma aplicação.

Carlos Eduardo G. Tosin e Luiz Lima Jr., Programa de Pós-Graduação em Informática, Pontifícia Universidade Católica do Paraná, Curitiba, Brasil, E-mails: tosin@ppgia.pucpr.br, laplima@ppgia.pucpr.br.

A dependência de contexto nas aplicações e serviços computacionais está relacionada à capacidade que determinada aplicação tem em utilizar informações de contexto no seu funcionamento. É desejável que aplicações dependentes de contexto dependam de infra-estruturas externas responsáveis por tudo o que diz respeito ao contexto, tornando o código reutilizável entre diversas aplicações de uma forma padronizada, através de uma infra-estrutura independente das aplicações que a utilizam.

Informações de contexto vindas diretamente de sensores nem sempre são úteis para as aplicações. Dessa forma, é necessário que haja uma forma de representar informações de contexto em um nível mais alto. Uma das formas de fazer essa representação computacionalmente é através do uso de ontologias.

B. Ontologias

Uma ontologia define formalmente um conjunto comum de termos usados para descrever e representar um domínio [2]. Os quatro elementos que compõem uma ontologia são: indivíduos, classes, atributos e relacionamentos. O propósito geral de uma ontologia é apresentar um meio de classificação de indivíduos, uma vez que ontologias representam uma descrição de conceitos e relacionamentos [3]. Dessa forma, são um instrumento promissor para modelar informações de contexto devido ao seu alto grau de expressividade.

C. Plataforma CORBA

A plataforma CORBA – criada pela OMG [4] – visa facilitar a criação de aplicações distribuídas, tornando transparente ao desenvolvedor questões como localização dos objetos na rede, diferenças de linguagem de programação, de sistema operacional e de plataforma de *hardware*. A plataforma CORBA possui uma gama de serviços dos mais variados tipos. Dentre eles, dois merecem destaque por serem os mais diretamente utilizados pela nossa infra-estrutura: o *Notification Service* e o *Trading Service*.

Notification Service [5]. É uma extensão do *Event Service* [5], o primeiro serviço de eventos em CORBA. É um serviço de comunicação assíncrona baseado em produtores (*suppliers*), responsáveis pela criação dos eventos; e consumidores (*consumers*), responsáveis pela recepção dos eventos. Este serviço suporta dois modos de entrega de eventos: *push* (o produtor coloca a informação no canal e notifica os consumidores) e *pull* (os consumidores buscam a informação desejada no canal). A existência de um serviço dessa natureza possibilita que a plataforma dê suporte à comunicação assíncrona, que permite uma clara separação entre produtores e consumidores da informação.

Trading Service [6]. A plataforma CORBA possui um serviço de nomes (*Naming Service*), o qual permite localizar um objeto a partir de um nome conhecido. O serviço de nomes é útil e muito importante na plataforma CORBA, mas possui uma desvantagem: o nome do objeto tem que ser conhecido de antemão para que seja possível localizá-lo. Com o objetivo de contornar esta limitação, foi criado o serviço de *trading*. No *Trading Service*, a busca é feita pelas características dos objetos. Nele, existe a figura do servidor – que registra os objetos no serviço de *trading* – e do cliente – que procura objetos no serviço de *trading* através de características.

III. CXFRAMEWORK

A arquitetura do CxFramework é composta de quatro serviços e um módulo coletor de lixo (*garbage collector*), como ilustra a Fig. 1.

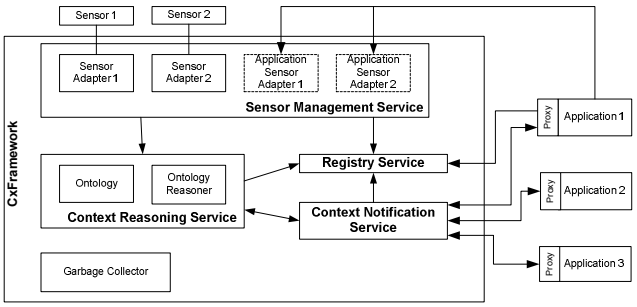

Fig. 1. Estrutura do CxFramework.

O *Sensor Management Service* é responsável pela comunicação entre a infra-estrutura e os sensores externos (representados pelos *sensors* na Fig. 1) e possui um conjunto de *Sensor Adapters*. Os sensores são dispositivos físicos (como sensores de temperatura, calendários, relógios, etc.) ou "virtuais" (outras aplicações) capazes de prover informações de contexto. O *Sensor Adapter* é responsável por encapsular o código de comunicação com um sensor externo, agindo como um representante do sensor dentro da infra-estrutura. Esta abordagem expõe à infra-estrutura um conjunto de componentes homogêneos (os *Sensor Adapters*), a qual não precisa conhecer detalhes de implementação e dos sensores propriamente ditos. O administrador do CxFramework é responsável por implementar o código de comunicação com os sensores e também por registrá-los. Os *Application Sensor Adapters* são *Sensor Adapters* especiais, que serão explicados na seqüência.

Quando um *Sensor Adapter* detecta uma mudança de estado, ele deve realizar apenas uma ação: notificar o *Context Reasoning Service* que o estado do sensor mudou. O *Sensor Adapter* deve informar seu ID único e qual o novo estado do sensor representado por ele.

O *Context Reasoning Service* é o serviço que recebe eventos vindos dos *Sensor Adapters* e os processa, inferindo informações de contexto de mais alto nível. O CxFramework utiliza ontologias para modelar informações de contexto e inferir informações de contexto de alto nível. O administrador da infra-estrutura é responsável por prover a ontologia e as regras de inferência.

O motivo da escolha de ontologias como forma de modelagem de informações de contexto é devido ao fato que é fácil inferir novas ontologias a partir de uma ontologia original, utilizando regras de inferência. Esta característica já vem embutida no conceito de ontologia, que é uma forma de classificação de indivíduos. Além disso, ontologias podem ser criadas por pessoas que não estão envolvidas com a área de programação, através de ferramentas gráficas que, muitas vezes, são intuitivas e fáceis de usar. Outras abordagens para modelagem de contexto se mostraram limitadas (como a utilização de pares de chave e valor) ou fortemente acopladas com a programação da infra-estrutura (como a modelagem de contexto orientada a objetos). Em ambos os casos, o mecanismo de inferência deve ser implementado pelo próprio programador, o que aumenta a complexidade e diminui a flexibilidade da solução.

Quando notificado pelos *Sensor Adapters*, o *Context Reasoning Service* muda a ontologia corrente baseada no novo estado do sensor enviado pelo *Sensor Adapter*. O mapeamento entre os *Sensor Adapters* e a ontologia está descrito na *Event Table*. A partir do ID do *Sensor Adapter*, a *Event Table* fornece a informação de qual relação da ontologia é afetada pelo *Sensor Adapter* que detectou a mudança do estado do sensor.

Quando a ontologia muda, o *Context Reasoning Service* utiliza as regras de inferência para inferir uma nova ontologia. Todos os relacionamentos da ontologia que são alterados são enviados para o *Context Notification Service*.

O *Context Notification Service* notifica as aplicações a respeito de mudanças de contexto. Este serviço gerencia o *Complete Event Channel* e os *Specific Event Channels*, que serão abordados mais detalhadamente na Seção III.B. Esta Seção também explicará como o *Context Notification Service* funciona sob o aspecto de implementação.

O *Registry Service* mantém o registro dos seguintes componentes da infra-estrutura:

- *Serviços da infra-estrutura*: Todos os serviços da infra-estrutura se registram no *Registry Service*. A razão para isto é que os serviços precisam se comunicar e o *Registry Service* é responsável por fornecer as referências aos serviços desejados (cada serviço possui um nome único).
- *Eventos de mudanças de contexto*: Todos os eventos de mudança de contexto suportados pela infra-estrutura devem ser registrados no *Registry Service*. Aplicações dependentes de contexto procuram por eventos de contexto que podem ser do seu interesse para que possam ser notificadas quando mudanças ocorrem. A Seção III.B explica o que são estes eventos de contexto.

O *Proxy* é outro componente importante na arquitetura. Ele não está diretamente inserido na infra-estrutura, mas é parte da aplicação conectada ao CxFramework. O *proxy* é uma API importada pela aplicação cuja responsabilidade é esconder da aplicação o código específico da plataforma (isto é, da plataforma distribuída usada na implementação do CxFramework). Toda a comunicação entre a aplicação dependente de contexto e a infra-estrutura deve, obrigatoriamente, ser feita através do *proxy*.

O *Proxy* também cumpre um papel importante em permitir que a infra-estrutura seja suficientemente genérica para ser construída sobre outras plataformas. Se as aplicações invocassem código específico de uma plataforma ao invés de utilizar o *Proxy*, elas teriam que ser alteradas se o CxFramework fosse implementado sobre uma plataforma diferente. É importante ressaltar que o *proxy* é uma camada fina de software, sendo facilmente implementado caso seja necessário.

Após detalhar a estrutura do CxFramework, é importante destacar a presença dos *Application Sensor Adapters*, destacados anteriormente na Fig. 1. Os *Application Sensor Adapters* funcionam como qualquer outro *Sensor Adapter*, exceto pelo fato de que são criados e gerenciados pelas aplicações dependentes de contexto conectadas ao CxFramework. Quando uma aplicação se registra na infra-estrutura, ela também assume o papel de sensor. Devido a isso, aplicações podem alterar o contexto do ambiente (usando seus *Application Sensor Adapters*) da mesma forma que qualquer outro sensor faria. Este recurso faz com que o CxFramework seja uma infra-estrutura reflexiva, isto é, a própria infra-estrutura é influenciada pelo ambiente que ela representa.

A. Reflexividade e Federação de Contextos

As aplicações dependentes de contexto, ao se conectarem à infra-estrutura, assumem também o papel de sensores. Utilizando os seus *Application Sensor Adapters*, elas podem fazer alterações no contexto, que se refletem na infra-estrutura. Em outras palavras, a infra-estrutura não só influencia as aplicações como também é influenciada por elas, caracterizando a reflexividade.

Utilizando a reflexividade, é possível montar uma *federação de contextos* onde dois ou mais contextos distintos, representados por instâncias diferentes de CxFrameworks, possam interferir uns nos outros por meio de trocas de informações, fazendo com que alterações em um contexto sejam sentidas pelos outros contextos federados. Isto é feito por intermédio de uma aplicação conectada a duas ou mais infra-estruturas desempenhando o papel de "ponte". Ela é notificada a respeito de mudanças relevantes de um contexto repassando esta informação para outro contexto utilizando um *Application Sensor Adapter*.

B. Aspectos de Implementação

O CxFramework foi implementado na linguagem Java sobre o OpenORB [7], uma implementação em Java da especificação CORBA. Cada serviço da infra-estrutura possui sua própria IDL [5] e é representada por um objeto CORBA.

O *Registry Service* encapsula o CORBA *Naming Service* e *Trading Service*. Desta forma, tudo o que é registrado no *Registry Service* é internamente registrado em um desses dois serviços.

O administrador deve implementar os *Sensor Adapters*, que também são objetos CORBA, e registrá-los no *Sensor Management Service*. Existe um arquivo de configuração em formato XML que contém a lista de *Sensor Adapters* registrados no serviço. Cada *Sensor Adapter* deve ter seu próprio ID único e uma classe Java que o representa (esta classe deve implementar uma interface comum a todos os *Sensor Adapters*). Quando o serviço é iniciado, o arquivo de configuração é lido e os *Sensor Adapters* são instanciados e iniciados.

O administrador deve fornecer a ontologia e as regras de inferência para configurar o *Context Reasoning Service*. O CxFramework suporta ontologias especificadas em linguagem OWL [8] e formatadas em XML. Ferramentas desenvolvidas por terceiros, como o Protégé [9] podem ser utilizadas para criar a ontologia. Ferramentas desta natureza normalmente são capazes de exportar as ontologias criadas para o formato OWL. O motor de inferência de ontologias utilizado pelo CxFramework é o framework Jena [10]. Conseqüentemente, o formato das regras de inferência fornecidas pelo administrador da infra-estrutura deve seguir o formato estabelecido pelo Jena.

Outro componente importante que deve ser fornecido pelo administrador na configuração do *Context Reasoning Service* é a *Event Table*. A *Event Table* é uma tabela que mapeia um *Sensor Adapter* a um relacionamento na ontologia. Por exemplo, um *Sensor Adapter* chamado **S1** é responsável por fornecer a temperatura de uma sala. Neste cenário, a *Event Table* mapearia **S1** a um indivíduo da ontologia chamado **sala** e

a um atributo chamado **temperatura**. Através da utilização desta tabela, o serviço é capaz de saber qual parte da ontologia será alterada quando o *Sensor Adapter* enviar uma notificação de mudança de contexto. A *Event Table* também é configurada por um arquivo XML.

O *Context Notification Service* é responsável por notificar as aplicações a respeito de mudanças no contexto. A Fig. 2 mostra seus componentes.

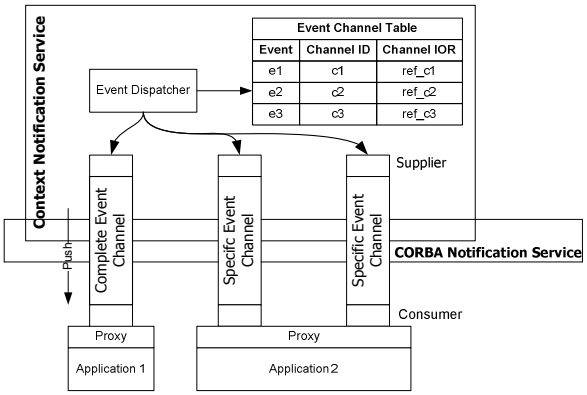

Fig. 2. Arquitetura do *Context Notification Service*.

A comunicação entre o *Context Notification Service* e as aplicações acontece através de canais de eventos do CORBA *Notification Service*, onde o *Context Notification Service* é o produtor e a aplicação é o consumidor da informação. As aplicações podem ser conectadas ao *Complete Event Channel*. Neste caso, elas serão notificadas quando qualquer mudança de contexto ocorrer. Já quando conectadas a um *Specific Event Channel*, elas serão notificadas apenas a respeito da mudança de contexto que o canal gerencia (os *Specific Event Channels* são responsáveis apenas por um tipo de mudança de contexto). É importante mencionar novamente que existe um *proxy* entre a aplicação e os canais de eventos por questões de transparência. Além disso, por questões de otimização, os *Specific Event Channels* são criados apenas quando alguma aplicação solicita o registro no canal. Isto significa que eles são criados dinamicamente sob demanda.

Quando o *Context Notification Service* recebe uma mudança de contexto vinda do *Context Reasoning Service*, ele imediatamente coloca o evento de contexto no *Complete Event Channel*. Depois disso, o serviço procura na *Event Channel Table* qual é o canal responsável por entregar às aplicações o evento de contexto ocorrido. Se nenhuma entrada na tabela for encontrada, significa que nenhuma aplicação está interessada no evento. Neste caso, o evento é descartado. Já se existir a entrada, significa que alguém está interessado em ser notificado a respeito do evento. As entradas na tabela são criadas no momento em que as aplicações registram o interesse em determinado evento. Esta entrada associa o evento de contexto com um ID único de canal de eventos. Além disso, este novo evento é registrado no *Registry Service*, permitindo que as aplicações que usam o *Registry Service* para procurar por eventos de contexto sejam capazes de encontrá-lo. Por exemplo, um possível evento de contexto poderia ser o recurso **sala** (indivíduo da ontologia) ligado à propriedade **temperatura** (atributo da ontologia), o que poderia ser descrito como **temperatura da sala**. O *Registry Service* registra os objetos de eventos de contexto no CORBA *Trading Service*. Se o evento for encontrado na tabela, o serviço verifica se o canal já foi criado. Caso não tenha sido, nada acontece. Caso contrário, o evento de contexto é colocado no *Specific Event Channel* associado.

As aplicações possuem duas formas de obter referências aos *Specific Event Channels* quando o modo de comunicação *push* é utilizado. A primeira forma é quando elas buscam eventos de contexto utilizando o *Registry Service*, onde são retornados eventos de contexto de acordo com as características desejadas, juntamente com as devidas referências aos *Specific Event Channels*. A segunda forma é quando as aplicações se registram no *Complete Event Channel*. Eventos de contexto colocados neste canal também estão associados com seus *Specific Event Channel* correspondentes. Logo, em ambas as situações, as aplicações são capazes de se conectar aos canais de eventos que as interessam. Quando o modo de comunicação *pull* é utilizado, os canais de eventos não são utilizados. Neste caso, as aplicações buscam diretamente um evento de contexto previamente conhecido e recebem seu valor corrente de forma síncrona.

É importante perceber que, apesar das tarefas do administrador da infra-estrutura demandarem grande esforço, esta abordagem simplifica o desenvolvimento de aplicações dependentes de contexto, já que todo código relacionado ao contexto é encapsulado pela infra-estrutura.

C. Coleta de Lixo

O CxFramework funciona num cenário altamente dinâmico. Conseqüentemente, um mecanismo eficiente de coleta de lixo é necessário.

O coletor de lixo (*garbage collector*) não é considerado um serviço da infra-estrutura por não expor nenhuma interface. Ele é um agente que atua em diversas partes da infra-estrutura, coletando tudo o que não está mais sendo usado pelas aplicações.

O primeiro cenário é aplicado aos *Specific Event Channels*. Cada canal tem um TTL (tempo de vida ou *time to live*), baseado no último acesso. Quando o TTL é alcançado, o *garbage collector* pede ao *Context Notification Service* para destruir o canal. Mas antes de fazer isto, o *Context Notification Service* avisa as aplicações conectadas ao canal que o mesmo está prestes a ser destruído, permitindo que elas decidam o que fazer. Se as aplicações decidirem se conectar novamente ao canal, ele será criado de novo. Já se nenhuma aplicação estiver conectada ou solicitar reconexão ao canal, ele permanecerá inexistente.

O segundo cenário é aplicado aos eventos contidos na *Event Channel Table* (que faz parte do *Context Notification Service*) e eventos registrados no *Registry Service*. Estes eventos também possuem um TTL associado. Eventos cujo TTL expirou e que estão associados a canais de eventos que não estão criados podem ser removidos da *Event Channel Table* e do *Registry Service*, pois eles estão ativos por bastante tempo e nenhuma aplicação registrou interesse nestes eventos. Este cenário está diretamente relacionado ao primeiro cenário explicado anteriormente. O coletor de lixo primeiro deve solicitar a destruição do canal (como explicado no primeiro cenário) e depois fazer a checagem dos eventos que podem ser destruídos.

O terceiro cenário é relacionado aos *Application Sensor Adapters*, os quais são dinâmicos, isto é, existem durante o período que a aplicação está conectada ao CxFramework.

Quando as aplicações desconectam, seus *Application Sensor Adapters* devem ser destruídos, juntamente com os canais gerados pra notificar seus eventos, já que eles não serão mais utilizados. Nesta situação, aplicações devem marcar seus *Application Sensor Adapters* como destrutíveis antes de desconectarem do CxFramework. Fazendo isto, o coletor de lixo é capaz de identificar quais *Application Sensor Adapters* podem ser destruídos (a destruição é feita pelo *Sensor Management Service* a pedido do coletor de lixo).

IV. Uso do CxFramework em Aplicações Interativas da TV Digital

A tecnologia da TV digital eleva a experiência de interatividade dos usuários a um novo nível. Com ela, os programas da TV poderão ser enriquecidos com informações adicionais, como ver os dados de um filme que está sendo exibido, a possibilidade de comprar um produto que aparece na tela ou identificar o(s) canal(ais) mais assistido(s) no momento. Enfim, o espectador deixa de ser passivo e passa a interagir diretamente com a programação da televisão [11].

Ao utilizarmos a noção de dependência de contexto nas aplicações interativas da TV digital, possibilitamos que tais aplicações sejam adaptadas dinamicamente de acordo com o contexto que cerca o telespectador. Embora contextos possam ser usados para enriquecer serviços de TV digital fixos, o seu maior interesse é no cenário da TV digital móvel [11]. Em [11], são mostrados diversos cenários onde aplicações desta natureza poderiam se beneficiar de informações de contexto. Entre eles, está a transmissão de informações úteis a telespectadores de acordo com a sua localização e a gravação automática de conteúdo de acordo com programas que o telespectador costuma assistir ou comprar.

Num ambiente como este, que alia a interatividade com a dependência de contexto, o CxFramework é utilizado para dar suporte ao desenvolvimento destas aplicações e daquelas que exijam reflexividade, disponibilizando toda a estrutura de obtenção e processamento de informações de contexto. A Fig. 3 mostra a integração do CxFramework com aplicações da TVDI (TV Digital Interativa).

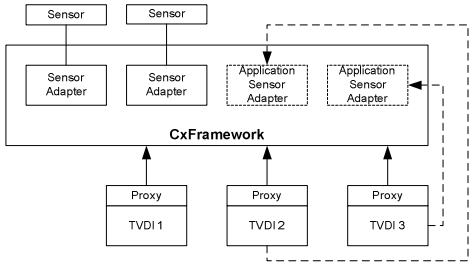

Fig. 3. Integração entre o CxFramework e a TV digital.

Cada aparelho de TV pode ser considerado uma aplicação dependente de contexto conectada ao CxFramework. Isto porque a televisão tem aplicações interativas que se adaptam de acordo com o contexto. O CxFramework recebe informações de contexto de baixo nível de sensores externos e processa estas informações, transformando-as em informações de alto nível, como por exemplo a transformação de coordenadas de latitude e longitude obtidas por um GPS em informação da cidade onde esta coordenada está localizada. As aplicações podem buscar ativamente por estas informações de contexto ou então registrarem o interesse de serem notificadas quando determinado evento ocorrer. Como exemplo, um sensor externo poderia ser um serviço de previsão do tempo. O CxFramework poderia ler as informações do tempo de diversas cidades e disponibilizá-las às aplicações conectadas à infra-estrutura (neste caso, os aparelhos de TV). Além disso, preferências do espectador expressas em termos de ontologias podem ser utilizadas de forma a não somente "filtrar" os eventos de seu interesse particular, mas também inferir novas classes de eventos de potencial interesse para o espectador, ainda que não explicitadas formalmente por ele. O aspecto reflexivo do CxFramework possibilita retro-alimentar cada aparelho de TV com informações de usuários individuais combinadas em eventos de alto nível por meio de regras de inferência. Estes eventos podem incluir a identificação de canais/programas mais assistidos (eventualmente classificados por região, gênero ou qualquer outra categoria), a taxa de acessos a serviços, se o telespectador muda de canal durante os comerciais e assim por diante. Evidentemente, este tipo de informação é de interesse não apenas de espectadores, mas de provedores de serviços e produtos. Na prática, a reflexividade é alcançada através dos *Application Sensor Adapters* pelas aplicações da TV, que são utilizados para alterar o contexto através de envio de informações ao CxFramework a respeito do que o telespectador está assistindo em determinado momento.

V. Trabalhos Relacionados

O *Context Toolkit* [12] foi uma das primeiras infra-estruturas criadas com a finalidade de suportar aplicações dependentes de contexto. Ela introduziu o conceito de encapsulamento da rede de sensores e a separação da aquisição da informação de contexto da sua representação (através dos *widgets*, que funcionam como *wrappers* de contexto). Uma desvantagem do *Context Toolkit* é não possuir um mecanismo centralizado para buscar informações de contexto: é necessário que a aplicação tenha referência a muitos objetos, o que causa grande dependência e limita a escalabilidade.

O SOCAM (*Service Context-Aware Middleware*) [13] é uma infra-estrutura para aplicações dependentes de contexto focada no uso de ontologias para modelar informações de contexto. Quanto à notificação, as aplicações dependentes de contexto são notificadas através de um método de *callback*. Este mecanismo faz com que a infra-estrutura e a aplicação fiquem fortemente acopladas, o que é uma desvantagem em um cenário distribuído. Além disso, o SOCAM é implementado em Java, utilizando RMI. Como RMI é uma tecnologia atrelada ao Java, não é possível fazer com que os componentes do SOCAM se comuniquem com outros componentes externos e até aplicações que não sejam escritos na linguagem Java.

A MoCA (*Mobile Collaboration Architecture*) [14] é uma arquitetura focada em usuários móveis. A notificação às aplicações é baseada nos protocolos TCP e UDP, o que acopla as aplicações à infra-estrutura. Além disso, a quantidade de contexto que a infra-estrutura é capaz de gerenciar é limitada (resume-se a algumas informações vindas do dispositivo móvel) e não pode ser amplamente estendida. Desta forma, o mecanismo de inferência também é limitado, além de ser atrelado ao código da infra-estrutura.

O ContexTV [11] é uma arquitetura que dá suporte a aplicações dependentes de contexto no cenário de convergência entre a TV digital e a telefonia móvel. Esta arquitetura permite que aplicações desenvolvidas para a TV digital móvel possam

ter seu funcionamento influenciado pelo contexto que cerca o usuário. No entanto, o ContexTV possui limitações com relação à modelagem de informações de contexto e inferências de informações de contexto de alto nível.

Diferentemente das infra-estruturas pesquisadas, o CxFramework utiliza comunicação baseada em eventos, o que proporciona o desacoplamento da infra-estrutura e das aplicações. Em ambientes distribuídos, onde nem sempre todos os componentes estão disponíveis, o desacoplamento é de fundamental importância. Outra característica inovadora do CxFramework, não presente em nenhuma infra-estrutura pesquisada, é o suporte à reflexividade. A reflexividade possibilita que o CxFramework seja capaz de ser influenciado pelo contexto que ele representa. Isto representa um grande benefício e confere ao CxFramework a capacidade de representar cenários que outras infra-estruturas não poderiam representar. Além disso, o suporte à reflexividade permite a utilização de federação de contextos, onde contextos diferentes (representados por instâncias de CxFramework diferentes) podem se comunicar.

VI. Conclusão e Trabalhos Futuros

Este trabalho propõe a arquitetura do CxFramework, uma infra-estrutura reflexiva para dar suporte a aplicações dependentes de contexto. A existência de uma infra-estrutura desta natureza significa que grande parte da complexidade da obtenção e do processamento de informações de contexto é transferida para a infra-estrutura, facilitando assim o desenvolvimento das aplicações. Outra vantagem importante é que diversas aplicações podem compartilhar a mesma infra-estrutura, o que permite reuso do código. Como aplicações dependentes de contexto freqüentemente utilizam dispositivos heterogêneos (em termos de *hardware*, sistemas operacionais, etc.) e devido à gama de serviços oferecidos, CORBA foi escolhida como plataforma para a implementação do protótipo do CxFramework. Note-se, no entanto, que os serviços fundamentais definidos podem ser aplicados a outras plataformas com maior ou menor grau de dificuldade, dependendo da infra-estrutura de comunicação escolhida. Em particular, o custo da transposição do CxFramework para o *middleware* Ginga [15] depende apenas da incorporação nesta plataforma de serviços similares ao de notificação e de *trading* do CORBA. A capacidade do CxFramework de ser reflexivo permite que as aplicações conectadas a ele possam alterar o contexto dos seus ambientes, fazendo com que a própria infra-estrutura seja dependente de contexto. Isto também possibilita a construção de federações de contextos onde contextos diferentes interferem uns nos outros.

Mesmo sendo uma infra-estrutura completa para suportar aplicações dependentes de contexto, o CxFramework abre várias possibilidades de pesquisa visando sua melhoria. Atualmente, é de responsabilidade do administrador informar (via arquivo de configuração) quais são os contextos suportados pela infra-estrutura. Esta lista de contextos é utilizada pelas aplicações que desejam obter todos os tipos de notificações possíveis para poderem escolher nas quais se registrarem. Delegar esta responsabilidade ao administrador é uma tarefa que poderia ser evitada se houvesse um mecanismo automático para descobrir os eventos de contexto possíveis de serem gerados pela infra-estrutura. Um mecanismo desta natureza deve levar em consideração a ontologia e as regras de inferência, a fim de poder concluir quais são estes eventos.

Para melhorar a qualidade da informação de contexto, é possível implementar os conceitos de qualidade de contexto (QoC) [16] na infra-estrutura. Além disso, a incorporação de privacidade, segurança e restrição às informações de contexto ao CxFramework podem ser tópicos de grande importância para alguns cenários, além do suporte à tempo real.

Para adequar o CxFramework ao ambiente da TV digital, é necessário implementar o CxFramework sobre o *middleware* Ginga, que no nosso protótipo utiliza o *middleware* CORBA. Além disso, é necessário verificar se a solução final atende o requisito de escalabilidade, necessário num ambiente como este.

Referências

[1] A. K. Dey e G. D. Abowd, *Towards a Better Understanding of Context and Context-Awareness*. Conf. Human Factors in Comp. Sys, 2000.

[2] S. Powers, *Practical RDF*. O' Reilly, 2003.

[3] M. Baldauf, S. Dustdar e F. Rosenberg, *A Survey on Context-Aware Systems*. Information Systems Institute, Vienna University of Technology, Austria, 2007.

[4] Object Management Group. Disponível em: http://www.omg.org. Acessado em: Março, 2007.

[5] J. Siegel, *CORBA 3 Fundamentals and Programming*. Wiley & Sons Inc, 2a edição, 2000.

[6] M. Henning e S. Vinoski, *Advanced CORBA Programming with C++*. Addison Wesley, 1999.

[7] OpenORB. Disponível em: http://openorb.sourceforge.net. Acessado em: Julho, 2009.

[8] M. Smith, C. Welty e D. McGuinness, *Web Ontology Language (OWL) Guide*. http://www.w3.org/TR/owl-guide, 2004.

[9] Protégé. Disponível em: http://protege.stanford.edu /overview/protege-owl.html. Acessado em: Julho, 2009.

[10] Jena: *A Semantic Web Framework for Java*. Disponível em: http://jena.sourceforge.net. Acessado em: Julho, 2009.

[11] Fernando C. A. Neto e Carlos A. G. Ferraz, "Uma arquitetura para suporte ao desenvolvimento de aplicações sensíveis a contexto em cenário de convergência". II Wksp de TV Digital do 24º SBRC, 2006.

[12] A. K. Dey e G. D. Abowd, *A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications*. Human-Computer Interaction, 2001.

[13] T. Gu, H. Pung e D. Zhang, *A service-oriented middleware for building context-aware services*. Journal of Network and Computer Applications, 2005.

[14] J. Viterbo, et al, "MoCA: Uma Arquitetura para o Desenvolvimento de Aplicações Sensíveis ao Contexto para Dispositivos Móveis". 24º SBRC, 2006.

[15] Ginga. Disponível em: http://www.ginga.org.br. Acessado em: Julho, 2009.

[16] T. Buchholz, A. Küpper e M. Schiffers, *Quality of Context: What It Is and Why We Need It*. Proc. Wksp. HP OpenView Univ. Assn, 2003.