Proposta de uma infra-estrutura de suporte para serviços de contexto e descoberta de recursos

Leonardo Cardoso[1], Alexandre Sztajnberg[2], Orlando Loques[1]

[1]Instituto de Computação - Universidade Federal Fluminense
[2]DICC/IME e PEL/FEN - Universidade do Estado do Rio de Janeiro

{lcardoso, loques}@ic.uff.br, alexszt@ime.uerj.br

Resumo. *Aplicações ubíquas e pervasivas são sensíveis ao contexto dos recursos utilizados, seja em relação à disponibilidade, ou em relação à qualidade dos mesmos. Tais aplicações podem se utilizar de mecanismos para descobrir recursos que atendam aos seus requisitos não-funcionais e para monitorar a qualidade destes recursos durante a execução. Propomos dois serviços, que devem idealmente integrar a infra-estrutura de suporte para as aplicações mencionadas: um Serviço de Contexto, que provê acesso às informações de contexto; e um Serviço de Descoberta, que permite a descoberta dinâmica de recursos considerando restrições de contexto a serem satisfeitas.*

Abstract. *Ubiquitous and pervasive applications are aware of the context of the used resources, regarding their availability and quality. Such class of application can benefit from mechanisms to discover resources that meet their non-functional requirements and mechanisms to monitor the quality of those resources. We present a proposal for two services that ideally have to be integrated into the managing infrastructure of the mentioned application: a Context Service that provides access to context information; and a Discovery Service, which allows the dynamic discovery of resources, considering context constraints to be satisfied.*

1 Introdução

Sistemas ubíquos e pervasivos [Sah03] são compostos por diversos elementos ou recursos. Qualquer entidade necessária para a execução do sistema, como por exemplo, um componente de software ou dispositivo de hardware, pode ser considerada um recurso. Cada tipo de recurso possui informações específicas de contexto. Por exemplo, informações de contexto de um computador podem estar relacionadas a sua capacidade de processamento ou de armazenamento, em determinado momento. No caso de aplicações pervasivas, pode-se ter uma grande variedade de tipos de recurso, desde uma sala, uma câmera, um sensor de alarme, ou pessoas presentes em determinada sala. Ao considerar uma pessoa como recurso do sistema, suas informações de contexto podem ser a sua localização atual e suas credenciais de segurança.

Informações de contexto normalmente apresentam valores dinâmicos e grande variabilidade. Na maioria dos casos, é preciso monitorar estes valores para, eventualmente, fazer adaptações de acordo com o estado atual dos recursos. Por exemplo, no caso de um sistema de *streaming* de vídeo, seria possível reduzir a qualidade da mídia transmitida, quando a largura de banda monitorada torna-se limitada, mantendo a aplicação em funcionamento com qualidade menor, mas aceitável.

Aplicações pervasivas necessitam do suporte de (i) uma instrumentação para coletar informações do estado de seu ambiente de operação, que chamamos informações de contexto, e (ii) uma infra-estrutura para a descoberta dinâmica e localização de recursos. Seria inviável desenvolver sistemas dinâmicos, auto-adaptáveis e sensíveis ao contexto somente com recursos ligados estaticamente. Neste sentido, seria conveniente dispor de serviços de suporte para descobrir e ligar recursos dinamicamente durante a implantação ou mesmo durante a execução destes sistemas. Idealmente, tal serviço deveria suportar a especificação de restrições de contexto dos recursos. Por exemplo, ao fazer uso de determinado componente uma aplicação poderia ter restrições específicas, como a localização de uma pessoa ou a temperatura de uma sala.

Sistemas de middleware e *frameworks* de suporte ao desenvolvimento de aplicações distribuídas atuais provêm mecanismos para a especificação de restrições de qualidade (QoS) e contexto, e gerenciam a adaptação do sistema de acordo com a variação na disponibilidade dos recursos monitorados. No entanto, o suporte à descoberta de recursos é limitado, normalmente integrado empiricamente e tem implementação *ad hoc*. Outro problema se refere aos mecanismos de monitoração que usualmente são implementados como *hot spots* do sistema, não permitindo sua reutilização, tornando a aplicação dependente de mecanismos usados no baixo nível.

Com estas questões em mente, apresentamos neste trabalho (i) um Serviço de Contexto de alto nível e independente de tecnologia, que abstrai e coordena os detalhes de comunicação com os *Agentes de Recurso*, facilitando as atividades de monitoramento; (ii) um Serviço de Descoberta que suporta a descoberta de recursos levando em consideração restrições de contexto a serem satisfeitas. Os serviços propostos são integrados ao CR-RIO [Cor05], estendendo sua funcionalidade, tornando possível desenvolver aplicações ubíquas e pervasivas através de sua arquitetura de software e contratos arquiteturais. Para exemplificar o funcionamento dos serviços propostos, em conjunto com o CR-RIO, utilizamos uma aplicação pervasiva.

A Seção 2 apresenta os conceitos utilizados. A Seção 3 apresenta a proposta dos serviços de Contexto e Descoberta. Discutimos a integração dos serviços propostos ao *framework* CR-RIO na Seção 4, onde mostramos como especificar os contratos em termos de recursos virtuais que devem ser descobertos dinamicamente. Na Seção 5, apresentamos uma aplicação de vídeo sob demanda em ambiente pervasivo. A Seção 6 conclui o trabalho.

2 Conceitos Básicos

Nesta seção apresentamos os conceitos e termos básicos utilizados, particularmente, a conceituação de *contexto* e de *aplicação sensível ao contexto*. Maiores detalhes e uma discussão sobre as principais referências podem ser obtidos em [Szt05 e Car06B].

Contexto. No decorrer do texto utilizamos as definições de contexto e aplicação sensível ao contexto apresentadas em [Dey00]. Assim, contexto significa: *"qualquer informação que pode ser utilizada para caracterizar o estado de uma entidade. Consideram-se entidades uma pessoa, lugar ou objeto que seja relevante para a interação entre o usuário e uma aplicação, incluindo estes próprios"*. De forma similar, uma aplicação sensível ao contexto pode ser definida como: *"uma aplicação que utiliza as informações de contexto de suas entidades para disponibilizar informações ou serviços adequados para os usuários"*.

Representação de recursos e atributos de contexto. A representação dos recursos e seus atributos está ligada diretamente ao monitoramento do contexto e a descoberta destes recursos. É necessário que o tipo, atributos e características funcionais do recurso tenham uma representação reconhecida pelos elementos que usam tais informações. As propostas atuais possuem foco específico, restringindo o tipo de informação associada a cada recurso. A iniciativa do DMTF denominada *Common Information Model* (CIM) [DMTF99] apresenta uma solução geral para a representação de recursos e poderá ser adotada por sistemas de gerenciamento de aplicações dinâmicas no futuro.

Sistemas de Monitoração. A atividade de monitorar o contexto e os recursos utilizados é fundamental para a operação de aplicações pervasivas. Ferramentas disponíveis para a monitoração podem ser utilizadas diretamente pela aplicação para a coleta de informações de contexto, dentre eles o NWS [Wol99] e Ganglia [Mas04]. Via de regra, tais ferramentas permitem monitorar recursos como CPU, memória e rede. Algumas ferramentas são direcionadas a domínios específicos como o GMA [Tie02], orientada para a monitoração para grades. Mesmo havendo um número razoável de opções, estas possuem configuração e interface de consulta distintas, tornando a aplicação dependente de uma tecnologia particular, o que leva ao conceito de *Agentes de Recurso*.

Agente de Recurso. O Agente de Recurso proposto em nosso trabalho é um elemento, cuja função é permitir que as aplicações tenham acesso a informações de contexto, escondendo os detalhes de baixo nível usados no monitoramento e coleta de dados brutos. Isso é possível através do provimento de uma interface uniforme. Assim, a aplicação não precisa se preocupar diretamente com o tipo de mecanismo de monitoramento ou sensor utilizado na coleta dos dados para cada tipo de recurso.

Descoberta de Recursos. Embora seja possível utilizar recursos previamente (estaticamente) conhecidos, diretamente (sem envolver um catálogo), é fundamental permitir que os recursos sejam descobertos e ligados dinamicamente. As aplicações de interesse operam em ambientes que se modificam constantemente e, para isso, precisam de um serviço de descoberta que localize dinamicamente instâncias de componentes e recursos que satisfaçam as suas necessidades [Kin02]. Existem atualmente serviços que suportam a descoberta de recursos, em termos de topologia de rede ou localização [Zhu05], porém, não foram projetados para considerar especificamente informações contextuais. Cada serviço geralmente possui um domínio de aplicação e uma linguagem de consulta específica. Com o objetivo de cobrir estas lacunas, novas arquiteturas têm sido projetadas. Uma proposta é o ONS [Lee06], uma arquitetura para sistemas de nomeação ubíquos e pervasivos, que provê às aplicações sensíveis ao contexto, identificação transparente de contexto e religação de serviços, independente de mudanças no seu contexto. Outra proposta, Q-Cad [Cap05], permite as aplicações pervasivas descobrirem e selecionarem recursos considerando o contexto corrente de execução e os requisitos de QoS. Os exemplos citados oferecem serviços, mecanismos de localização e representação de recursos distintos.

Gerenciamento de Aplicações Sensíveis ao Contexto. A execução e gerência de aplicações auto-adaptáveis, ubíquas e pervasivas requer um infra-estrutura de suporte que integre os elementos e características discutidas anteriormente. Esta infra-estrutura deve permitir a especificação das restrições de qualidade desejadas para a aplicação e políticas de adaptação para mantê-la em operação, retirando esta responsabilidade da própria aplicação. Sistemas de suporte para a configuração dinâmica para implantar ou adaptar a aplicação também devem ser providos. Além disso, é preciso que essa infra-

estrutura facilite a integração da monitoração dos recursos utilizados e a descoberta de novos recursos [Car06A]. Propostas recentes oferecem serviços mais convenientes para aplicações ubíquas e pervasivas. Por exemplo, o *Rainbow* [Gar04] permite a especificação dos elementos a serem monitorados e estratégias de adaptação para garantir os requisitos de qualidade de uma aplicação. O monitoramento é feito por *probes* associadas a propriedades do modelo arquitetural da aplicação. A descoberta de recursos é feita a partir de operadores usados nas estratégias de adaptação. Na área da computação ubíqua tem-se o Gaia [Rom02], uma infra-estrutura que trata um "espaço ativo" e seus dispositivos, de forma análoga a um sistema operacional tradicional, fornecendo serviços básicos, incluindo eventos, presença de entidades (dispositivos, usuários e serviços), notificação de contexto e sistema de nomes.

O *framework* CR-RIO (*Contractual Reflective-Reconfigurable Interconnectable Objects*) consiste de uma infra-estrutura de suporte a especificação e gerenciamento de contratos não-funcionais [Loq04]. Um contrato permite especificar serviços diferenciados e associá-los a requisitos não-funcionais que expressam níveis desejados de qualidade. A monitoração desses requisitos é feita por Agentes de Recurso. O CR-RIO tenta adaptar dinamicamente o sistema para um serviço compatível com os níveis de recursos atuais quando a qualidade atual não pode ser mais mantida. Detalhes sobre o CR-RIO e de sua integração com o Serviço de Descoberta serão discutidos na Seção 4.

3 Proposta dos Serviços de Contexto e Descoberta de Recursos

A motivação para a proposta dos serviços de Contexto e de Descoberta é facilitar o desenvolvimento de aplicações pervasivas, incluindo informações sobre requisitos e restrições de contexto nestes serviços. Tivemos em mente facilitar a integração destes serviços em infra-estruturas de suporte a aplicações adaptativas.

3.1 Representação dos Recursos

Em nossa abordagem, cada recurso é associado a um descritor que especifica seu tipo e propriedades específicas. Para cada propriedade registram-se o nome, o tipo de dado (eg. numérico, *string*) e a unidade de medida (eg. *%, MB, Mhz*). Utilizamos XML para tal representação[1]. A Figura 1 apresenta um exemplo de representação de um recurso do tipo *Processing*. Nas linhas 5-8 são informadas as propriedades deste recurso com os seus respectivos tipos e unidades de medida. Por exemplo, a linha 5 informa que este recurso, tem o atributo *CPUClock* do tipo *float* e tem unidade de medida em *Mhz*.

```
1 <Resource>
2   <Type>"Processing"</Type>
3   <Description>"Processing Resource"</Description>
4   <Attributes>
5      <Attribute Name="CPUClock"    Type="float" Units="Mhz" />
6      <Attribute Name="MemFree"     Type="float" Units="MB"  />
7      <Attribute Name="CPUIdle"     Type="float" Units="%"   />
8      <Attribute Name="OSName"      Type="string" Description="OS Name" />
9    </Attributes>
10 </Resource>
```

Figura 1. Representação XML do recurso *Processing*

A representação dos recursos serve de base para o Serviço de Contexto, que poderá obter as informações e consultar os valores das propriedades de cada tipo de

[1] Os *schemas* XML, base para as representações, consultas e respostas são definidos em [Car2006B].

recurso. De forma similar, o Serviço de Descoberta também utiliza, via Serviço de Contexto, as informações armazenadas de acordo com essa representação para validar os registros de recursos e as consultas submetidas. As representações dos recursos devem ser armazenadas em um *Diretório de Recursos.* Ao adicionar um novo tipo de recurso, é necessário, primeiro, registrar e armazenar sua descrição neste *Diretório*. Na Figura 2 temos o registro do recurso de acordo com a descrição anterior (*Processing*, linha 2). O atributo da linha 4 é estático e não varia para este recurso: a propriedade *CPUClock* é *1800 MHz*. As propriedades de consumo de CPU e memória (linhas 5-6) são dinâmicas e devem ser obtidas, sob demanda, através da uma consulta ao Serviço de Contexto. O elemento *URI* contém o identificador da localização do recurso, que neste caso é um endereço IP.

```
1  <ResourceRegister>
2    <Type> Processing </Type>
3    <Attributes>
4      <Attribute Name="CPUClock"    Val="1800MHZ" />
5      <Attribute Name="MemFree"     AttrType="Dynamic"   />
6      <Attribute Name="CPUIdle"     AttrType="Dynamic"   />
7    </Attributes>
8    <URI> 192.168.1.102 </URI>
9  </ResourceRegister>
```

Figura 2. Representação XML de uma instância do recurso *Processing*

3.2 Serviço de Contexto

O Serviço de Contexto é responsável por disponibilizar informações de contexto e esconder os detalhes de baixo nível usados na comunicação com os (vários) Agentes de Recurso. A proposta inclui uma interface de acesso de alto nível. Desta forma, a aplicação preocupa-se somente com os dados necessários e não como eles são obtidos.

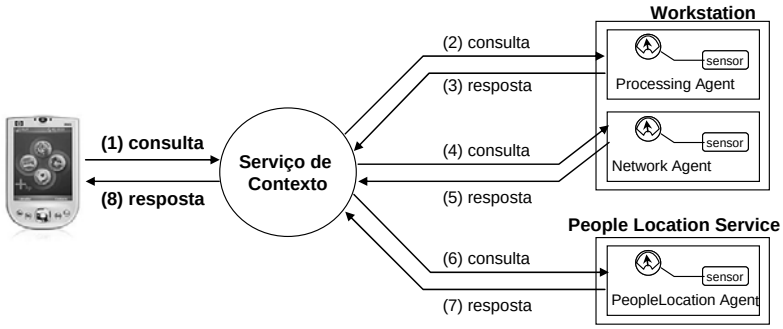

Figura 3. Arquitetura do Serviço de Contexto

Ao receber uma consulta (Figura 3), o Serviço de Contexto verifica quais propriedades de contexto são requisitadas de cada um dos recursos. Uma consulta pode conter um ou mais recursos-alvo, possibilitando monitorar conjuntos de recursos. Após a interpretação da consulta, o Serviço de Contexto (e não a aplicação) comunica-se com cada Agente de Recurso remoto envolvido para obter as informações de contexto individuais. Após coletar as informações, o serviço consolida as mesmas e repassa o resultado ao solicitante. É possível também instruir nos parâmetros da consulta que resultados parciais sejam retornados assincronamente assim que estiverem disponíveis.

3.2.1 Consulta de Contexto

A Figura 4 mostra um exemplo de consulta ao Serviço de Contexto. O elemento *Synchronized* (linha 2), indica que a aplicação quer os resultados sincronizados. *Target* (linha 3) informa o endereço de um dos recursos-alvo da consulta através da

propriedade *URI*. Na linha 4 especifica-se que os próximos atributos são do recurso do tipo *Processing*. Nas linhas 5-6 são especificados os atributos de contexto que a aplicação quer obter: *CPUIdle e MemFree*. O intervalo mínimo com que os valores dos atributos devem ter sido coletados pode ser especificado. No exemplo a cada 1 segundo (linha 7). Note-se que a consulta informa um segundo tipo de recurso (linha 10). Na linha 12 o operador lógico que informa que a aplicação somente deseja receber os resultados, caso o atributo *userId* tenha valor igual a 712.

```
1 <ContextQuery>
2   <synchronized>true</synchronized>
3   <Target URI="wokstation.ic.uff.br">
4      <Attributes From="Processing">
5        <Attribute Name="CPUIdle" />
6        <Attribute Name="MemFree" />
7        <CollectInterval Min="1" units="sec"/>
8      </Attributes>
9   </Target>
10  <Target URI="PL.ic.uff.br:7895">
11     <Attributes From="UserLocation">
12        <Attribute Name="userId" op="==" Val="712" />
13        <Attribute Name="currentRoom" />
14     </Attributes>
15  </Target>
16 </ContextQuery>
```

Figura 4. Representação XML de uma consulta ao Serviço de Contexto

A Figura 5 apresenta a resposta gerada com informações sobre os dois recursos (linhas 2-8 e 9-14). *ResourceInfo* informa que os dados foram coletados do recurso localizado em *workstation.ic.uff.br* com endereço IP 192.168.1.1, às 12:10 a.m com intervalo de 1 segundo. Nas linhas de 5-6 temos os atributos com as respectivas propriedades para este recurso. Por exemplo, na linha 5, um atributo representa a informação de que a CPU livre (*Name="CPUIdle"*) do recurso é de 68% (propriedades *Val=68.0* e *Units="%"*). As demais informações seguem a mesma estrutura.

```
1 <ContextResponse>
2   <ResourceInfo URI="wokstation.ic.uff.br" IP="192.168.1.1" Timestamp="12:10 a.m"
3                                                                  Interval="1" >
4     <Attributes>
5       <Attribute Name="CPUIdle" Val="68.0"  Type="float" Units="%"/>
6       <Attribute Name="MemFree" Val="338.8" Type="float" Units="MB"/>
7     </Attributes>
8   </ResourceInfo>
9   <ResourceInfo URI ="PL.ic.uff.br" IP="192.168.1.12" Timestamp="12:12 a.m">
10     <Attributes>
11        <Attribute Name="userId"      Val="712" Type="int" Units=""/>
12        <Attribute Name="currentRoom" Val="12"  Type="int" Units=""/>
13     </Attributes>
14   </ResourceInfo >
15 </ContextResponse>
```

Figura 5. Representação XML de uma respostado Serviço de Contexto

3.3 Serviço de Descoberta

A arquitetura do Serviço de Descoberta (Figura 6) é composta por três elementos: (i) Gerenciador; (ii) Diretório de Recursos; e (iii) Serviço de Contexto. Para descobrir um novo recurso, uma aplicação submete uma consulta ao Serviço de Descoberta e recebe como resposta uma lista de recursos. Para isso ela informa a classe de recurso desejada e também as restrições de contexto que este deve satisfazer. Por exemplo, a aplicação precisa acessar um servidor Web que tenha um atraso de rede fim-a-fim menor que 50 ms. A consulta da aplicação então é repassada ao Gerenciador que a interpreta e obtêm

do Diretório de Recursos todas as instâncias de recursos da classe requerida pela aplicação. Nos casos onde são informadas restrições de contexto, o Gerenciador deve consultar o Serviço de Contexto obtendo os valores das propriedades de contexto de todas as instâncias recebidas anteriormente do Diretório de Recursos. Com essas informações, ele poderá classificar os recursos encontrados e filtrar a resposta, retornando somente as instâncias que satisfazem a consulta. Depois de descartar os recursos não satisfatórios, o Gerenciador retorna uma lista de recursos para a aplicação.

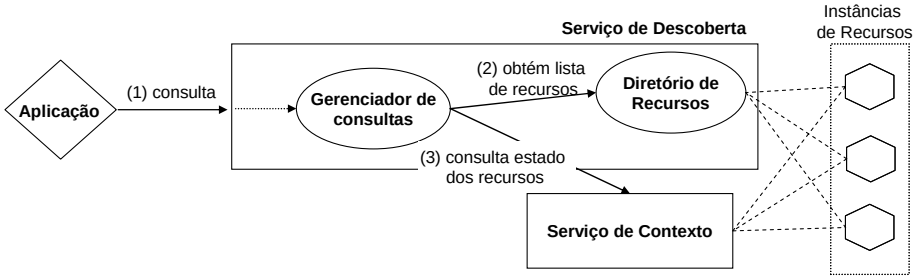

Figura 6. Arquitetura do Serviço de Descoberta

3.3.1 Consulta de Descoberta

A Figura 7 mostra uma de consulta de descoberta, em que o usuário quer localizar um recurso do tipo *VideoServer* (linha 1 - elemento *ResourceQuery*, propriedade *type*) e obter no máximo 5 resultados (linha 2). As linhas 3-10 indicam as restrições de contexto que devem ser satisfeitas pelos recursos a serem encontrados. Nas linhas 3-6 são indicadas as restrições do tipo *Processing*, informando que a CPU deve ter um *clock* (*CPUClock*) de, no mínimo, 1.8Ghz. A linha 5 especifica que o recurso deve ter, no máximo, *50%* de uso de CPU (*CPUIDle*). Nas linhas 7-10 são indicadas as restrições do tipo *Transport* de comunicação do nó onde a aplicação executa até o recurso encontrado, informando que a largura de banda (*Bandwidth*) deve ser maior ou igual a 256*kbps* e o atraso (*Delay*) menor ou igual a *50ms*.

```
1  <ResourceQuery type=”VideoServer”>
2     <MaxResults>5</MaxResults>
3     <Constraints From=”Processing”>
4        <Attribute Name=”CPUClock”  op=”>=” Val=”1800MHZ” />
5        <Attribute Name=”CPUIDle”   op=”>=” Val=”50” />
6     </Constraints>
7     <Constraints From=”Transport”>
8        <Attribute Name=”Bandwidth” op=”>=” Val=”256” />
9        <Attribute Name=”Delay”     op=”<=” Val=”50” />
10    </Constraints>
11 </ResourceQuery >
```

Figura 7. Representação XML de uma consulta ao Serviço de Descoberta

Uma resposta à consulta anterior conteria uma lista de recursos, incluindo, para cada um, o tipo (*VideoServer*) e suas respectivas propriedades (*Processing* e *Transport*).

4 Integração com *Framework* CR-RIO

Esta seção aborda como o Serviço de Contexto e o Serviço de Descoberta são inseridos na arquitetura do *framework* CR-RIO, desenvolvido em nosso grupo, que oferece o suporte para aplicações pervasivas. O CR-RIO é centrado em um modelo arquitetural e utiliza uma linguagem de descrição de contratos (CBabel) para expressar os requisitos não-funcionais das aplicações. Com base nestes elementos, desenvolveu-se uma infra-

estrutura de suporte para: (i) interpretar a especificação dos contratos e armazená-la como meta-informação, associada à aplicação; (ii) prover mecanismos de reflexão e adaptação dinâmica, que permitem adaptar a configuração da aplicação (incluindo os seus elementos de suporte), visando atender as exigências de contratos; e (iii) prover elementos para impor, monitorar e manter os contratos associados à aplicação.

4.1 Ligação e Seleção Dinâmica de Recursos baseadas em Contexto

Quando os recursos a serem utilizados na aplicação são conhecidos com antecedência, a ligação entre a descrição da arquitetura e os artefatos de software pode ser feita de forma estática. Neste caso, o projetista especifica, em tempo de projeto, quais recursos serão usados pela aplicação. A linha (1) descreve que o módulo *srvVoD* será ligado à *plasmaDisp*, uma instância específica de um recurso da classe *Display*, se os perfis *plsmProf* e *commProf* (que restringem as características do *display* de plasma e do conector de comunicação, respectivamente) forem válidos. Se estes perfis não foram válidos, a ligação não pode ser feita e o serviço não pode ser estabelecido.

```
(1)   link srvVoD to plasmaDisp by commCon with plsmProf, commProf;
```

No entanto, outras instâncias da classe *Display*, também satisfazendo as restrições dos perfis, poderiam estar disponíveis. Neste caso, seria conveniente que uma referência a um *Display* "virtual" fosse utilizada na descrição e que, em tempo de implantação, o Serviço de Descoberta fosse consultado de forma a se obter um *Display* registrado que atendesse às restrições descritas. A linha (2) apresenta nossa solução para a integração destas funcionalidades no nível do contrato, considerando o mesmo exemplo da linha (1). O módulo *videoServ* (estaticamente selecionado) deve ser ligado ao módulo *dp*, da classe *Display*. A notação *dp = Display* informa que *dp* é uma referência não-ligada ("virtual") e que, em tempo de execução uma instância será dinamicamente obtida através de uma consulta ao Serviço de Descoberta. Ainda, o domínio da procura deve ser restrito a *building* (limitando-se a um Serviço de Descoberta particular). Os perfis *plsmProf* e *commProf* serão usados, também, para limitar os resultados da descoberta de recursos.

```
(2)   link srvVoD to dp = Display at building select(uDispAlone)
           by commCon with plsmProf, commProf;
```

Observe que o Serviço de Descoberta pode retornar uma ou mais referências a recursos que satisfaçam as restrições dos perfis. Entretanto, pode ser conveniente selecionar o recurso mais apto, dentre as referências retornadas. Tal seleção poderia ser realizada por pesos, lista de preferência ou uma função de utilidade. Com este objetivo, introduzimos no contrato uma forma para indicar que um filtro de seleção será utilizado e os critérios de seleção. Na linha (2) a notação *select (uDispAlone)* informa que um filtro de seleção deve ser usado e o "perfil de seleção" a ser considerado é *uDispAlone*.

Um *perfil de seleção* permite especificar, no nível da arquitetura, para cada propriedade de um recurso descoberto, indicativos de preferência tais como: se ela deve ser maximizada ou minimizada; um peso para expressar sua importância relativa e uma ordem de preferência. A Figura 8 apresenta um exemplo de perfil de seleção onde a propriedade *CPUIdle* da categoria (recurso) *Processing* deve ser maximizada e tem peso *2*. Na linha 3, é especificado que a propriedade *MemFree* deve ser maximizada com peso no valor igual a *1*. Com estas informações, pode-se inferir, neste exemplo, que a aplicação tem preferência pelos recursos com maior quantidade de processador livre

(*CPUIdle*). Para as propriedades não numéricas ou enumerações, pode-se estabelecer uma ordem de preferência, como mostrado na aplicação-exemplo (ver Seção 5.2).

```
1  selection {
2    Processing.CPUIdle: maximize weight 2;
3    Processing.MemFree: maximize weight 1;    } hqRef;
```

Figura 8. Exemplo de perfil de seleção

5 Aplicação de Vídeo sob Demanda Pervasiva

A aplicação de vídeo sob demanda pervasiva (VodP) apresenta um cenário onde o usuário transita por várias salas em um prédio, cada uma contendo um conjunto de recursos. Cada usuário possui um Agente que informa sua localização atual. A aplicação é implantada sobre o *framework* CR-RIO, integrado aos serviços propostos. Assim, os requisitos de contexto e as políticas de adaptação são descritas em um contrato.

Um fluxo de vídeo deve ser reproduzido em um dos dispositivos disponíveis na sala em que o usuário se encontra. São estabelecidas ordens de preferência do usuário por certos dispositivos, que variam de acordo com o contexto e levam em conta se o usuário esta ou não em sua sala e também se ele esta acompanhado de outras pessoas. No contrato estas preferências são indicadas por perfis de seleção. Por exemplo, sempre que o usuário estiver em uma sala onde outras pessoas estiverem presentes, o vídeo deve ser reproduzido no *display* pessoal do usuário (Figura 9). Para esta aplicação, são considerados os seguintes dispositivos: Monitor, PDA, TV de Plasma e Projetor.

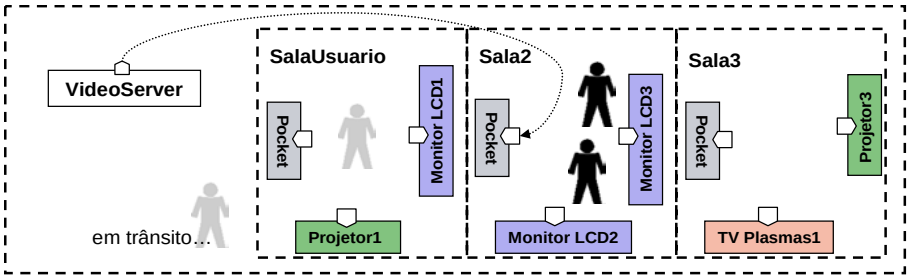

Figura 9. Cenário da aplicação (usuário na sala 2, com outros usuários)

5.1 Definição das Categorias

Identificamos nesta aplicação três tipos de recursos: usuário (*User*), *display* (*Disp)*, e sala (*Room*). Em CR-RIO recursos são descritos em Categorias (Figura 9). A categoria *User* (linhas 5-9) representa um usuário, onde a propriedade *id* é sua identificação, *userRoom* informa a sala de usuário, *currentRoom* a localização corrente, e *envChange* indica se o usuário esta trânsito. Na categoria *Room* as propriedades *nUsers*, e *number* são utilizadas para informar o número de usuários presentes e o número da sala. A categoria *Disp* que contém as propriedades *location*, *type* e *mobile*, usadas para informar a localização do *display*, seus possíveis tipos e se ele é um dispositivo móvel.

```
1  Category Room {                          5  Category User {
2    nUsers: numeric in;                    6    id: numeric in;
3    number: numeric in;                    7    userRoom: numeric in;
4  }                                        8    currentRoom: numeric in;
                                            9    envChange: boolean in;   }
10  Category Disp {
11    location: numeric in;
12    type: enum (TvPlasma, Projetor, PCMonitor, PDA) out;
13    mobile: boolean in;    }
```

Figura 9. Categorias da Aplicação Vod Pervasiva

5.2 Contrato da Aplicação

Na descrição do cenário da aplicação, foram mencionadas situações onde o usuário encontra-se sozinho ou acompanhado em uma sala e também se ele esta ou não em sua própria sala. Surge então a necessidade de especificar como são identificadas tais situações, e o que fazer quando cada uma delas for verdadeira. A Figura 10 apresenta o contrato *VodP* para a aplicação e a Figura 11 mostra os perfis utilizados.

Na linha 1 do contrato *VodP* são recebidas variáveis de contexto no momento de sua instanciação. As variáveis *userId* e *userRoom* são usadas respectivamente para indicar a identificação e número da sala do usuário para o qual o contrato está sendo estabelecido e devem ser repassadas ao mesmo para que seja possível instanciá-lo.

No primeiro serviço, *sOwnRoom*, o usuário está em sua própria sala, situação definida pelo perfil *OwnRoom* (linha 16). O perfil *DispAvail* (linha 40) define que devem ser localizados todos os *Displays* da sala onde o usuário se encontra. O perfil de seleção *uDispOwnRoom* (linha 44) especifica a ordem de preferência dos *Displays*. Este é usado para classificar a lista de *displays* obtida do Serviço de Descoberta e indica que o Monitor do computador é o preferencial; No segundo serviço, *sOtherRoomAlone*, o usuário muda de sala e se encontra sozinho, situação em que o vídeo deve ser mostrado no *Display* disponível e de acordo com a preferência do perfil de seleção *uDispAlone* (linha 45). Neste perfil tem-se a TV de Plasma como preferencial. Os perfis *OtherRoom* e *Alone* definem o contexto em que o usuário não esta em sua sala, mas sozinho.

```
1  contract (userId, userRoom){
2    service {  link srvVoD to dp = Display select (uDispOwnRoom)
3               with OwnRoom, DispAvail;                } sOwnRoom;
4    service {  link srvVoD to dp = Display select (uDispAlone)
5               with OtherRoom, Alone, DispAvail;       } sOtherRoomAlone;
6    service {  link srvVoD to dp = Display select (uDispNotAlone)
7               with OtherRoom, NotAlone, DispAvail;    } sOtherRoomNotAlone;
8    service {  unlink svVod to dp;                      } sInTransit;
9    negotiation {
10     sOwnRoom <-> (sOtherRoomAlone || sOtherRoomNotAlone || sInTransit);
11 } } VodP;
```

Figura 10. Contrato Vod pervasivo

No terceiro serviço *sOtherRoomNotAlone* (linhas 6-7), o usuário está em outra sala, porém na companhia de outras pessoas. O perfil *NotAlone* define esta última situação. Neste caso, por questões de privacidade, o vídeo deve ser transmitido para o PDA do usuário de acordo com o perfil de seleção *uDispNotAlone* (linhas 46); Por fim, o quarto serviço *sInTransit* (linha 8), é ativado quando o usuário está transitando de uma sala para outra. Nestes casos, a transmissão do vídeo deve ser interrompida (*unlink*), até que outro serviço possa ser estabelecido.

```
16 profile {
17   User.id = %userId;
18   User.currentRoom = %userRoom;
19   User.envChange = false;
20 } OwnRoom;

21 profile {
22   User.id = %userId;
23   Room.number = User.currentRoom;
24   Room.nUsers = 1;
25 } Alone;
```

```
30 profile {
31   User.id = %userId;
32   User.currentRoom != %userRoom;
33   User.envChange = false;
34 } OtherRoom;

35 profile {
36   User.id = %userId;
37   Room.number = User.currentRoom;
38   Room.nUsers >= 1;
39 } NotAlone;
```

```
26 profile {                                40 profile {
27    User.id = %userId;                    41   User.id = %userId;
28    User.envChange = true;                42   Disp.location = User.currentRoom;
29 } InTransit;                             43 } DispAvail;
44 selection { Disp.type = (PCMonitor > palm); } uDispOwnRoom;
45 selection { Disp.type = (Plasma > Projetor > monitor > palm); } uDispAlone;
46 selection { Disp.type = (palm);  } uDispNotAlone;
```

Figura 11. Perfis da aplicação Vod pervasivo

5.3 Implantação da Aplicação e Gerência do Contrato

Antes de implantar a aplicação e impor o contrato *VodP*, é preciso implementar (caso não existam) os Agentes de Recurso para as categorias *User*, *Room* e *Displays*, e registrá-los no Diretório de Recursos. Também é necessário especializar os *Contractors* que gerenciarão e validarão os perfis em cada nó. Além disso, as instâncias de recursos (usuários, salas e *displays*), também devem ser criadas e registradas.

Ao implantar a aplicação o Gerenciador de Contratos do CR-RIO interpreta o contrato e identifica na cláusula de negociação (Figura 10, linhas 9-10) que *sOwnRoom* é o serviço preferencial. Neste serviço existe uma declaração *link* para ligar o servidor de vídeo (*srvVoD*, instanciado estaticamente) ao módulo virtual *dp*. A partir daí a seqüência de operações realizada é a descrita na Seção 4.2, parametrizada pelos serviços, perfis e perfis de seleção do contrato *VodP*. Caso o Serviço de Descoberta retorne uma lista contendo mais de uma instância de *Display*, o *Selector* escolhe o melhor deles de acordo com o perfil de seleção *uDispOwnRoom* (estabelece uma ordem de preferência, Figura 11, linha 44).

Depois que o serviço estiver implantado, o conjunto de perfis continua sendo monitorado. Se qualquer *Contractor* detectar que uma das propriedades viola as restrições para o serviço corrente (um perfil está inválido), o Gerenciador de Contratos é notificado, e tenta estabelecer outro serviço, de acordo com a cláusula de negociação. Por exemplo, se o serviço *sOwnRoom* é o corrente e o usuário muda de sala, violando a restrição para esse serviço (Figura 110, linha 3), este não poderá ser mantido. De acordo com a cláusula de negociação, ele tentará estabelecer o serviço *sOtherRoomAlone* iniciando para isso a mesma seqüência de operações.

6 Conclusão

Nosso trabalho propõe dois elementos importantes para o suporte aplicações sensíveis ao contexto: (i) o Serviço de Contexto para coletar informações dos recursos da aplicação e do ambiente de operação, e (ii) o Serviço de Descoberta, para a descoberta dinâmica de recursos, considerando-se restrições de contexto. A integração destes serviços ao CR-RIO resultou em uma infra-estrutura que facilita o desenvolvimento de aplicações pervasivas que se beneficiem destes serviços, através da especificação de contratos não-funcionais em alto nível considerando restrições dinâmicas de contexto.

Atualmente estamos trabalhando em um protótipo estruturado dos serviços, já integrados ao CR-RIO para realizar avaliações de desempenho e escalabilidade. Em conjunto com estas atividades estamos investigando estilos de interface que permitam aos usuários customizar seus próprios contratos (a partir de contratos genéricos) em alto-nível, facilitando a aplicação da proposta. Um outro ponto de pesquisa são as funções de utilidade, incorporadas em nossa proposta através dos perfis de seleção. Estamos explorando alternativas, como às propostas em [Hua03], mas no nosso caso mantendo-se num alto nível de abstração.

Bibliografia

Capra. L, Zachariadis, S., Mascolo. C., "Q-CAD: QoS and Context Aware Discovery Framework for Mobile Systems", Int. Conf. on Pervasive Services (ICPS'05), Santorini, Grécia, Julho, 2005.

Cardoso, L. T., Sztajnberg, A.; Loques, O. G., "Self-adaptive applications using ADL contracts", 2nd. IEEE International Workshop on Self-Managed Networks, Systems & Services, 2006, Dublin. LNCS, 2006. Vol. 3996. p. 87-101.

Cardoso, L. T., "Integração de serviços de monitoração e descoberta de recursos a um suporte para arquiteturas adaptáveis de software", Dissertação de Mestrado, Instituto de Computação, UFF, Novembro, 2006.

Corradi, A., "Um Framework de Suporte a Requisitos Não-Funcionais para Serviços de Nível Alto", Dissertação de Mestrado, Instituto de Computação, UFF, Agosto, 2005.

Dey, A., "Providing Architectural Support for Context-Aware applications", Tese de Doutorado, Georgia Institute of Technology, Novembro 2000.

Distributed Management Task Force, Inc., "Common Information Model Specification", Ver. 2.2, Junho, 1999. www.dmtf.org/standards/cim/cim_spec_v22

Garlan, D., Cheng, S.-W., Huang, et al., "Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure", IEEE Computer, Vol. 37, No. 10, p. 46–54, 2004.

Huang, A.-C., Steenkiste, P., "Network Sensitive Service Discovery", USENIX Symposium on Internet Technologies and Systems, 2003.

Kindberg ,T., Fox, A., "System software for ubiquitous computing", Pervasive Computing Magazine, Vol. 1 , No. 1, pp. 70-81, Janeiro, 2002.

Lee, K., Lee, D., Ko, Y. W., et al., "An Objectified Naming System for Providing Context Transparent to Context-Aware applications", 4th. Wksp. on Software Technologies for Future Embedded and Ubiquitous Systems, 2006.

Loques, O., Sztajnberg, A., Cerqueira, R. C., et al., "A contract-based approach to describe and deploy non-functional adaptations in software architectures". JBCS, Vol. 10, No. 1, pp. 5-18, Julho, 2004.

Massie, M. L, Chun, B. N., Culler, D. E., "The Ganglia Distributed Monitoring System: Design, Implementation, and Experience", Parallel Computing, Vol. 30, No. 7, 2004.

Román, M., Hess, C. K., Cerqueira, R., et al., "Gaia: A Middleware Infrastructure to Enable Active Spaces", IEEE Pervasive Computing, pp. 74-83, Out-Dez, 2002.

Saha, D., Mukherjee, A., "Pervasive computing: A paradigm for the 21st century", IEEE Computer, 36(3): 25–31, 2003.

Sztajnberg, A., Corradi, A. M., Santos, A. L., et al., "Especificação e Suporte de Requisitos Não-Funcionais para Serviços de Nível Alto", Minicursos do 23º. SBRC, pp. 223-279, Fortaleza, CE, 2005.

Tierney, B., Aydt, R., Gunter, D. et al, "A Grid Monitoring Architecture", Tech. Rep. GWD-PERF-16-2, Global Grid Forum, Janeiro, 2002.

Wolski, R.; Spring, T. N.; Hayes, J., "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing", Future Generation Computer Systems, Vol. 15, No. 5-6, pp. 757-768, 1999.

Zhu, F., Mutkaand, M. W., Ni, L. M., "Service Discovery in Pervasive Computing Environments", IEEE Pervasive Computing, Vol. 4, pp. 81-90, 2005.