Universidade Federal Fluminense

LEONARDO XAVIER T. CARDOSO

Integração de serviços de monitoração e descoberta de recursos a um suporte para arquiteturas adaptáveis de software

Niterói, 2006

LEONARDO XAVIER TEIXIERA CARDOSO

Integração de Serviços de Monitoração e Descoberta de Recursos a um Suporte para Arquiteturas Adaptáveis de Software

Dissertação apresentada ao Curso de Pós-Graduação em Computação Aplicada da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Mestre. Área de Concentração: Processamento Distribuído e Paralelo.

Orientador: Prof. Dr. ORLANDO GOMES LOQUES FILHO
Co-Orientador: Prof. Dr. ALEXANDRE SZTAJNBERG

UNIVERSIDADE FEDERAL FLUMINENSE

NITERÓI
2006

Ficha Catalográfica elaborada pela Biblioteca da Escola de Engenharia e Instituto de Computação da UFF

C268 Cardoso, Leonardo Xavier T.
Integração de serviços de monitoração e descoberta de recursos a um suporte para arquiteturas adaptáveis de software / Leonardo Xavier T. Cardoso. – Niterói, RJ : [s.n.], 2006.
122 f.

Orientador: Orlando Gomes Loques Filho.
Dissertação (Mestrado em Computação) - Universidade Federal Fluminense, 2006.

1. Processamento paralelo (Computadores). 2. Computação ubíqua. 3. Processamento distribuído. 4. Arquitetura de software. 5. Computação móvel. I. Título.

CDD 004.35

INTEGRAÇÃO DE SERVIÇOS DE MONITORAÇÃO E DESCOBERTA DE RECURSOS A UM SUPORTE PARA ARQUITETURAS ADAPTÁVEIS DE SOFTWARE

LEONARDO XAVIER TEIXEIRA CARDOSO

Dissertação apresentada ao corpo docente da Coordenação do Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre. Área de Concentração: Processamento Paralelo e Distribuído.

Aprovada em 24/11/2006

BANCA EXAMINADORA

_____

Prof. Orlando Gomes Loques Filho, Ph.D. – Orientador
Universidade Federal Fluminense (UFF)

_____

Prof. Alexandre Sztajnberg, D.Sc. – Co-orientador
Universidade Estadual do Rio de Janeiro (UERJ)

_____

Prof. Antônio Jorge Gomes Abelém, D.Sc.
Universidade Federal do Pará (UFPA)

_____

Prof. Célio V. N. Albuquerque, Ph.D.
Universidade Federal Fluminense (UFF)

_____

Prof. Maria Luiza D Almeida Sanchez, D. Sc
Universidade Federal Fluminense (UFF)

Niterói, RJ – Brasil
24 de novembro de 2006

Agradecimentos

Agradeço primeiramente meus pais que me incentivaram e deram todo o apoio necessário para conclusão deste mestrado.

Agradeço à minha namorada Cristiana, pelo carinho, amor e compreensão.

Ao meu orientador Orlando Loques pela orientação, conselhos e lições de vida aprendidas que me ajudaram a crescer.

Ao meu co-orietandor Alexandre Sztajnberg por toda a paciência, disponibilidade e dedicação. Sua ajuda foi essencial para o desenvolvimento desta dissertação.

Aos companheiros de república André (Mokado), Vander (Vampeta), Marcel (Cabelo), Tiago (Jóvem), Rafael (Pitoko), Sandro pelos momentos de alegria e aprendizado vividos.

À empresa Automâtos pelo tempo cedido.

À FAPERJ por financiar parcialmente este trabalho.

As we enjoy great advantages from inventions of others, we should be glad of an opportunity to serve others by any invention of ours; and this we should do freely and generously.

Benjamin Franklin

Sumário

Lista de Figuras

Lista de Tabelas

Glossário

API	:	Application Programming Interface
CPU	:	Central Processing Unit
DNS	:	Domain Name System
DTD	:	Document Type Definition
FTP	:	File Transfer Protocol
HTTP	:	HyperText Transport Protocol
IDL	:	Interface Definition Language
IETF	:	Internet Engineering Task Force
IP	:	Internet Protocol
JINI	:	Jini Is Not Initials
JVM	:	Java Virtual Machine
LAN	:	Local Area Network
LDAP	:	Lightweight Directory Access Protocol
PDA	:	Personal Digital Assistant
QoS	:	Quality Of Service
RDF	:	Resource Description Framework
RMI	:	Remote Method Invocation
RPC	:	Remote Procedure Call
SDK	:	Software Development Kit
SDL	:	Service Description Language
SLP	:	Service Location Protocol
TCP	:	Transmission Control Protocol
UDP	:	User Datagram Protocol
UPnP	:	Universal Plug and Play
URI	:	Unique Resource Identifier
W3C	:	World Wide Web Consortium
WAN	:	Wide Area Network
WSDL	:	Web Service Description Language
XML	:	Extensible Markup Language
XSD	:	XML Schema Definition

Resumo

Sistemas distribuídos ubíquos, pervasivos e auto-adaptáveis apresentam requisitos específicos em relação aos recursos utilizados sejam eles de infra-estrutura ou componentes da própria aplicação. Além disso, aplicações auto-adaptáveis, ubíquas e pervasivas, por sua vez, são sensíveis ao contexto destes recursos, seja em relação à disponibilidade, ou em relação à qualidade dos mesmos. Este trabalho apresenta a proposta de dois serviços, que devem idealmente integrar a infra-estrutura de suporte para aplicações das classes mencionadas: (i) um Serviço de Contexto que provê acesso às informações de contexto dos recursos; e (ii) um Serviço de Descoberta que permite a descoberta dinâmica de recursos levando em conta restrições de contexto a serem satisfeitas. Para os dois serviços, é proposta uma interface padronizada de alto nível que abstrai os detalhes de baixo nível usados na implementação de cada um deles.

Os Serviços de Contexto e de Descoberta foram integrados ao CR-RIO (*Contractual Reflective - Reconfigurable Interconnectable Objects*), um *framework* para gerência de aplicações adaptativas, que adota Arquiteturas de Software como base conceitual. Para isso, (i) estendemos a linguagem de descrição de contratos do CR-RIO para permitir a especificação de requisitos dinâmicos de contexto de recursos e (ii) mapeamos as extensões propostas na linguagem em ações na infra-estrutura de suporte existente, incorporando os Serviços de Contexto e de Descoberta de forma modular.

A relevância dos Serviços de Contexto e de Descoberta e a adequação da integração destes serviços à infra-estrutura de CR-RIO para o desenvolvimento e gerenciamento de aplicações sensíveis ao contexto são mostradas através de exemplos. Estes exemplos abrangem aplicações distribuídas “legadas” e também as pervasivas, que possuem restrições de contexto em seus serviços e precisam, assim, da descoberta dinâmica e monitoração de recursos. Os resultados obtidos permitiram validar nossa proposta.

Abstract

Ubiquitous, pervasive and self-adaptive distributed systems have specific requirements regarding the resources to be used by the infrastructure or by the components of the applications. Morover, auto-adaptive, ubiquitous and pervasive applications are aware of the context of these resources, regarding their availability or their quality. This work presents a proposal for two services that have to be ideally integrated into the infrastructure on the mentioned class of application: (i) a Context Service, which provides access to context information of the resources; and (ii) a Discovery service, which allows the dynamic discovery of resources, considering resources constraints to be satisfied. A standard high-level interface is proposed for both services, hiding the low-level implementation details.

The Context and Discovery Services were integrated to CR-RIO (Contractual Reflective-Reconfigurable Interconnectable Objects), a framework to manage adaptive applications, which adopts Software Architectures as its conceptual foundation. To this end we (i) extended CR-RIO's contract description language allowing the specification of dynamic requirements regarding the context of the resources and (ii) we mapped the extensions proposed in the language into actions to be performed in the existent support infra-structure, introducing the Context and Discovery Services in a modular fashion.

The relevance of the Context and Discovery Services to the development and management of context-aware applications, and the adequacy of the integration of these services to CR-RIO's infrastructure are shown using examples. These examples comprise "legacy", as well as pervasive, distributed applications, which have context constraints in its offered services. In that way, they also need to dynamically discover and monitor the resources. The achieved results allowed us validate our proposal.

1 Introdução

Avanços recentes da tecnologia de computação móvel, bem como de diversas tecnologias de comunicação sem fio, resultaram em um cenário ideal para o desenvolvimento de ambientes que suportam a criação de aplicações pervasivas. A primeira menção à idéia de aplicações pervasivas foi feita em [Weiser 1991], e descreve um ambiente repleto de dispositivos computacionais e de comunicação, que interagem naturalmente com as pessoas, de tal forma que esses dispositivos passam a fazer parte do próprio ambiente.

Dentro desta visão, surgiu o termo "computação ubíqua", hoje também chamada de "computação pervasiva" [Saha & Mukherjee 2003; Satyanarayanan 2001]. Este tipo de sistema trouxe novos desafios ao desenvolvimento de software. Problemas, como a habilidade de os sistemas operarem em ambientes heterogêneos de *hardware*, tornaram-se ainda mais complicados, na medida em que dispositivos portáteis passaram a fazer parte da dinâmica de execução das aplicações. Estes dispositivos têm restrições tanto de *hardware* (baixa capacidade de processamento e memória) quanto de software (sistemas operacionais com poucos recursos), que devem ser levadas em consideração no projeto do sistema.

Paralelamente ao desenvolvimento de aplicações pervasivas, há também os sistemas autônomos, comumente chamados de sistemas *auto*-* (do inglês, *self*-*). Dentre as principais características de um sistema autônomo, pode-se citar: (i) **auto-configuração**: o sistema deve adaptar-se automaticamente às mudanças na disponibilidade de seus recursos; (ii) **auto-reparação**: o sistema deve recuperar-se de falhas ocorridas em um de seus componentes, detectando e isolando o ponto de falha; (iii) **auto-otimização**: o sistema deve ser capaz de monitorar e ajustar os seus recursos, automaticamente e (iv) **auto-proteção**: os sistemas devem ser capazes de prevenir, detectar, identificar e se proteger de ataques [Ganek & Corbi 2003].

Pode-se notar que tanto as aplicações pervasivas quanto os sistemas autônomos têm alguns requisitos mínimos para possibilitar o seu desenvolvimento. Por exemplo, ambos necessitam de alguns elementos básicos: (i) uma instrumentação para coletar informações a respeito do estado de seu ambiente de operação, que se

pode chamar de informações de contexto, e (ii) uma infra-estrutura para a descoberta dinâmica de recursos. Esses dois elementos devem funcionar em conjunto, provendo o suporte básico a ser utilizado na construção dessas aplicações.

Geralmente, uma aplicação é composta de diversos componentes independentes - os recursos. Qualquer entidade necessária para a execução do sistema, como por exemplo, um componente de software (como um servidor de páginas Internet) ou dispositivo de *hardware* (como um roteador), pode ser considerada um recurso. Cada tipo de recurso, seja ele de software, ou de *hardware*, tem informações específicas de contexto. Por exemplo, as informações de contexto de um computador estão relacionadas à 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, câmera, sistema de alarme, ou pessoas presentes em determinado auditório. Neste caso, quando se diz que uma pessoa é um recurso do sistema, suas informações de contexto podem estar relacionadas à sua localização e também às suas credenciais de segurança dentro de um prédio.

Ao se trabalhar com a disponibilidade dos recursos computacionais envolvidos em uma aplicação, utiliza-se, tradicionalmente, o termo "restrições de qualidade de serviço" (*Quality of Service* – QoS). Neste trabalho, são adotados os termos restrições de contexto e aspectos não-funcionais para trabalhar com restrições de qualquer tipo de recurso.

Uma característica comum das informações de contexto é que elas normalmente têm valores dinâmicos e grande variabilidade. Na maioria dos casos, é preciso monitorar os valores dessas informações 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. Desta maneira, é possível continuar provendo o serviço mesmo que a disponibilidade de rede se deteriore. Seria inviável desenvolver aplicações adaptáveis sem que estas pudessem monitorar o nível de operação de seus recursos. A proposta desse trabalho denomina a entidade responsável pela coleta de informação de contexto de **Agente de Recurso**.

Antes de fazer uso de recursos, um sistema deve ser capaz de localizá-los no ambiente de operação. Seria inviável desenvolver sistemas dinâmicos, auto-adaptáveis e sensíveis ao contexto com a ligação estática dos recursos durante o desenvolvimento. Assim, devem-se utilizar mecanismos que permitam a descoberta e a ligação dinâmica de recursos durante a implantação ou mesmo durante a execução destes sistemas. Por exemplo, na aplicação de videoconferência descrita no Capítulo 5 Seção 5.2, é utilizado um mecanismo de descoberta de recursos para localizar dinamicamente um refletor de comunicação que satisfaça os requisitos mínimos de operação, em termos de largura de banda e latência de rede requeridos pelo cliente.

Idealmente, um serviço de descoberta deveria suportar a especificação de restrições relacionadas ao contexto dos recursos. Por exemplo, aplicações pervasivas podem ter restrições específicas, como a localização de uma pessoa ou a temperatura de uma sala, ao fazer uso de determinados recursos. Isto requer uma tecnologia extensível de descoberta capaz de suportar a especificação das mais diversas restrições de contexto. Com esses requisitos em mente, foi possível identificar três desafios principais para o desenvolvimento de um serviço de descoberta, que são:

- **Nomeação**: um sistema de nomeação dos recursos, dentro de certo contexto de operação, é a base para que estes possam ser localizados. Um exemplo clássico de sistema de nomeação é o *Domain Name System* – DNS – usado na rede mundial de computadores - Internet. Com o DNS, cuja importante característica é o sistema hierárquico de nomes, é possível identificar qualquer recurso através de um nome único e ainda saber o seu domínio.
- **Base de dados**: uma base de dados responsável pelo armazenamento do estado atual do ambiente de operação. Alguns exemplos de informações que ela deve manter: (i) lista de recursos disponíveis; (ii) localização de cada recurso; e (iii) estado conhecido dos recursos.
- **Interface de consulta**: uma interface expressiva e flexível é fundamental para o Serviço de Descoberta. Ela deve ser capaz de suportar vários tipos de consultas geradas por aplicações com requisitos distintos. Também é importante que a interface suporte à especificação de restrições de contexto dos recursos como, por exemplo, sua disponibilidade, em termos de processamento e memória.

O Capítulo 2 contém uma discussão mais geral sobre serviço de descoberta e Agentes de Recurso, onde são apresentandos alguns trabalhos relacionados e o Capítulo 3 mostra em mais detalhes como cada um desses problemas foi abordado na proposta deste trabalho.

As restrições de contexto de um sistema devem ser descritas separadamente da parte funcional da aplicação. A utilização de contratos para descrever os requisitos relacionados ao contexto, no nível da arquitetura, é uma metodologia que tem se mostrado adequada para prover o princípio de separação de interesses [Loques & Sztajnberg 2004; Garlan et al 2004], permitindo aos projetistas se concentrarem nos aspectos específicos da aplicação, em diferentes etapas do desenvolvimento.

A concepção de sistemas de software com a capacidade de adaptação dinâmica baseada no princípio de **Arquitetura de Software**, considera que um sistema pode ser desenvolvido com base em sua organização, como uma composição de componentes e interações entre eles [Loques & Sztajnberg 2004]. Essa abordagem facilita a reutilização dos módulos do sistema e incentiva a separação de interesses, possibilitando separar os requisitos funcionais, que representam a funcionalidade básica de uma aplicação, dos requisitos não-funcionais, relacionadas ao contexto, como distribuição, tolerância a falhas ou a localização de uma pessoa dentro de um ambiente inteligente. Dessa forma, é possível que um sistema de software seja concebido através de sua descrição arquitetural, que inclui os componentes funcionais, as interações entre eles e os requisitos não-funcionais [Carvalho 2001]. Isso possibilita a criação de um ambiente que permite a implantação e execução de um sistema, seguindo as informações contidas em sua configuração arquitetural (seus módulos e as instruções para instanciá-los e conectá-los) descrita em uma linguagem de descrição de arquitetura (*Archicteture Description Language* – ADL) [Lisbôa, 2003].

Neste contexto, é necessário uma maneira de especificar as restrições de qualidade de serviço, permitindo definir os recursos mínimos necessários para que os serviços de uma aplicação possam ser oferecidos [Frolund & Koistinen 1998; Aagedal & Ecklund 2002; Becker & Geihs 1997]. Além disso, é necessário que esses recursos possam ser monitorados e gerenciados em tempo de execução, o que possibilita detectar quando os níveis de qualidade de serviço não são mais satisfatórios. Nesse caso, pode ser efetuada uma tentativa de adaptação dinâmica para que os serviços

continuem a ser oferecidos. Uma maneira de especificar essas restrições é através de um contrato de QoS, que expressa um relacionamento formal entre duas ou mais partes que utilizam ou fornecem recursos [Beugnard et al 1999]. Em um contrato, podem ser definidos os direitos e as obrigações dos participantes e regras de adaptação, que consistem de ações a serem tomadas quando ocorrem mudanças na disponibilidade dos recursos que violam os níveis de qualidade declarados no contrato [Curty 2002; Loyall et al 1998].

Vários *frameworks* vêm sendo propostos para dar suporte ao desenvolvimento de aplicações distribuídas que oferecem serviços com qualidade diferenciada, de acordo com o estado do contexto de execução [Curty 2002; Loques & Sztajnberg 2004; Loyall et al 1998; Cheng et al 2002; Wang et al 2001; Nahrstedt et al 2001]. Eles permitem que as restrições de contexto sejam especificadas e gerenciam a adaptação do sistema de acordo com a variação na disponibilidade dos recursos monitorados. No entanto, muitos *frameworks* não têm a entidade de descoberta de recursos integrada em suas infra-estruturas, deixando o problema a ser resolvido por trabalhos futuros. Isso limita, por exemplo, o desenvolvimento de aplicações pervasivas cuja disponibilidade de recursos varia constantemente. Estes entram e saem do contexto da aplicação a todo o momento, tornando inviável o desenvolvimento dessas aplicações sem o suporte de um mecanismo para a descoberta dinâmica de recursos. Outro problema se refere aos mecanismos de monitoração que usualmente são implementados como *hotspots* do sistema, não permitindo sua reutilização por outras aplicações. Além disso, a aplicação muitas vezes fica dependente de um mecanismo específico de monitoração usado no baixo nível, dificultando executá-la em diferentes ambientes de operação.

Um exemplo de *framework* é o CR-RIO (*Contractual Reflective Reconfigurable Interconnectable Objects*) [Curty 2002], projetado para permitir a especificação e o suporte a contratos de QoS, associados aos componentes da arquitetura de uma aplicação. Atualmente, entretanto, este *framework* não possui suporte explícito à descoberta de recursos em sua linguagem de descrição de contratos e em sua infra-estrutura de suporte. Além disso, a monitoração é feita através da consulta direta aos Agentes de Recurso. Isso requer dos projetistas o conhecimento dos detalhes de comunicação com cada tipo de agente e inviabiliza o gerenciamento

de aplicações pervasivas, devido à carga que a atividade de monitoração pode impor ao dispositivo.

1.1 Investigação e Resultados Preliminares

Durante a etapa de investigação inicial de nosso trabalho, pudemos participar de estudos específicos nos assuntos discutidos na seção anterior, realizados por nosso grupo de pesquisa. Alguns destes estudos resultaram em contribuições preliminares, que nos levaram à proposta desta dissertação:

- em [Sztajnberg et al 2005] apresentamos questões relacionadas à especificação e à garantia de requisitos não-funcionais de qualidade para sistemas compostos por componentes ou serviços independentes. Dentre outras questões, mostramos alguns exemplos de ferramentas que podem ser usadas para a monitoração e coleta das informações de contexto dos recursos. Também é abordada uma ferramenta para a descoberta de recursos com suporte à especificação de restrições de contexto;
- em [Freitas et al 2005] mostramos um *framework* que inclui um conjunto de conceitos e mecanismos apropriados para especificar os requisitos a serem cumpridos e implantar as políticas para a utilização e compartilhamento dos recursos disponíveis;
- visando incorporar suporte à descoberta de recursos no CR-RIO, em [Cardoso et al 2006; Cardoso et al 2006b] apresentamos uma proposta inicial de extensão da linguagem de descrição de contratos para permitir a especificação da aplicação em termos de recursos virtuais que devem ser descobertos dinamicamente em tempo de execução. Demostramos a viabilidade da proposta por meio de uma aplicação de videoconferência. Para esta aplicação desenvolvemos um modelo de objetos para Agentes de Recursos sobre o qual foram desenvolvidos agentes específicos para a monitoração dos diversos tipos de recursos utilizados pela aplicação.

Observamos que, idealmente, uma infra-estrutura de gerenciamento de sistemas deve utilizar uma interface padronizada na monitoração e descoberta de recursos para não ficar presa a tecnologias particulares. Isso facilitaria portar e executar a infra-estrutura para diferentes ambientes de operação. Por exemplo, ambientes de computação em grade e de sistemas pervasivos normalmente usam

diferentes tecnologias para a descoberta de recursos e monitoração. A utilização de uma interface padronizada possibilita abstrair os detalhes específicos da instrumentação usada no baixo nível. Para o desenvolvimento de uma interface padronizada para a monitoração e descoberta de recursos, é importante identificar seus requisitos gerais, de forma a cobrir suas funcionalidades mais importantes.

A partir dos resultados preliminares e com estas questões em mente, este trabalho tem três focos principais: (i) desenvolver uma interface padronizada para os Agentes de Recurso através de um Serviço de Contexto independente de tecnologia; (ii) desenvolver uma interface padronizada para a descoberta de recursos e um Serviço de Descoberta que permita especificar restrições de contexto no processo de descoberta; e (iii) refinar o CR-RIO através de extensões na linguagem de contratos, visando incorporar uma semântica para suportar explícitamente recursos virtuais e também para a especificação de perfil de seleção. Com isto, será possivel descrever os contratos em termos de recursos virtuais a serem descobertos dinâmicamente. Já na infra-estrutura de suporte, foram estudadas as responsabilidades de cada componente do CR-RIO e proposta a integração dos Serviços de Contexto e de Descoberta. Para validar a proposta, são descritos alguns casos de uso que permitem ilustrar o funcionamento do Serviço de Contexto e do Serviço de Descoberta, em conjunto com o CR-RIO.

1.2 Objetivos

Este trabalho tem como objetivos:

- Revisar alguns conceitos referentes a sistemas sensíveis ao contexto, Agente de Recurso, serviço de descoberta e ao gerenciamento autônomo de aplicações sensíveis ao contexto. Apresentar detalhes do *framework* CR-RIO que é usado como estudo de caso desta dissertação.
- Propor um Serviço de Contexto que provê uma interface padronizada para a coleta de informações de contexto dos recursos. Este serviço vai abstrair os detalhes de comunicação com os Agentes de Recurso. Observa-se que este serviço será utilizado pelo Serviço de Descoberta, também proposto neste trabalho.

- Propor um Serviço de Descoberta que inclui uma forma de representação dos recursos e das consultas de descoberta. Será mostrado como este serviço integra-se ao Serviço de Contexto, permitindo o processamento de consultas com suporte a restrições de contexto.
- Discutir como o Serviço de Contexto e o Serviço de Descoberta podem ser integrados a *frameworks* de suporte a aplicações ubíquas e sensíveis ao contexto, tomando como base a infra-estrutura de suporte do CR-RIO. Investigar a validade da proposta através de exemplos de aplicações, demonstrando como o CR-RIO, já integrado com os serviços propostos, permite a descrição, implantação e o gerenciamento destas aplicações.

1.3 Estrutura da Dissertação

Os capítulos restantes da dissertação estão organizados da seguinte maneira:

- No Capítulo 2 há uma descrição dos conceitos básicos utilizados na dissertação, como a definição de contexto, aplicação sensível ao contexto, Agente de Recurso e Serviço de Descoberta. Também se discute o gerenciamento de aplicações sensíveis ao contexto, identificando as principais atividades desse processo.
- O Capítulo 3 apresenta a proposta desta dissertação, que é dividida em três partes. A primeira mostra uma maneira de se representar os recursos e suas propriedades de contexto. A segunda aborda o Serviço de Contexto, um serviço de alto nível que tem o propósito de abstrair os detalhes de comunicação com os Agentes de Recurso. A terceira discute o Serviço de Descoberta que suporta a descoberta de recursos levando em consideração restrições de contexto a serem satisfeitas. Para cada um dos serviços é proposta uma interface padronizada de utilização com base na linguagem XML.
- O Capítulo 4 apresenta, de forma geral, a arquitetura do *framework* CR-RIO, que é uma infra-estrutura de suporte para o gerenciamento de aplicação sensível ao contexto, usado como base para os exemplos e validação da proposta. Posteriormente, é discutido como realizamos a integração do Serviço

de Contexto e do Serviço de Descoberta à infra-estrutura de suporte do *framework* CR-RIO. É mostrado como especificar os contratos em termos de recursos virtuais que devem ser descobertos dinamicamente e também como os serviços propostos se encaixam na infra-estrutura de suporte do CR-RIO.

- No Capítulo 5, apresentamos exemplos de aplicações que permitem ilustrar e validar a proposta deste trabalho. Selecionamos três aplicações com restrições de contexto de domínios distintos: (i) vídeo sob demanda, (ii) videoconferência, e (iii) vídeo sob demanda em ambiente pervasivo. O objetivo é demonstrar, utilizando o suporte ao gerenciamento de contratos do *framework* CR-RIO, a flexibilidade de nossa proposta, que pode ser usada, tanto no gerenciamento de aplicações "convencionais", quanto pervasivas.
- Finalmente, o Capítulo 6 apresenta as conclusões do trabalho, um resumo de suas contribuições e sugestões de trabalhos futuros.

2 Conceitos Básicos

Este Capítulo apresenta os conceitos e termos básicos relacionados à dissertação, particularmente, a conceituação de *contexto* e de *aplicação sensível ao contexto*. São introduzidos conceitos sobre Agente de Recurso e também de serviços de descoberta de recursos, mostrando suas funcionalidades, principais atividades e alguns trabalhos relacionados. Em seguida, são apresentadas as responsabilidades principais de uma infra-estrutura geral de suporte ao gerenciamento de aplicações sensíveis ao contexto e alguns trabalhos relacionados.

2.1 Contexto

No decorrer deste trabalho iremos utilizar as definições de contexto e aplicação sensível ao contexto apresentadas em [Dey 2000]. 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 considerado 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, em que a relevância depende da tarefa que o usuário quer fazer.*

A maioria dos sistemas computacionais modernos é tratada como "caixa-preta", para minimizar a complexidade e prover a funcionalidade adequada. Ao disponibilizar o acesso às informações de contexto nestes sistemas, podem-se desenvolver aplicações adaptáveis sensíveis ao contexto. Espera-se, no futuro, que os sistemas computacionais sejam desenvolvidos com essa preocupação.

Schilit (1994) define três categorias de contexto que podem ser refinadas em subcategorias [Mendoza 2003]:

- **Contexto de Computação**: informações contextuais relacionadas a aspectos computacionais de um sistema sensível ao contexto.
    - Contexto de Aplicação: *e-mails* recebidos, *websites* visitados;
    - Contexto de Sistema: tráfego da rede, estado dos recursos, largura de banda, QoS.
- **Contexto de Usuário**: informações contextuais relacionadas aos usuários de uma aplicação.
    - Contexto Pessoal: estado de saúde, humor, agenda, atividade;
    - Contexto Social: atividade de grupo, relacionamento social, pessoas próximas.
- **Contexto Físico**: informações contextuais relacionadas aos aspectos físicos de uma aplicação sensível ao contexto.
    - Contexto Geográfico: localização dos recursos, hora atual;
    - Contexto de Ambiente: temperatura, luminosidade, altitude.

Algumas dessas categorias já são usadas por sistemas não sensíveis ao contexto, como o sistema de telefonia celular, que têm diversas informações, como hora, capacidade da rede e, em alguns casos, localização do usuário. No entanto, essas informações normalmente não são utilizadas para provisionar serviços diferenciados para o usuário.

2.2 Agentes de Recurso

Agente de Recurso é um elemento cuja função primária é permitir que as aplicações tenham acesso a informações de contexto do seu ambiente de operação. Da mesma maneira que o protocolo IP esconde os detalhes de comunicação de baixo nível usados pelo *hardware* de rede, é desejável que os Agentes de Recurso escondam os detalhes de baixo nível usados na coleta dos dados brutos. Isso é possível com o estabelecimento de uma camada com interface uniforme, usada para obter informações de contexto. Essa idéia é ilustrada na Figura 1 que mostra como os Agentes de Recurso (*ProcessingAgent* e *NetworkAgent*) "escondem" os sensores da aplicação, que interagem somente com o agente que, por sua vez, conhece os detalhes necessários para obter os dados dos sensores. Assim, a aplicação não precisa se preocupar com o tipo de sensor utilizado na coleta dos dados para cada tipo de recurso. Esta sempre usa a mesma interface disponibilizada pelos Agentes de Recurso. Na Seção 3.2 é discutida em mais detalhes a proposta de interface padronizada, independente da tecnologia usada na obtenção das informações de contexto dos recursos.

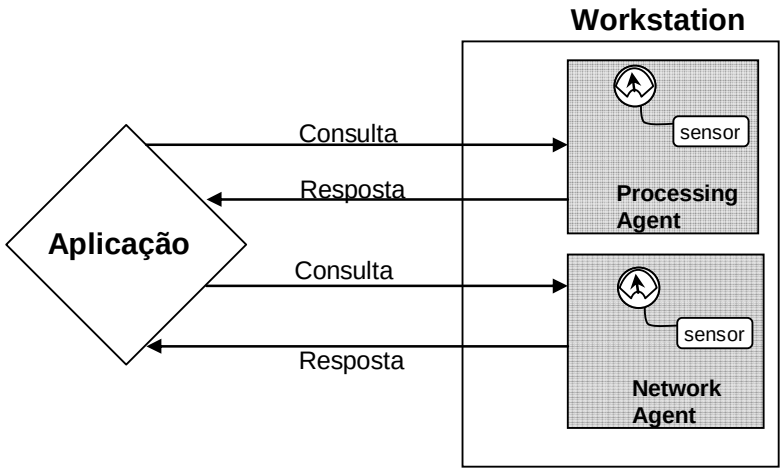


Figura 1 - Agentes de Recurso e sensores.

De forma geral, os Agentes de Recurso provêem os seguintes benefícios para as aplicações:

- **Separação de interesses,** por esconder a complexidade de baixo nível dos sensores utilizados na coleta das informações. Dessa forma, o projetista pode desenvolver seus sistemas orientados a uma interface padronizada, e não a

tecnologias particulares. Conseqüentemente, ele poderá escrever sua aplicação sem precisar se importar, por exemplo, se uma pessoa está sendo localizada através de sensores infravermelhos, GPS ou por um sistema de processamento de imagens.

- **Abstração de mais alto nível** das informações de contexto, para que as mesmas se encaixem nos requisitos das aplicações. Suponha que a aplicação precise saber a largura de banda entre dois pontos que estão separados, física e logicamente, por vários enlaces. Um agente de rede, utilizando SNMP, deve coletar as medidas entre todos os enlaces no caminho e retornar uma medida consolidada que, neste caso, será a medida do enlace com menor capacidade. Resumindo, a aplicação quer saber apenas o atraso ou largura de banda fim-a-fim. Outro exemplo seria uma aplicação pervasiva que precisa monitorar a localização de uma pessoa e utiliza notificação assíncrona por eventos. O Agente de Recurso deve monitorar a localização do usuário dentro do prédio ou cidade e apenas notificar movimentos relevantes, como de uma sala para outra, ou de uma rua para outra, descartando eventos menos “significantes”, como a movimentação dentro de uma mesma sala. A relevância de cada evento é específica de cada aplicação e os Agentes de Recurso devem prover um meio de tratar os requisitos de diferentes sistemas.

- **Interface uniforme** para coleta e notificação de informação de contexto. Não é preciso se preocupar com o tipo de informação de contexto que está sendo coletado, visto que todos os agentes vão disponibilizá-la da mesma maneira. Isto é possível, utilizando uma forma de representação “neutra”, independente de tecnologia na representação das consultas aos Agentes de Recurso e de suas respostas. O Capítulo 3 aborda uma proposta para este problema, utilizando a linguagem XML.

- Agentes **reutilizáveis e customizáveis**. Um Agente de Recurso pode ser implementado e depois reutilizado por diversas aplicações que necessitem das informações por ele disponibilizadas. Por exemplo, um agente que provê informações do ambiente de rede será útil para aplicações distintas, como videoconferência, *streaming* de vídeo e de voz sobre IP. Portanto, uma vez desenvolvido, ele poderá ser reutilizado por diferentes sistemas, facilitando o seu desenvolvimento.

2.2.1 Requisitos Gerais de Agentes de Recurso

Como já discutido anteriormente, os Agentes de Recurso fornecem o suporte básico para a coleta de informações de contexto. Porém, sua utilização exige a satisfação de alguns requisitos básicos, como consultas leves, já que dispositivos móveis têm bateria limitada e é preciso reduzir a comunicação da aplicação com os agentes, visando economizar a carga da bateria. Outro requisito diz respeito ao intervalo de tempo e à precisão com que as informações são coletadas, visto que devem ser suportadas tanto aplicações que requerem alta precisão, quanto aquelas em que esse requisito pode ser mais relaxado.

Dentre os principais requisitos que os Agentes de Recurso devem satisfazer, temos:

Consultas leves: muitas vezes, a aplicação é executada em dispositivos com capacidade limitada de processamento, como um computador de bolso (*Personal Digital Assistance* – PDA), que requer a capacidade de obter as informações de contexto, exigindo o mínimo de esforço na aplicação. Entretanto, os Agentes de Recurso podem estar distribuídos e a monitoração de um conjunto de recursos a partir de um PDA, pode ser inviável devido ao *overhead* de comunicação. Assim, esse requisito pode ser satisfeito por um serviço de contexto de mais alto nível que sintetize as informações oriundas de diversos Agentes de Recurso e depois as repasse para a aplicação. Este serviço deve disponibilizar uma interface que esconda a complexidade da consulta submetida pela aplicação, o que potencialmente simplifica o seu desenvolvimento, já que toda a carga necessária para obter e processar as informações de contexto é responsabilidade de tal serviço. O Capítulo 3 descreve uma proposta de Serviço de Contexto.

Consultas assíncronas x síncronas: existem duas maneiras diferentes de acessar as informações de contexto: através das interfaces de (i) Consulta e (ii) Monitoração. Geralmente, a interface de consulta é usada quando se quer acessar imediatamente as informações de contexto. Por exemplo, ela é necessária no momento da instanciação da aplicação, quando é preciso saber o estado atual do contexto de seus recursos. No entanto, depois da inicialização, é preciso saber somente as variações significativas das propriedades de contexto monitoradas. É neste ponto que entra a interface de monitoração baseada em eventos, estabelecendo suporte à monitoração assíncrona.

Com isso, os clientes não precisam fazer consultas contínuas (*polling*) no processo de monitoração, economizando, assim, os seus recursos. Por exemplo, em uma aplicação executada em um PDA (em que é necessária uma comunicação constante pelo enlace sem-fio somente para monitorar informações de contexto) a carga da bateria pode esgotar-se rapidamente, inviabilizando o desenvolvimento de aplicações sensíveis ao contexto, em dispositivos móveis. Isso evidencia a necessidade de monitoração baseada em eventos, que notifica a aplicação somente quando ocorrerem mudanças relevantes nas propriedades do recurso monitorado.

Facilidade de implementação: considerando a diversidade de informações contextuais, deve-se permitir ao projetista escolher a forma mais conveniente de implementar os agentes e seus mecanismos de coleta e armazenamento de informações. Muitas vezes, há informações de contexto essencialmente estáticas, como o sistema operacional de um servidor ou número da sala pertencente a uma pessoa. Nestes casos, o modo mais natural de implementação seria permitir aos agentes que provêem informações estáticas usarem um banco de dados ou qualquer outra forma de armazenamento. No entanto, na maioria das vezes as informações de contexto são altamente dinâmicas, o que inviabiliza armazená-las estaticamente em um banco de dados. Situações desse tipo requerem que as informações sejam computadas dinamicamente pelo sistema. Por exemplo, um sistema de localização de pessoas não precisa saber onde alguém está o tempo todo. Ele somente precisa fornecer essa informação quando alguma aplicação o requisitar. Além disso, o serviço normalmente deve obter as informações no maior nível de precisão exigido pelas aplicações. Por exemplo, saber que uma pessoa está em determinado prédio é mais fácil do que saber em qual sala específica ela se encontra.

Suporte a meta-atributos: as aplicações necessitam que os Agentes de Recurso suportem alguns tipos de meta-atributos, tais como: (i) **precisão,** que informa o grau de precisão do valor da propriedade fornecida. Por exemplo, no sistema de vídeo sob demanda pervasivo, apresentado na Seção 5.3, é preciso identificar se um usuário está em determinado prédio, mas não em qual sala exatamente; (ii) hora da **última atualização,** que especifica a última hora em que o valor da propriedade foi medido ou modificado; e (iii) **intervalo de amostragem,** que diz o intervalo de tempo usado na coleta do valor da propriedade no sensor. A aplicação deve especificar suas restrições a esses meta-atributos, para garantir que os resultados obtidos sejam úteis.

Monitoração de grupos de recursos: em muitos casos é preciso monitorar grupos de recursos no ambiente de execução. Por exemplo, este requisito é importante para uma aplicação que necessita fazer balanceamento de carga entre os recursos. Através da monitoração do grupo de recursos, a aplicação é capaz de obter o estado global dos recursos e fazer a distribuição de carga de acordo com uma política específica. Em [Freitas 2005] há vários exemplos de aplicação com balanceamento de carga e experimentos envolvendo aplicações do tipo *Bag Of Tasks*, que precisam de informações sobre os recursos distribuídos para agendar e distribuir as tarefas. Sem conhecer o estado dos seus recursos, uma aplicação deste tipo teria dificuldades para agendar apropriadamente as tarefas e distribuir a carga de modo satisfatório.

Sincronização das Respostas: muitos sistemas dependem de informações de contexto oriundas de diversos agentes para tomar uma decisão de adaptação, ou seja, a decisão é tomada a partir da composição destas medidas. Uma medida isolada não contém informação suficiente de contexto para a tomada de decisões. Assim, torna-se necessário um mecanismo de sincronização das medidas retornadas pelos agentes distribuídos pelo ambiente. Fazer com que os agentes se sincronizem é uma tarefa particularmente desafiadora. No entanto, o auxílio de uma entidade de mais alto nível, responsável por enviar as consultas aos agentes e agregar as respostas para as aplicações, torna esta tarefa viável. O Capítulo 3 descreve como a proposta de Serviço de Contexto atende a esse requisito.

Representação das Informações: outro requisito importante é uniformizar o modo como são representadas e organizadas as informações disponibilizadas pelos Agentes de Recurso, já que há diversos tipos de agentes fornecendo informações distintas. Portanto, é desejável uma representação padronizada dessas informações para não depender de codificação específica para cada uma delas. Com a padronização do formato das informações, é possível ter uma melhor interoperabilidade dos Agentes de Recurso e das aplicações.

2.2.2 Sistemas de Monitoração

O Global Grid Fórum definiu uma arquitetura básica de monitoração em ambientes de computação em grade, chamado de Arquitetura de Monitoração em Grade (*Grid Monitoring Archicteture* - GMA) [Aydt et al 2001], que consiste em três

componentes: (i) consumidores; (ii) produtores; e (iii) serviço de diretório (*Directory Service* - DS). Os produtores se registram no DS, informando o tipo e a estrutura das informações que disponibilizarão na grade. Os consumidores podem consultar o DS sobre os tipos de informações disponíveis e localizar os produtores que provêem tais dados. Assim que obtêm essas informações, os consumidores podem contatar os produtores diretamente e requisitar os dados necessários.

Em [Judd & Steenkiste 2003] é proposto o *Contextual Information Service* – CIS - que provê às aplicações, acesso a informações de contexto do recurso através de um banco de dados virtual. Uma característica importante é a interface de acesso ao serviço, feita por meio de uma linguagem de consulta tipo SQL (*Structured Query Language*), usada em banco de dados relacional. Através desta linguagem, o CIS provê uma abstração de alto nível para a coleta das informações de contexto, escondendo os detalhes de como estas são coletadas e armazenadas. Assim, as aplicações só precisam se preocupar com os dados necessários - toda a complexidade de buscar essas informações fica sob responsabilidade do CIS.

Outro trabalho importante é o *Context Toolkit* [Dey 2000] que propõe uma plataforma de alto nível para acessar as informações de contexto dos recursos. Seu objetivo principal é separar o processo de aquisição das informações de contexto da maneira como elas são utilizadas e entregues às aplicações. O autor utiliza uma metodologia orientada a objetos, introduzindo três abstrações: *widget*, servidor (*server*) e interpretador (*interpreter*). O *widget* de contexto é um componente usado para manipular o contexto e abstrair a maneira como essas informações são coletadas, provendo separação de interesses e facilidade de reutilização. O interpretador é responsável pela interpretação das informações de contexto, transformando os dados brutos em informações de alto nível para a aplicação. O servidor de contexto implementa a agregação dos dados de contexto e é usado para comunicar e coletar todos os dados de contexto associados a cada tipo de recurso, a partir de seus *widgets*. O servidor de contexto pode ser considerado um *widget* composto, agindo como um *proxy* entre a aplicação e outros *widgets* de interesse. Isto elimina a necessidade da aplicação se comunicar com vários *widgets* associados a um tipo de recurso.

Além das propostas apresentadas nesta seção, outros trabalhos envolvendo técnicas e mecanismos de monitoração são descritos na literatura. Dentre as mais abrangentes podem ser citados *Network Weather Service* – NWS [Wolski et al 1999],

Ganglia [Massie et al 2004], Remos [Dinda et al 2001], Nprobe [Moore et al 2003], Windmill [Watson et al 2004], EdgeMeter [Pias & Wilbur 2001], GNP [Eugene & Zhang 2002], MonaLISA [Newman 2003] e Nagios [Nagios 2006]. Dentre ferramentas *standalone* disponíveis podem-se citar NTop [NTop 2006] e Pathload [Jain et al 2002]. A Tabela 1 sumariza as características de algumas das plataformas citadas. Todos os exemplos mostrados estão amplamente disponíveis na Internet.

Tabela 1 - Quadro comparativo

	NWS	**Ganglia**	**GNP**	**Remos**
Configuração	Hierárquica	Partições	Partições	Hierárquica
Atividade do projeto	Alta	Alta	Baixa	Baixa
Esforço inicial [a]	Baixo	Baixo	Alto	Baixo
Overhead na rede	Alto	Baixo	Baixo	Médio
Sincronização nas medições [b]	Sim	Sim	--	Não

a. Estimativa da dificuldade de se adicionar um novo *host* no sistema.
b. Refere-se à capacidade de múltiplos *hosts* sincronizarem o envio de mensagens de medição.

De acordo com a Tabela 2, o NWS cobre medidas básicas de qualidade em redes, as quais são obtidas através de testes ativos. O Remos, por sua vez, usa as informações obtidas por meio de coletores SNMP, que são amplamente disponíveis, sem introduzir *overheads* adicionais (exceto para medidas entre domínios diferentes, quando testes ativos têm que ser realizados inserindo tráfego na rede). O GNP permite a obtenção da latência a um custo mínimo, já que obtenção desta medida entre dois nós é feita a partir de um simples cálculo de distância entre dois pontos, a qual é obtida numa fase preliminar. Tanto a Remos quanto o NWS são capazes de realizar previsões dos valores de suas métricas, o que pode ser útil para a configuração de aplicações em ambientes previsíveis. Já o Ganglia somente é usado para obter os valores das propriedades locais dos recursos, tais como: CPU, memória e disco.

Tabela 2 - Resumo das propriedades coletadas e dos recursos das ferramentas

	Largura de Banda	**Latência**	**Uso de CPU**	**Previsões**
NWS	sim	sim	sim	sim
Remos	sim	sim*	sim	sim
GNP	não	sim	não	não
Ganglia	não	não	sim	não

Todas essas ferramentas poderiam ser diretamente utilizadas pelas aplicações na coleta de informações de contexto. No entanto, elas possuem interfaces distintas, fazendo com que a aplicação fique dependente de uma tecnologia particular. Surge então a necessidade dos projetistas desenvolverem as aplicações sobre uma interface padronizada, usada na coleta de informações de contexto, que vai criar uma camada de abstração sobre as ferramentas utilizadas no baixo nível.

2.3 Descoberta de Recursos

Embora seja possível que as aplicações utilizem recursos previamente conhecidos, diretamente, (sem envolver alguma forma de catálogo), é fundamental permitir que os recursos sejam descobertos dinamicamente para o desenvolvimento de aplicações adaptáveis e pervasivas. Principalmente nestas últimas, seus componentes devem interagir em ambientes que se modificam constantemente e que, para isso, precisam de um serviço de descoberta que localize dinamicamente uma instância de recurso que satisfaça as suas necessidades [KindBerg & Fox 2002]. Além disso, um Serviço de Descoberta pode potencialmente minimizar o *overhead* administrativo e aumentar a usabilidade das aplicações [Fen et al 2005].

No entanto, a atividade de descoberta de recursos normalmente é complexa e propensa a resultados insatisfatórios. Geralmente, os recursos são distribuídos e têm tipos heterogêneos, usando diferentes sistemas operacionais e tecnologias de transporte. Outra característica importante é que recursos similares podem ter diferentes propriedades. Por isso, muitas vezes não é possível saber, dentre um conjunto de recursos “similares”, quais atendem às funcionalidades requeridas. Todos estes problemas devem ser resolvidos por uma infra-estrutura de descoberta de recursos moderna.

Atualmente, existem alguns serviços de descoberta de recursos que apesar de serem muito parecidos, possuem origens, infra-estrutura tecnológica, recursos e público-alvo diferentes, pois cada um enxerga o problema por um ângulo distinto, com diferentes implementações, prós e contras. Além disso, cada serviço possui uma linguagem de consulta diferenciada. Na Seção a seguir, são apresentados alguns dos principais componentes e atividades gerais que são comuns à maioria das tecnologias de descoberta.

2.3.1 Componentes e Atividades Gerais

As tecnologias de descoberta freqüentemente usam terminologias diferentes para definir as características de suas escolhas de projeto. Uma terminologia comum pode ajudar na análise das tecnologias. Dentre os componentes e atividades mais importantes no contexto desse trabalho, temos:

Infra-estrutura de descoberta: podem-se utilizar dois modelos na construção de uma infra-estrutura de descoberta de recursos: (i) com Serviço de Diretório, e (ii) sem Serviço de Diretório. Os serviços que utilizam o modelo de diretório possuem um componente dedicado, chamado de **Diretório de Recursos** responsável pelo armazenamento das informações dos recursos e do processamento das consultas e registros de recursos. Este diretório pode ser um diretório distribuído e também hierárquico, possibilitando dividir o domínio dos recursos. Um exemplo de diretório hierárquico que pode ser usado na construção do Diretório de Recursos é o *Lightweight Directory Access Protocol* –LDAP [Yeong et al 1995]. Os serviços de descoberta que não usam o modelo de diretório não possuem um Diretório de Recursos dedicado, fazendo com que cada consulta recebida seja processada por todos os recursos, e caso um deles case com a consulta, esta deve ser respondida imediatamente. Este último modelo tem sido utilizado por serviços de descoberta que utilizam técnicas *Peer-to-Peer*-P2P, onde não há uma entidade centralizada.

Representação de recursos e seus atributos: usada para descrever o tipo, atributos e características funcionais do recurso. Através da representação, uma aplicação pode descobrir um recurso, especificando o seu tipo ou alguns de seus atributos. Uma técnica comumente usada é a representação baseada em *templates*. Por exemplo, o *Service Location Protocol* – SLP [Guttman et al 1999] usa *templates* com pares atributo-valor, usando a notação ASN.1/ABNF. Já outros protocolos, como o UPnP [Microsoft 2000], utilizam uma representação em XML.

Descoberta de recursos: uma aplicação pode descobrir um recurso de duas maneiras: a primeira é através de anúncios, em que as aplicações interessadas “escutam” um canal compartilhado. Dessa forma, quando um recurso anuncia sua presença, todas as aplicações ficam cientes. No entanto, essa técnica não é aconselhável em ambientes muito dinâmicos, onde os recursos entram e saem o tempo todo, devido a possível sobrecarga de mensagens de anúncios que seriam geradas no ambiente. Uma saída

para esse problema é usar um canal de notificação com filtros como proposto em [Motta 2005]. A segunda maneira é baseada em consultas, em que as aplicações enviam consultas de descoberta para o serviço somente quando precisam de um recurso. A vantagem desta técnica é que ela elimina a necessidade das aplicações tratarem anúncios que não interessam.

Localização do Serviço de Descoberta: antes que uma aplicação possa submeter uma consulta ao serviço de descoberta, ela deve saber a localização deste serviço. Idealmente, ela deve ser capaz de obter dinamicamente a localização do serviço de descoberta em seu domínio de operação. Uma solução empregada por alguns serviços é a utilização de um canal *multicast* compartilhado pela aplicação e o serviço de descoberta. Assim, para saber a localização do serviço de descoberta, a aplicação envia uma mensagem para o canal. O serviço ao receber a mensagem retorna sua localização corrente para a aplicação. Obtendo a localização do serviço de descoberta a aplicação pode começar a submeter suas consultas. Cada solução empregada possui suas vantagens e desvantagens em termos de confiabilidade e segurança que devem ser tratadas apropriadamente.

Domínio de descoberta: pode-se minimizar a carga nas aplicações, recursos e diretórios dividindo o domínio de recursos em subconjuntos de interesse. O escopo do domínio pode levar em conta a topologia das redes, as responsabilidades das aplicações, gerência administrativa, informações de contexto ou uma combinação dessas informações. Por exemplo, na descoberta baseada em hierarquia de rede, alguns serviços usam a rede local ou o alcance de uma rede sem fio para delimitar o domínio da descoberta. Alguns serviços mais sofisticados utilizam informações contextuais de alto nível para delimitar o escopo. Estas informações podem ser dados temporais, espaciais ou atividades dos usuários das aplicações que podem ser usadas para diminuir o esforço das aplicações no processo de descoberta**.**

Seleção dos recursos: Apesar de o escopo de descoberta limitar o número de recursos contido no resultado de uma consulta, este ainda pode conter uma lista com dois ou mais elementos. Quando isso ocorrer, os serviços podem oferecer opção de seleção automática ou manual. O modo manual provê às aplicações controle total, passando a elas a responsabilidade da escolha dos recursos contidos na lista obtida como resposta. No outro extremo, situa-se a seleção totalmente automática, em que o serviço de descoberta escolhe os recursos que melhor atende à aplicação, com base

em uma função de utilidade ou uma política *default* de seleção**.** Através da função de seleção o Serviço de Descoberta é capaz de classificar os recursos maximizando ou minimizando algumas propriedades. Na Seção 3.3 apresentamos uma proposta de Serviço de Descoberta que provê o modo manual de seleção. A aplicação pode ser configurada para aplicar seus próprios perfis de seleção sobre a lista de recursos. Isso é mostrado no Capítulo 5 onde o *framework* CR-RIO filtra os recursos a partir de perfis de seleção especificados, no nível de arquiteutra, na descrição dos contratos.

2.3.2 Descoberta de Recursos e Recuperação de Informações

Descoberta de Recursos e Recuperação de Informações (*Information Retrieval*) são atividades que compartilham questões similares. A questão da coleta de informações aborda o problema de uma aplicação em busca de algum tipo de informação, a partir de um conjunto de objetos a partir dos quais esta busca precisa ser satisfeita [Weide 2001]. Pode-se notar que esse conceito é análogo ao de descoberta de recursos. Assim, alguns dos conceitos aplicados na recuperação de informações também podem ser empregados na descoberta de recursos.

Em [Weide 2001] discute-se a divisão das técnicas de recuperação de informações em quatro tipos diferentes:

- **Recuperação baseada em palavras chaves:** a busca baseia-se em palavras-chaves informadas na consulta. Esta técnica é bem conhecida pelos mecanismos de busca na Internet, como o Google. O principal problema desta técnica é que o uso de palavras-chaves não permite capturar aspectos ligados à semântica em uma consulta, visto que as palavras podem ser sinônimas (palavras sintaticamente diferentes podem ter o mesmo significado) ou homônimas (a mesma palavra-chave pode ter diferentes significados) o que, conseqüentemente, compromete a precisão da consulta. Além disso, a relação de diferentes palavras-chaves não pode ser manipulada na consulta.
- **Recuperação baseada em conceitos**: define ontologias para a classificação dos recursos, permitindo, assim, a recuperação baseada em tipos, ao invés de palavras-chaves. Esta técnica possibilita uma boa precisão, mas é de difícil manutenção e desenvolvimento. A principal dificuldade, neste caso, é definir uma ontologia

consistente do domínio, e também combinar ontologias com conceitos muitas vezes contraditórios [Weide 2001].

- **Recuperação dedutiva**: neste método, a semântica dos recursos é expressa formalmente, usando lógica. A recuperação consiste na dedução de quais recursos satisfazem à funcionalidade descrita na consulta. Em termos teóricos, este método pode alcançar uma precisão perfeita [Weide 2001]. Os problemas desta técnica são de ordem prática, pois a modelagem formal da descrição dos recursos e das consultas não é um problema trivial. Além disso, o processo de casamento neste método, que é feito através de prova, pode ser complexo e lento.
- **Recuperação baseada em tabelas:** consiste em pares atributo-valor que representam as propriedades dos recursos. Neste caso, os recursos e requisição são representados por tabelas com pares atributo-valor que devem ser satisfeitos por meio de casamento de padrão. Este método permite que a semântica seja informada com mais precisão do que a obtida pelo uso de palavras-chaves. No entanto, o problema de sinônimo e homônimo continua a existir. Posteriormente no Capítulo 3, será mostrado como nossa proposta de Serviço de Descoberta faz uso desta técnica para descoberta dinâmica de recursos.

2.3.3 Descoberta de Recursos Baseada em Ontologias

Um dos tópicos abordados pelas subseções anteriores foi a importância de uma representação comum dos recursos entre os elementos envolvidos na atividade de descoberta. Esta seção mostra como seria a descoberta de recursos baseada em ontologias (ou conceitos), cujo mecanismo usa o conceito de ontologias com o propósito de obter, entre as aplicações, um entendimento semântico comum e compartilhado dos recursos. Embora nossa proposta seja baseada em tabelas, ela pode ser estendida para suportar a descoberta por meio de ontologias. Para isso, as informações de propriedades dos recursos poderiam ser utilizadas como indicações para organizar ou estabelecer relações semânticas entre as diferentes classes de recursos. Assim, deixaremos registrada uma discussão inicial sobre o uso de ontologias em serviços de descoberta.

2.3.3.1 Ontologias e Representação de Recursos

O processo de descoberta de recursos é desafiador, devido ao seu aspecto dinâmico e distribuído, aliado à falta de entendimento comum do relacionamento entre os recursos. O potencial grande volume de informações usadas para representar os recursos torna difícil encontrar, organizar, acessar e manter as informações requeridas pelas aplicações na identificação dos mesmos. Neste contexto, as ontologias oferecem um meio de lidar com a representação de recursos de informação. O modelo de domínio descrito por uma ontologia pode ser usado como uma estrutura unificadora para respeitar a semântica e dar uma representação comum à informação [Davies et al 2003].

As ontologias têm se tornado populares principalmente porque seu objetivo é promover um entendimento comum e compartilhado sobre um domínio, que pode ser comunicado entre pessoas e sistemas de aplicação [Davies et al 2003]. Uma ontologia define um vocabulário específico para descrever certa realidade mais um conjunto de decisões explícitas, fixando de forma rigorosa o significado pretendido para o vocabulário. Assim, ela envolve um vocabulário de representação que captura os conceitos e relações em algum domínio, além de um conjunto de axiomas, que restringem a sua interpretação [Guarino 1998]. O uso potencial de ontologias para lidar com o problema da semântica de recursos, sobretudo quando há grande volume de informações, tem sido largamente explorado pelas pesquisas da *Web* Semântica [Davies et al 2003] e da Gerência de Conhecimento [Staab et al 2001].

2.3.3.2 Ontologias e Descoberta de Recursos

O mecanismo de descoberta baseado em palavras-chaves não considera a semântica dos objetivos do requisitante, dos recursos e do contexto. Desta forma, retorna objetos com descrições com as palavras-chaves da consulta, o que, na maioria dos casos, gera resultados com pouca precisão. Outro problema desse tipo de mecanismo é não capturar completamente a semântica das requisições das aplicações por não considerar as possíveis relações entre as palavras-chaves. Isso dificulta a descoberta sensível ao contexto, já que as informações de contexto podem estar altamente relacionadas e ter diversas representações, tornando-as difíceis de serem

interpretadas e usadas [Pokraev 2003]. Uma possível solução para esse problema é a utilização de ontologias para classificar os objetos com base em suas propriedades, o que permite fazer a descoberta através da semântica dos recursos. Além disso, as ontologias podem especificar as inter-relações dos recursos e prover uma representação não ambígua dos mesmos.

Em [Broens 2004], são descritos dois requisitos para construir sistemas baseados em ontologias que devem ser considerados ao se criar mecanismos de descoberta baseados nelas:

- **Linguagem**: as ontologias precisam ser codificadas em alguma linguagem antes de ser usadas pela aplicação. Essa linguagem deve ser capaz de codificar conceitos no domínio do problema e ser expressiva o bastante para possibilitar a representação dos relacionamentos entre os conceitos. Atualmente, há várias linguagens para codificar ontologias, como OWL [W3c 2006a], e RDF/RDF-S [W3C 2006].
- **Ambiente**: generalizar, analisar, modificar e manter ontologias são aspectos importantes para um sistema baseado nas mesmas. Quem cria as ontologias, o que inserir e como mantê-las consistentes são questões que devem ser tratadas. No Serviço de Descoberta, esses aspectos podem ser cobertos pela plataforma de anúncio ou registro de recursos, já que ele é uma entidade centralizada do sistema. Isso pode facilitar a criação e manutenção das ontologias.

2.3.4 Serviços de Descoberta de Recursos

Vários grupos de pesquisa têm desenvolvido serviços de descoberta de recursos. Exemplos de projetos incluem: *Intentional Naming System* (INS) [William et al 1999], Ninja *Service Discovery Service* (SDS) [Hodes et al 2002], Jini [Arnold 1999], Microsoft *Universal Plug and Play* (UPnP) e o Apple *Rendezvous* [Cheshire & Krochmal 2005]. Outros serviços, igualmente importantes e bastante conhecidos, incluem *Salutation Protocol* [Salutation 1999] e o *Service Location Protocol* (SLP) [Guttman et al 1999]. Em [Batista 2003] pode-se encontrar uma descrição detalhada de cada um destes protocolos.

A Tabela 3 mostra a classificação de alguns dos protocolos citados de acordo com os componentes e atividades descritas anteriormente na Seção 2.3.1.

Tabela 3 - Comparação de serviços de descoberta de recursos

Funcionalidade	**Serviço de Descoberta**			
	Ninja SDS	**Jini**	**SLP**	**INS**
Nomeação dos recursos e atributos	N/D	*templates*	*templates*	N/D
Localização do Serviço de Descoberta	*unicast, multicast e broadcast*	*unicast e multicast*	*unicast, multicast e broadcast*	*unicast e multicast*
Registro e descoberta	consulta e anúncio	consulta e anúncio	consulta e anúncio	consulta e anúncio
Infra-estrutura de descoberta	Diretório ou hierárquico	Diretório ou hierárquico	Com ou sem diretório	Diretório ou hierárquico
Domínio de diretório	domínio administrativo, contexto(localização) e topologia de rede	domínio administrativo, contexto(localização) e topologia de rede	Topologia de rede	domínio administrativo
Seleção do recurso	manual	manual	manual	automático

Uma tabela mais detalhada, envolvendo outros serviços e funcionalidades, pode ser encontrada em [Fen et al 2005]. Todos os serviços citados anteriormente suportam a descoberta de recurso em ambientes computacionais, em termos de topologia de rede ou localização [Fen et al 2005]. Porém, nenhum deles foi projetado para escalas globais, considerando informações contextuais como localização física e atividade do usuário. Além disso, nenhum desses serviços tem um esquema de nomeação claramente definido.

Com o objetivo de oferecer esses e outros suportes, novas arquiteturas têm sido projetadas. Um exemplo é o *Objectified Naming System* - ONS [Kyungmin et al 2006] que é 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 ou domínios administrativos. ONS trata um nome como um objeto de primeira classe, chamado *name object* que oculta a resolução de nomes e procedimento de religação de serviços quando o contexto ou domínios administrativos mudam. Esta característica simplifica o desenvolvimento das aplicações e permite que os desenvolvedores concentrem-se mais na lógica da aplicação.

Outro exemplo de nova arquitetura é o Olympus (*A High Level Programming Model for Pervasive Computing Environments*) [Rangangathan & Campbell 2003] parte integrante de outro projeto, denominado Gaia, cujo objetivo é dar suporte a aplicações pervasivas. Gaia tem um conjunto de serviços centrais que administram uma coleção de recursos e provê uma interface para o desenvolvedor da aplicação. O objetivo do Olympus é prover uma máquina abstrata com operadores de alto nível que permitam ao desenvolvedor construir sua aplicação num nível mais alto de abstração, permitindo abstrair detalhes relacionados à disponibilidade de recursos no momento do desenvolvimento da aplicação. O ponto chave do projeto é distinguir descoberta de classe e de instância, referindo, respectivamente, à descoberta de uma classe de recursos que atenda aos requisitos do usuário e à obtenção de um recurso desta categoria no ambiente de operação.

Também é importante citar o Q-Cad (*QoS and Context Aware Discovery Framework*) [Capra et al 2005], um *framework* para a descoberta de recursos, que permite as aplicações pervasivas descobrirem e selecionarem recursos que melhor satisfaçam às necessidades dos usuários, considerando o contexto corrente de execução e os requisitos de qualidade de serviço. Ele diferencia sensibilidade ao contexto e à qualidade de serviço, por meio de perfis de aplicação e funções de utilidade, respectivamente. Os perfis de aplicação especificam como o usuário quer que o contexto influencie na descoberta remota de recursos. O elemento central do Q-Cad são os recursos, que podem ser serviços (disponibilizados por provedores remotos), sensores (de onde as aplicações podem obter dados), e componentes (que podem ser carregados remotamente). O *framework* diferencia estes recursos daqueles localizados localmente no *host* da aplicação (exemplo: bateria, memória e CPU). Ele também considera que os recursos remotos podem ser identificados unicamente por meio de um sistema de nomeação disponibilizado por sistema de comunicação de baixo nível. A descoberta de recursos pode ser pró-ativa (conseqüência de uma consulta explícita do usuário) ou reativa (resultado de mudança de contexto). Os dois tipos de descoberta requerem um comportamento similar do *framework*, que é localizar e referenciar um recurso que melhor se encaixe no contexto corrente de execução e que esteja dentro dos requisitos não-funcionais da aplicação.

Cada uma das soluções citadas anteriormente possui limitações, benefícios e formas de implementação, localização e representação de serviços diferenciados.

Surge então a necessidade de se construir uma interface padronizada que abstraia os detalhes específicos de implementação e que gere representações independentes para os recursos e as consultas. Em [Batista 2003] é proposta uma forma de integração de protocolos de localização de serviços em computação pervasiva. Ela permite o compartilhamento de informações entre diferentes tecnologias de descoberta. No entanto, essa proposta não aborda a descoberta de recursos levando em conta restrições de contexto, essencial para aplicações dinâmicas. Para que isso seja possível, deve-ser integrar o serviço de descoberta com alguma plataforma que provê informações de contexto sobre os recursos.

O Capítulo 3 discute como nossa proposta de Serviço de Descoberta integrado ao Serviço de Contexto, provê suporte à descoberta de recursos permitindo a especificação de restrições de contexto a serem satisfeitas. A proposta dispõe de uma interface de alto nível abstraindo as tecnologias usadas no baixo nível aumentando a portabilidade das aplicações para diferentes ambientes de operação.

2.4 Gerenciamento de Aplicações Sensíveis ao Contexto

Com o intuito de gerenciar aplicações com requisitos não-funcionais, de forma que o sistema possa se adaptar dinamicamente diante de variações nos níveis dos recursos utilizados, é necessária uma infra-estrutura que ofereça suporte a esse tipo de gerenciamento. Ela deve permitir especificar as restrições de qualidade desejadas por uma aplicação e as configurações de sistema necessárias para implantar ou adaptar os serviços por ela oferecidos. Além disso, é preciso que essa infra-estrutura suporte a monitoração dos recursos utilizados pela aplicação, a descoberta de novos recursos e o gerenciamento de adaptações no sistema, quando forem necessárias. A seguir são discutidas as principais questões que devem ser cobertas por uma infra-estrutura de suporte, usada para gerenciar as aplicações sensíveis ao contexto.

2.4.1 Infra-Estrutura Geral de Suporte ao Gerenciamento de Aplicações Sensíveis ao Contexto

Como descrito em [Corradi 2005], uma infra-estrutura de suporte ao gerenciamento de aplicações sensíveis ao contexto deve realizar atividades que são refletidas nos seguintes elementos:

- **Especificação:** permite descrever os requisitos funcionais e não-funcionais de uma aplicação e as configurações necessárias para adaptar o sistema, quando esses requisitos forem violados;
- **Monitoração:** mecanismo que monitora as propriedades de contexto dos recursos utilizados pela aplicação;
- **Gerência:** responsável por gerenciar os serviços oferecidos por uma aplicação, considerando as políticas, os níveis dos recursos e o nível de qualidade desejado;
- **Configuração:** tem a função de efetuar as configurações dos elementos funcionais e dos recursos para suportar os serviços oferecidos pela aplicação, assim como a alocação dos recursos a serem utilizados.

A Figura 2 ilustra o relacionamento entre os elementos da infra-estrutura geral de suporte, responsáveis pelas atividades envolvidas no suporte de aplicações sensíveis ao contexto. Para facilitar o entendimento do seu funcionamento, considere o exemplo de vídeo sob-demanda citado na Seção 5.1 baseado em clientes e servidores de vídeo remotos. O elemento Especificação descreve as restrições de contexto que devem ser satisfeitas para os serviços desta aplicação e as configurações necessárias para implantar ou adaptar os serviços oferecidos (passo 1, na Figura 2). Um serviço poderia ter uma restrição, por exemplo, em que a banda passante disponível no canal de comunicação entre o cliente e o servidor teria que ser de, no mínimo, 2 *Mbps*.

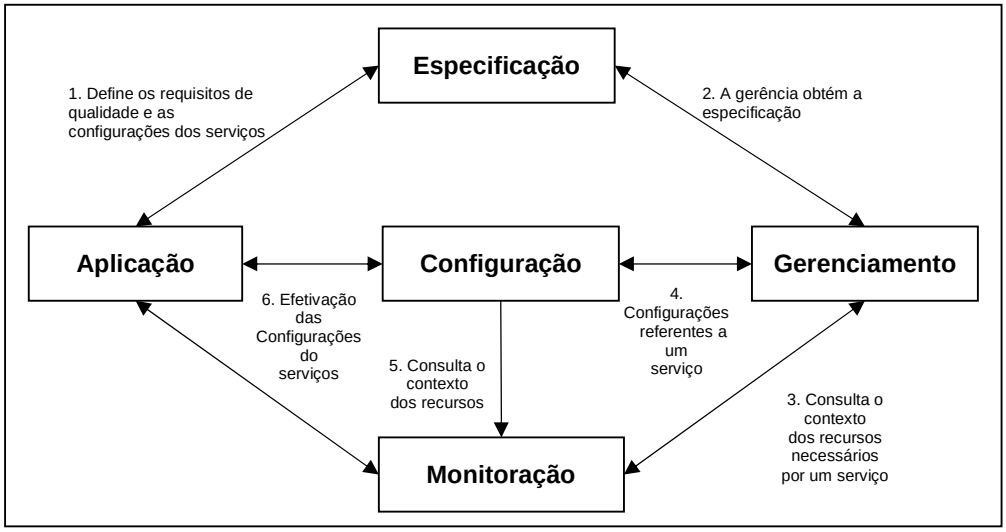


Figura 2 - Infra-estrutura geral de suporte ao gerenciamento de aplicações sensíveis ao contexto.

Ao iniciar o gerenciamento de uma aplicação, de acordo com os seus requisitos não-funcionais, o elemento Gerência consulta as restrições de contexto da aplicação em Especificação (passo 2), no exemplo, os requisitos de processamento e canal de comunicação. Conhecendo os requisitos da aplicação, o elemento Gerência precisa obter informações sobre o nível de operação dos recursos a serem utilizados. Para isso, ele utiliza o elemento Monitoração (passo 3). No exemplo, haveria uma consulta a um Agente de Recurso para medir a banda passante disponível entre o cliente e o servidor. Quando for necessário fazer descoberta de recursos, Gerência deve descobrir servidores que satisfaçam os requisitos mínimos e escolher o melhor deles aplicando uma função de utilidade. Ao constatar que o nível do recurso satisfaz à restrição de contexto para a aplicação, o elemento Gerência passa as configurações do serviço a ser estabelecido, obtidas de Especificação, para o elemento Configuração (passo 4). No exemplo, as configurações orientam a ligação do cliente com um determinado servidor. O elemento Configuração mapeia as informações contidas nas configurações recebidas em ações de sistema que efetivarão a configuração do serviço da aplicação (passo 6).

Depois que o serviço da aplicação estiver em execução, o elemento Gerência coordena a monitoração dos recursos utilizados para verificar se as restrições de contexto da aplicação estão sendo satisfeitas. No exemplo de vídeo sob-demanda, se em um determinado momento a banda passante do canal de comunicação fique reduzida, violando os requisitos da aplicação, o elemento Gerência coordenará as

ações necessárias para tentar manter a aplicação funcionando. Por exemplo, ele poderá solicitar ao elemento Configuração que procure um outro servidor onde o cliente possa ser conectado, respeitando os requisitos de qualidade exigidos para o canal comunicação.

Através da descrição das atividades gerais de uma infra-estrutura de gerenciamento, nota-se a importância dos elementos de Monitoração e Descoberta de Recurso. Idealmente, esses elementos devem prover uma interface padronizada, além de serem reutilizáveis por outras aplicações. No entanto, a maioria dos trabalhos deixa esses mecanismos embutidos na infra-estrutura, dificultando sua reutilização. Além disso, a maioria dos trabalhos deixa o problema da Descoberta de Recurso parcialmente resolvido ou faz o tratamento através de *hotspots*.

2.4.2 Sistemas de Suporte à Adaptação

O sistema QuO (*Quality Objects*) da BBN [Loyall et al 1998] é um framework que estende o CORBA para dar suporte a aplicações que se adaptam de acordo com a disponibilidade de recursos. Neste sistema, o usuário pode definir regiões de operação, enquanto os contratos especificam o desempenho esperado da aplicação. Tais definições são feitas, utilizando o conjunto de linguagens (*Quality Description Languages* - QDL) para descrição de contratos de QoS. O sistema monitora o ambiente de execução e a aplicação, e quando a aplicação muda de região de operação, *handlers* específicos da aplicação são invocados. Os requisitos não funcionais são descritos através do conceito de regiões, que misturam um pouco de contrato de QoS com implementação de sistema, o que dificulta sua legibilidade. Tal fato também compromete a separação de interesses da proposta, uma vez que embute aspectos de implementação nas descrições de alto nível dos requisitos não funcionais. Para obter informações de contexto, o QuO usa os objetos de condição de sistema (*System Condition Objects* – SysCond). Estes provêem interfaces para recursos, gerenciadores de mecanismos ou propriedades, objetos e *Object Request Brokers* - ORBs do sistema que precisam ser medidos e controlados pelos contratos do QuO. Um aspecto negativo do QuO é que ele não tem suporte à descoberta dinâmica dos recursos a serem utilizados pela aplicação, restringindo as possibilidades de adaptação de uma aplicação.

O *Rainbow* [Cheng et al 2002; Garlan et al 2004] permite a especificação dos aspectos a serem monitorados e das situações em que uma adaptação é necessária com o propósito de garantir os requisitos de qualidade da aplicação informados pelo projetista. Em tempo de execução da aplicação, uma infra-estrutura de suporte monitora as propriedades definidas no modelo arquitetural, avalia a ocorrência de violações nas restrições especificadas no mesmo e, se necessário, efetua adaptações no sistema. O *Rainbow* utiliza o conceito de invariante para definição do nível de qualidade desejado e, através das táticas de adaptação, exprime as ações a serem tomadas, se as invariantes não forem respeitadas. Essas táticas incluem os operadores de adaptação que são usados para auxiliar o processo adaptativo. As informações de monitoração são coletadas através de *probes*, depois abstraídas e relacionadas com as propriedades do modelo arquitetural. A descoberta de recursos é feita a partir de operadores usados nas estratégias de adaptação. Na implementação atual, *Rainbow* depende de uma infra-estrutura de descoberta denominada *Network Sensitive Service Discovery* – NSSD [Huang & Steenkiste 2003], que apesar de permitir a descoberta de recursos levando em conta restrições de processamento e de rede, não é capaz de usar outras propriedades de contexto dinâmicas como a localização de um usuário. Além disso, o NSSD usa um conjunto pré-definido de políticas de seleção de recursos.

No contexto de computação ubíqua tem-se o Gaia [Gaia 2006] que é uma infra-estrutura que trata um espaço ativo (*ActiveSpace*) 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. Gaia utiliza uma nova abstração para a computação que é chamada de espaço virtual do usuário (*User Virtual Space*). Um espaço virtual do usuário é composto por dados, tarefas e dispositivos que estão associados a um usuário; ele é permanentemente ativo e independente de dispositivo; move-se com o usuário e mapeia dados e tarefas no ambiente de computação ubíqua do usuário de acordo com seu contexto atual. Gaia converte um espaço físico e seus dispositivos de computação ubíqua em um sistema de computação programável. O *kernel* do Gaia é composto por cinco serviços básicos: serviço de presença, serviço gerenciador de eventos, serviço de contexto, serviço de repositório do espaço e sistema de arquivos de contexto. O serviço de contexto permite que aplicações pesquisem e se subscrevam

para receber informações de contexto. A infra-estrutura de contexto é composta por componentes chamados de provedores de contexto (*context providers*) que fornecem informações sobre o contexto atual. Existe um componente que deduz que tipo de atividade acontece em uma sala específica (ex: reunião ou aula) baseado, por exemplo, em quem está na sala ou que aplicação está sendo executada. O serviço de presença é responsável por detectar entidades físicas (pessoas e dispositivos) e digitais (serviços e aplicações) no espaço ativo. O gerenciador de eventos é responsável por criar canais de eventos e fazer a manutenção dos serviços de evento.

O *framework* CR-RIO (*Contractual Reflective - Reconfigurable Interconnectable Objects*) [Curty 2002; Ansaloni 2003] consiste em uma infra-estrutura projetada para suportar a especificação e o gerenciamento de contratos de qualidade de serviço associados aos serviços oferecidos por um sistema. Os contratos permitem especificar os serviços e associá-los a requisitos não funcionais que expressam os níveis desejados de qualidade. A monitoração desses requisitos é feita por Agentes de Recurso e quando um nível desejado não pode ser mais atendido, o *framework* tenta adaptar dinamicamente o sistema para um novo serviço compatível com os níveis de recursos atuais. A versão do CR-RIO com a qual realizamos nossos experimentos não oferece suporte à descoberta de recursos que até então era feita por meio de *hotspots*. Um dos objetivos desta dissertação é adicionar esta funcionalidade, estendendo a descrição dos contratos e inserindo o elemento de descoberta de recursos na infra-estrutura de suporte. Este *framework* foi escolhido por permitir a especificação de aplicações tradicionais e pervasivas, possibilitando a avaliação da proposta através de um domínio maior de aplicações, além de pertencer ao nosso grupo de pesquisa. Mais detalhes sobre o CR-RIO e de sua integração com o Serviço de Descoberta serão discutidos no Capítulo 4.

2.5 Conclusão do Capítulo

Esse capítulo apresentou os conceitos básicos utilizados na dissertação. Definiram-se os termos *contexto* e *aplicação sensível ao contexto*, e também introduzido conceitos sobre Agentes de Recurso e serviços de descoberta. Identificamos os seus requisitos e as principais atividades, além de apresentar alguns trabalhos relacionados. Discutiram-se os requisitos básicos necessários para o

gerenciamento de aplicação sensível ao contexto e mostrado alguns trabalhos relacionados.

Para a construção de um serviço de descoberta que suporte a especificação de restrições de contexto, é necessário dispor de uma infra-estrutura para coletar e disponibilizar estas informações de contexto dos recursos. Assim, deve-se ter dois serviços desacoplados, de contexto e de descoberta, mas que, em conjunto, possam prover uma infra-estrutura de alto nível usada pelas aplicações. Um ponto importante é a representação dos recursos e também das consultas e respostas para cada um destes serviços. Esta representação deve ter, idealmente, uma linguagem expressiva e ser independente de plataforma.

Com relação à infra-estrutura gerenciamento de aplicação sensível ao contexto, esta deve ser capaz de obter as informações de contexto e fazer a descoberta dos recursos de forma transparente, sem se prender a tecnologias particulares. Para atingir este objetivo é necessário utilizar uma interface de alto nível, que esconda os detalhes da forma como essas operações são realizadas.

Com base nos conceitos e definições apresentados neste Capítulo, o próximo será dedicado à proposta desta dissertação. Primeiro será discuto uma forma para representar os recursos em conjunto com suas propriedades de contexto. Posteriormente será mostrado um Serviço de Contexto que provê informações de contexto dos recursos e esconde os detalhes de comunicação com os Agentes de Recurso. Depois será apresentada uma proposta de interface padronizada para serviços de descoberta e também um Serviço de Descoberta para a descoberta com suporte a restrições de contexto. Para estes dois serviços as interfaces propostas são baseadas na linguagem XML com o objetivo de obter uma representação independente de tecnologia. Posteriormente, no Capítulo 4, será mostrado como integrar os dois serviços propostos na infra-estrutura de suporte do *framework* CR-RIO.

3 Proposta

Este capítulo descreve nossa proposta de um Serviço de Contexto e de um Serviço de Descoberta que suporta a especificação de restrições de contexto.

Inicialmente é discutida uma forma padronizada para a representação de recursos que será posteriormente utilizada pelos Serviços de Contexto e de Descoberta. Através dessa representação é possível registrar e conhecer as propriedades e características específicas de cada tipo recurso. Em seguida é apresentado o Serviço de Contexto, responsável por disponibilizar informações de contexto dos recursos e esconder os detalhes de baixo nível, usados na comunicação com os Agentes de Recurso específicos de cada tipo de recurso. Na seqüência é apresentada a proposta de um Serviço de Descoberta, que permite a descoberta e localização de recursos levando em consideração restrições de contexto que devem ser satisfeitas pelos mesmos.

O desenvolvimento do Serviço de Descoberta requer sua integração com o Serviço de Contexto que vai disponibilizar informações a respeito do contexto dos recursos. Além disso, estes serviços devem ser integrados com um Diretório de Recursos que mantém as descrições dos tipos de recursos e também as instâncias de recursos que estão disponíveis.

Uma organização geral para os elementos do Serviço de Descoberta é mostrada na Figura 3 e é constituída de três elementos: (i) Gerenciador; (ii) Diretório de Recursos; e (iii) Serviço de Contexto. Para descobrir um novo recurso, uma aplicação adaptativa submete uma consulta ao Serviço de Descoberta. Para isso ela informa a classe de recurso desejada e também as restrições de contexto que este deve satisfazer. Por exemplo, a aplicação pode querer uma instância de servidor Web (dentre várias réplicas disponíveis) 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. Ao receber

os dados de contexto dos recursos, o Gerenciador aplica um filtro e separa todas as instâncias que satisfazem a consulta da aplicação. Finalmente, ao descartar os recursos não satisfatórios, o Gerenciador retorna a lista de recurso para a aplicação. Detalhes desta integração são discutidos na Seção 3.3.2.

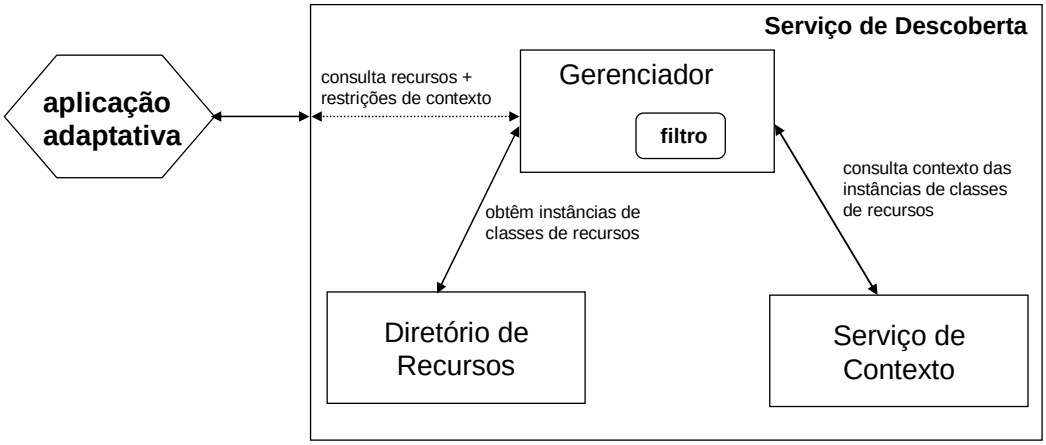


Figura 3 - Elementos do Serviço de Descoberta

Tanto para o Serviço de Contexto quanto para o Serviço de Descoberta, apresentamos uma interface padronizada para a representação de suas consultas e respostas usando a linguagem XML. Usando XML obtemos independência de tecnologia ou linguagem, e também uma maior facilidade para acrescentar novas funcionalidades por meio da inclusão de novos elementos no *schema* XML. Além disso, ela possui técnicas para verificar e validar as informações a partir de modelos pré-definidos como, por exemplo, um *Document Type Definition* - DTD ou um XML *schema*.

3.1 Representação dos Recursos

O desenvolvimento do Serviço de Contexto e do Serviço de Descoberta exige um modo de representar os recursos que podem ser consultados e encontrados. Cada tipo de recurso deve ter um descritor que descreve suas propriedades específicas, o qual deve ser disponibilizado para consulta. Usando a representação do recurso, uma aplicação pode conhecer, para cada propriedade descrita, o nome, o tipo de dado (eg. numérico, *string*) e a unidade de medida (eg. *%, MB, MHz*). A proposta apresentada nesta seção utiliza uma representação de recursos como uma lista atributo-valor,

escrita em linguagem XML Poderíamos utilizar alguma proposta existente para representação padronizada dos recursos, como a descrita em [Batista 2003] que propõe uma extensão do WSDL para mapear representações de outras propostas de serviços de descoberta, como um *template* SLP para uma representação XML. Entretanto, esta extensão apresenta um nível de detalhe que mistura aspectos de representação com aspectos de operação (ex. métodos e parâmetros) dos recursos tornando a representação confusa e pesada. Nosso objetivo não é cobrir todos os aspectos relativos a representação do recursos, como por exemplo detalhes de invocação e uso dos mesmos. Assim, propomos uma maneira mais "limpa" para descrever os recursos e suas propriedades de contexto. A interoperabilidade com outros sistemas que usam alguma outra forma de representação pode ser alcançada, por exemplo, realizando conversão de formato por meio de uma transformação XSLT [W3C 2006b].

A Figura 4 mostra o *schema* XML usado na representação de um recurso. Na linha 4 é definido o elemento principal do *schema*, *Resource*, que contém dois elementos simples (*Type* e *Descritption)* e um elemento complexo (*Attributes*). *Type* especifica o tipo (ou classe) do recurso e *Description* uma descrição do recurso. *Attributes* pode conter uma ou mais ocorrências do elemento complexo *Attribute* que é usado para representar os atributos (ou propriedades) do recurso e contém três propriedades para informar o nome do atributo (*Name*), o tipo de dado (*Type*), e sua unidade de medida (*Units*).

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema
3 elementFormDefault="qualified">
4  <xs:element name="Resource">
5    <xs:complexType>
6      <xs:sequence>
7        <xs:element ref="Type"/>
8        <xs:element ref="Description"/>
9        <xs:element ref="Attributes"/>
10      </xs:sequence>
11    </xs:complexType>
12  </xs:element>
13  <xs:element name="Type" type="xs:string"/>
14  <xs:element name="Description" type="xs:string"/>
15  <xs:element name="Attributes">
```

```
16    <xs:complexType>
17      <xs:sequence>
18        <xs:element maxOccurs="unbounded" ref="Attribute"/>
19      </xs:sequence>
20    </xs:complexType>
21  </xs:element>
22  <xs:element name="Attribute">
23    <xs:complexType>
24      <xs:attribute name="Description"/>
25      <xs:attribute name="Name" use="required" type="xs:NCName"/>
26      <xs:attribute name="Type" use="required" type="xs:NCName"/>
27      <xs:attribute name="Units" type="xs:NCName"/>
28    </xs:complexType>
29  </xs:element>
20 </xs:schema>
```

Figura 4 - Schema XML para a Representação de Recurso

Na Figura 5 é mostrado um exemplo de representação de um recurso do tipo *Processing* (linha 2). Nas linhas 5-12 são informadas as propriedades deste recurso com os seus respectivos tipos e unidades de medida. A linha 5 informa que este recurso, tem o atributo *CPUClock* (velocidade da cpu) do tipo *float* e tem unidade de medida em *MHz*. Já na linha 11, informa-se que ele tem um atributo chamado *OSName* (nome do sistema operacional), que é do tipo *string*. Adicionalmente, pode-se informar a descrição de um atributo através da propriedade *Description* do elemento *Attribute*.

```
1<Resource>
2     <Type>“Processing”</Type>
3     <Description>“Processing Resource”</Description>
4     <Attributes>
5        <Attribute Name=”CPUClock”    Type=”float” Units=”MHz” />
6        <Attribute Name=”TotalMemory” Type=”float” Units=”MB”  />
7        <Attribute Name=”TotalDisk”   Type=”float” Units=”MB”  />
8        <Attribute Name=”FreeMemory”  Type=”float” Units=”MB”  />
9        <Attribute Name=”FreeDisk”    Type=”float” Units=”MB”  />
10       <Attribute Name=”CPUIdle”     Type=”float” Units=”%”   />
11       <Attribute Name=”OSName”      Type=”string”
12                  Description=”Operating System Name” />
13    </Attributes>
14</Resource>
```

Figura 5 - Representação XML de um recurso

A proposta de representação de recursos apresentada nesta seção serve de base para o Serviço de Contexto que, através dela, poderá saber quais propriedades cada tipo de recurso contém e que podem ser consultadas. De forma similar, o Serviço de Descoberta também utiliza essa representação para validar os registros de recursos e as consultas submetidas pelas aplicações.

Usando as representações de recurso, o projetista pode iniciar o desenvolvimento dos Agentes de Recurso que disponibilizam acesso aos valores de cada propriedade. É importante que os agentes não fiquem presos a uma implementação especifica usada na coleta dos valores das propriedades. No Apêndice A são mostrados detalhes de um padrão de projeto que pode ser aplicado no desenvolvimento destes elementos.

As representações dos recursos devem ser armazenadas em um **Diretório de Recursos** que contém as descrições de todos os recursos existentes no domínio. Assim, ao adicionar um novo tipo de recurso, é necessário, primeiro, registrar e armazenar sua descrição no diretório. Uma implementação deste diretório pode utilizar, por exemplo, um banco de dados nativo XML, como o Apache Xindice [Xindice 2006] ou o Exist [Exist 2006], ambos de código aberto.

O responsável pela administração do Diretório de Recursos pode ser o desenvolvedor da aplicação ou o administrador do domínio. Cada opção tem suas vantagens e desvantagens em termos de facilidade de administração e de segurança e a discussão desse problema foge do escopo deste trabalho.

3.2 Serviço de Contexto

Nas próximas subseções será apresentada a proposta do Serviço de Contexto, responsável por disponibilizar informações de contexto dos recursos e esconder os detalhes de baixo nível usados na comunicação com os Agentes de Recurso. A proposta inclui uma interface de acesso padronizada de alto nível. Com esta interface, a aplicação precisa preocupar-se somente com os dados necessários e não como eles são obtidos.

3.2.1 Arquitetura do Serviço de Contexto

Numa arquitetura simplificada para a coleta de informações de contexto, a aplicação comunica-se diretamente com os agentes específicos de cada tipo de recurso utilizado, conforme mostra a Figura 6, que ilustra uma aplicação executada em um PDA que interage com três Agentes de Recurso (*ProcessingAgent*, *NetworkAgent* e *PeopleLocationAgent*). Esta arquitetura tem alguns problemas, tais como: (i) pouca escalabilidade, podendo gerar um tráfego excessivo de dados na medida em que se aumenta o número de aplicações acessando os agentes através da rede, (ii) o possível excesso de comunicação da aplicação com os agentes, inviabilizando o desenvolvimento de sistemas para dispositivos móveis, como descrito na Seção 2.2.1; e (iii) deixa a sincronização dos resultados retornados pelos agentes sob responsabilidade da aplicação.

Além destes pontos negativos, o desenvolvimento das aplicações seria dificultado, já que estas deveriam conhecer os detalhes de interface e comunicação dos diferentes tipos de Agentes de Recurso.

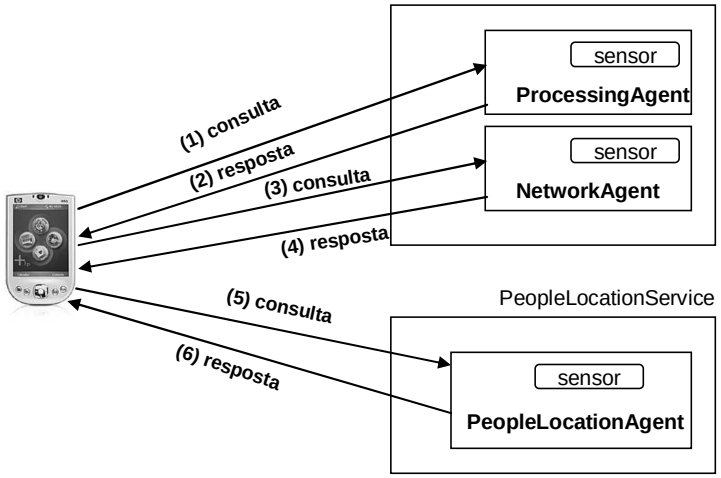


Figura 6 - Comunicação direta do cliente com os Agentes de Recurso

Com o objetivo de esconder a complexidade do processo de consulta às informações de contexto, este trabalho propõe desenvolver um Serviço de Contexto com o qual as aplicações interagem para obter informações. Assim, quem se comunica com os Agentes de Recurso é o Serviço de Contexto, que obtém as informações requeridas para satisfazer determinada consulta comunicando-se diretamente com os agentes, processa todo conjunto de informações obtido e retorna o

resultado, contendo dados consolidados, para a aplicação. O Serviço de Contexto também é responsável pela sincronização das informações coletadas. De forma geral, este serviço pode utilizar a maioria das características específicas disponíveis nos Agentes de Recurso descritos na subseção 2.2.1. Detalhes de como estes requisitos podem ser satisfeitos serão descritos nas próximas seções.

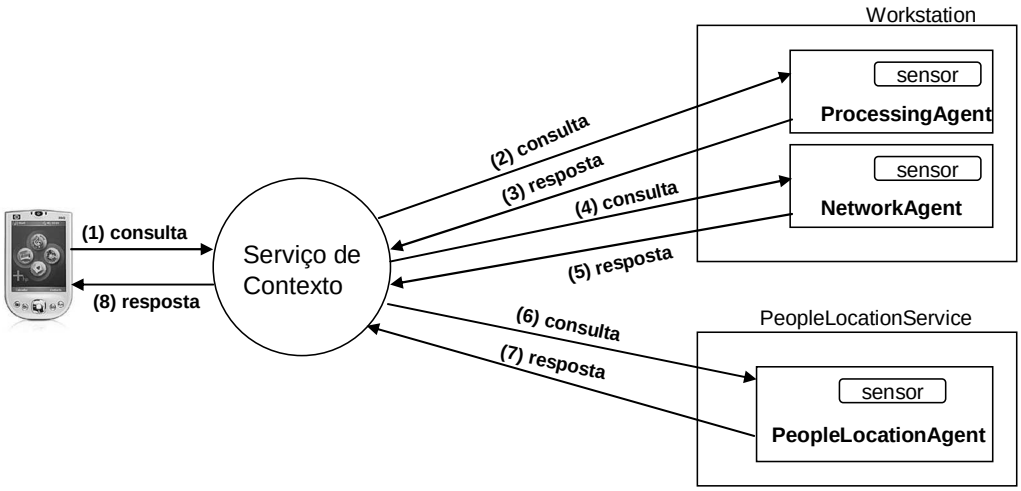


Figura 7 - Proposta de Arquitetura do Serviço de Contexto

O desenvolvimento do Serviço de Contexto torna-se desafiador porque há diversos tipos de informações envolvidos, além de uma variedade de consultas que devem ser suportadas. A Figura 7 mostra a arquitetura geral do Serviço de Contexto mais detalhadamente. Nela, uma aplicação está sendo executada em um PDA e deseja obter informações sobre a disponibilidade de CPU em determinada estação de trabalho, e também localizar o usuário dentro do prédio. Para isso, a aplicação submete uma consulta ao Serviço de Contexto (Figura 7, passo 1), informando os recursos-alvo, e respectivas propriedades, sobre os quais ela quer informações. Adicionalmente, pode-se informar se os vários resultados devem ser sincronizados ou não. Caso seja necessário um resultado sincronizado, o Serviço de Contexto somente retornará a resposta após obter todos os resultados dos agentes remotos.

Ao receber uma consulta da aplicação, o Serviço de Contexto verifica quais propriedades de contexto ela quer obter de cada um dos recursos. É importante ressaltar que uma consulta pode conter um ou mais recursos-alvo, possibilitando monitorar conjuntos de recursos. Após a interpretação da consulta, o Serviço de Contexto comunica-se com cada Agente de Recurso remoto envolvido para obter as

informações de contexto individuais (Figura 7, passos 2, 4 e 6). Ao receber todos resultados (Figura 7, passos 3, 5, 7), o serviço consolida essas informações e as repassa à aplicação (Figura 7, passo 8).

Nas subseções seguintes será apresentada uma proposta de representação para as consultas e respostas do Serviço de Contexto.

3.2.2 Representação de Consultas

Como já descrito nas subseções anteriores, o Serviço de Contexto deve prover uma interface de consulta com poder de expressão suficiente para que as aplicações possam realizar os mais variados tipos de consultas, suportando a especificação de conjuntos de recursos e das propriedades requeridas. A Figura 8 mostra o *schema* XML usado na representação de uma consulta do Serviço de Contexto. Na linha 3, define-se o elemento principal do documento chamado *ContextQuery* que contém um elemento simples (*Synchronized*) e uma ou mais ocorrências do elemento complexo *Target*. *Synchronized* informa que os resultados devem ser retornados ao mesmo tempo, ou seja, o Serviço de Contexto precisa aguardar as respostas de todos os agentes contidos na consulta. *Target* possui uma propriedade chamada URI (*Unique Resource Identifier*) que indica o recurso alvo do qual se deseja obter as informações, e também um elemento complexo, *Attributes*, que contém quatro elementos complexos (*Attribute*, *CollectTime*, *CollectInterval* e *Results*) e uma propriedade (*From*) que aponta qual tipo de recurso fornece os atributos requeridos. *Attributes* pode conter uma ou mais ocorrências do elemento *Attribute*, que por sua vez contém uma propriedade chamada *Name* que informa o nome do atributo. Adicionalmente, pode-se inserir um operador lógico no elemento *Attribute* através das propriedades *Op* (operador lógico) e *Value (*valor). Dessa maneira, é possível restringir os resultados retornados pelo Serviço de Contexto. Por exemplo, a aplicação somente quer obter uma resposta caso o valor de um atributo seja menor ou igual a algum valor qualquer. O elemento *CollectTime* não é obrigatório e tem uma propriedade *Min* que restringe a hora que os dados devem ter sido coletados. Com este elemento, pode-se especificar, por exemplo, que a aplicação somente está interessada em dados coletados nos últimos 50 minutos. Para isso, é importante que a aplicação e o Serviço de Contexto tenham o relógio sincronizado por algum protocolo de sincronização. Além disso, a

aplicação deve estar ciente da margem de erro imposta pela técnica usada na sincronização do relógio. O elemento *CollectInterval*, através das propriedades *Min* e *Units*, permite informar o intervalo mínimo com que os dados são coletados. Finalmente, o elemento *Results*, com a propriedade *Max*, especifica a quantidade máxima de resultados pretendida pela aplicação.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema
  " elementFormDefault="qualified">
3  <xs:element name="ContextQuery">
4    <xs:complexType>
5      <xs:sequence>
6        <xs:element ref="synchronized"/>
7        <xs:element maxOccurs="unbounded" ref="Target"/>
8      </xs:sequence>
9    </xs:complexType>
10  </xs:element>
11  <xs:element name="synchronized" type="xs:boolean"/>
12  <xs:element name="Target">
13    <xs:complexType>
14      <xs:sequence>
15        <xs:element ref="Attributes"/>
16      </xs:sequence>
17      <xs:attribute name="URI" use="required" type="xs:NCName"/>
18    </xs:complexType>
19  </xs:element>
20  <xs:element name="Attributes">
21    <xs:complexType>
22      <xs:sequence>
23        <xs:element maxOccurs="unbounded" ref="Attribute"/>
24        <xs:element ref="CollectTime"/>
25        <xs:sequence minOccurs="0">
26          <xs:element ref="CollectInterval"/>
27          <xs:element ref="Results"/>
28        </xs:sequence>
29      </xs:sequence>
30      <xs:attribute name="From" use="required" type="xs:NCName"/>
31    </xs:complexType>
32  </xs:element>
33  <xs:element name="Attribute">
34    <xs:complexType>
35      <xs:attribute name="Name" use="required" type="xs:NCName"/>
36      <xs:attribute name="Value" type="xs:integer"/>
```

```
37      <xs:attribute name="op"/>
38    </xs:complexType>
39  </xs:element>
40  <xs:element name="CollectTime">
41    <xs:complexType>
42      <xs:attribute name="Min" use="required"/>
43    </xs:complexType>
44  </xs:element>
45  <xs:element name="CollectInterval">
46    <xs:complexType>
47      <xs:attribute name="Min" use="required" type="xs:integer"/>
48      <xs:attribute name="Units" use="required" type="xs:NCName"/>
49    </xs:complexType>
50  </xs:element>
51  <xs:element name="Results">
52    <xs:complexType>
53      <xs:attribute name="Max" use="required" type="xs:integer"/>
54    </xs:complexType>
55  </xs:element>
56</xs:schema>
```

Figura 8 - Schema XML para a Representação de Consultas do Serviço de Contexto

A Figura 9 mostra um exemplo de consulta do Serviço de Contexto. Na linha 2, o elemento *Synchronized* indica que a aplicação quer os resultados sincronizados. Na linha 3, o elemento *Target* informa o endereço de um dos recursos-alvo contido na consulta através da propriedade URI. Na linha 4, define-se o elemento *Attribute* e se especifica que os atributos pertencem ao recurso do tipo *Processing.* Nas linhas 5-7 são especificados os atributos de contexto que a aplicação quer obter. O exemplo mostra, respectivamente, CPU Livre (*CPUIdle*), memória livre (*MemFree*) e disco disponível (*DiskFree*). Na linha 8, o elemento *CollectTime* aponta a hora em que os dados foram coletados e a propriedade *Min* indica que a última consulta deve ter sido feita, no mínimo, às 12h12. *CollectTime* não é obrigatório e caso não seja informado, o Serviço de Contexto retorna as informações mais recentes. Na linha 9, informa-se através do elemento não obrigatório *CollectInterval*, o intervalo mínimo com que os valores dos atributos devem ter sido coletados. Para muitas aplicações, valores que foram coletados de 30 em 30 segundos não são úteis e com o elemento *CollectInterval* a aplicação se assegura que está recebendo dados relevantes. O exemplo especifica que os valores devem ter sido coletados no mínimo a cada 15 segundos. Na linha 10,

o elemento *Results* indica que se pretende, no máximo, uma resposta com 30 resultados para esta consulta.

A Figura 9 também mostra como se monitora conjunto de recursos. Nesta consulta, a aplicação informa dois recursos-alvo dos quais os dados serão obtidos. Além disso, é ilustrado um exemplo de uso de operador lógico (Figura 9, linha 15) que informa que a aplicação somente deseja obter os resultados, caso o atributo de nome *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                  <Attribute Name=”DiskFree” />
8                  <CollectTime Min=”12:12:35 a.m”/>
9                  <CollectInterval Min=”15” units=”sec”/>
10                 <Results Max=”30”/>
11           </Attributes>
12     </Target>
13     <Target URI=”UserLocation.ic.uff.br:7895”>
14           <Attributes From=”UserLocation”>
15                 <Attribute Name=”userId” op=”==” Value=”712” />
16                 <Attribute Name=”currentRoom” />
17           </Attributes>
18     </Target>
19</ContextQuery>
```

Figura 9 - Representação XML de uma Consulta do Serviço de Contexto

3.2.3 Representação da Resposta

A Figura 10 mostra o *schema* XML de uma resposta retornada pelo Serviço de Contexto. Na linha 4, define-se o elemento principal, *ContextResponse*, que contém uma ou mais ocorrências do elemento complexo, *ResouceInfo* que, por sua vez, tem quatro propriedades e um elemento complexo chamado *Attributes*. As propriedades de *ResourceInfo* são usados para indicar a localização (*URI* e *IP*) de onde os dados foram obtidos, a hora da última coleta (*Updated*) e também o intervalo (*Interval*) com que se fez a coleta. É necessário usar as propriedades *URI* e *IP*, visto que uma resposta pode

conter informações de um conjunto de recursos. O elemento *Attributes* pode conter uma ou mais ocorrências do elemento complexo *Attribute* usado para retornar os valores de cada atributo requisitado na consulta. *Attibute* tem quatro propriedades que indicam o nome do atributo coletado (*Name*), o seu valor (*Val*), o tipo (*Type*) e a unidade de medida (*Units*).

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3     elementFormDefault="qualified">
4  <xs:element name="ContextResponse">
5    <xs:complexType>
6      <xs:sequence>
7        <xs:element maxOccurs="unbounded" ref="ResourceInfo"/>
8      </xs:sequence>
9    </xs:complexType>
10  </xs:element>
11  <xs:element name="ResourceInfo">
12    <xs:complexType>
13      <xs:sequence>
14        <xs:element ref="Attributes"/>
15      </xs:sequence>
16      <xs:attribute name="IP" use="required" type="xs:NMTOKEN"/>
17      <xs:attribute name="Interval" type="xs:integer"/>
18      <xs:attribute name="URI" use="required" type="xs:NCName"/>
19      <xs:attribute name="Updated" use="required"/>
20    </xs:complexType>
21  </xs:element>
22  <xs:element name="Attributes">
23    <xs:complexType>
24      <xs:sequence>
25        <xs:element maxOccurs="unbounded" ref="Attribute"/>
26      </xs:sequence>
27    </xs:complexType>
28  </xs:element>
29  <xs:element name="Attribute">
30    <xs:complexType>
31      <xs:attribute name="Name" use="required" type="xs:NCName"/>
32      <xs:attribute name="Type" use="required" type="xs:NCName"/>
33      <xs:attribute name="Units"/>
34      <xs:attribute name="Val" use="required" type="xs:string"/>
35    </xs:complexType>
36  </xs:element>
37 </xs:schema>
```

Figura 10 - Schema XML da resposta do Serviço de Contexto

Na Figura 11 é mostrada a resposta gerada pelo Serviço de Contexto à consulta realizada na Figura 9. Na linha 2, o elemento *ResourceInfo* informa que os dados foram coletados do recurso localizado em *workstation.ic.uff.br* com endereço IP 192.168.1.1. Além disso, aponta que a última coleta ocorreu às 12h12m12s, com intervalo de 1 segundo. Nas linhas de 05 a 07 há alguns atributos de contexto com suas respectivas propriedades para este recurso em particular. Por exemplo, na linha 5, um atributo representa a informação de que a CPU livre (propriedade *Name=“CPUIdle”*) do recurso é de 68% (propriedades *Val=68.0* e *Units=“%”*). Os outros dois atributos das linhas 6 e 7 informam, respectivamente, que o recurso tem 338.80 *MB* de memória livre e 1.053 *GB* de disco livre.

É importante ressaltar que pode-se obter informações de vários recursos em uma única resposta do Serviço de Contexto. O exemplo da Figura 11 contém informações de contexto de dois recursos. Nas linhas de 10 a 16 há dados referentes de uma consulta realizada a um serviço de localização de pessoas. Nesta resposta, ele informa que o usuário com identificador 712 estava na sala 12 as 12:12:12 a.m.

```
1<ContextResponse>
2  <ResourceInfo URI=”wokstation.ic.uff.br” IP=”192.168.1.1”
3    Updated=”12:12:12 a.m” 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      <Attribute Name=”DiskFree” Val=”1053.8” Type=”float” Units="MB"/>
8    </Attributes>
9  </ResourceInfo>
10  <ResourceInfo URI =” PeopleLocation.ic.uff.br” IP=”192.168.1.12”
11    Updated=”12:12:12 a.m”>
12    <Attributes>
13      <Attribute Name=”userId” Val=”712” Type=”int” Units=””/>
14      <Attribute Name=”currentRoom” Val=”12” Type=”int” Units=””/>
15    </Attributes>
16  </ResourceInfo >
17</ContextResponse>
```

Figura 11 - Representação XML de uma Respostado Serviço de Contexto

Se a aplicação quisesse a resposta de forma assíncrona, seria possível ter duas respostas independentes (*ContextResponse*) para cada um dos recursos. Como, na maioria das vezes, as aplicações precisam da informação conjunta sobre o estado dos

diversos recursos para tomar uma decisão, o modo síncrono normalmente é o mais utilizado.

Com a proposta de representação de recurso e das consultas e respostas do Serviço de Contexto, tem-se a base para o desenvolvimento de um Serviço de Descoberta que suporte a especificação de restrições de contexto.

3.2.4 Utilização do Serviço de Contexto

As seções anteriores discutiram a arquitetura geral do Serviço de Contexto e também as representações para suas consultas e repostas. Com isso, pode-se identificar os passos necessários que uma aplicação deve executar na utilização do serviço que são: (i) identificar as propriedades e recursos de interesse, preenchendo a consulta em XML; (ii) Submeter a consulta; (iii) aguardar a reposta; (iv) interpretar a resposta.

Os detalhes relacionados à submissão das consultas e recebimento das respostas ficam dependentes de implementação. Por exemplo, isso pode ser feito por meio de uma chamada de função ou até mesmo por uma requisição HTTP (POST ou GET) em um servidor WEB. Nossa proposta de Serviço de Contexto deixa o projetista livre para escolher o melhor modo para o seu desenvolvimento. No Capítulo 5 são apresentados exemplos mais detalhados de utilização a partir de aplicações gerenciadas pelo *framework* CR-RIO.

3.3 Serviço de Descoberta

Discutiu-se, na Seção 2.3, os conceitos básicos das tecnologias de descoberta de recursos. Foram apontados os seus principais componentes, problemas e também algumas tecnologias disponíveis. Esta seção apresenta a proposta de um Serviço de Descoberta com capacidade de localização levando em conta restrições de contexto que devem ser satisfeitas pelos recursos. É mostrada uma arquitetura genérica independente de tecnologia de software. Inicialmente são discutidas as atividades de registro e desregistro dos recursos e do processamento de consultas. Depois, apresentam-se a interface para a representação de registro de recurso e de consulta de

descoberta. Todas estas representações baseiam-se na linguagem XML, de modo semelhante ao realizado pelo Serviço de Contexto.

3.3.1 Registro e Desregistro de Recursos

Todo recurso deve ser registrado no Diretório de Recursos, após a sua instanciação. Alguns serviços de descoberta também exigem a atualização periódica do registro, com o objetivo de manter o diretório consistente e atualizado. Assim, normalmente existem dois meios de remover o recurso do diretório: (i) desregistrando-o explicitamente, quando ele não estiver mais disponível; e (ii) não atualizar o registro, levando a sua referência a ser removida automaticamente. Normalmente, o segundo caso ocorre quando o recurso falha e não consegue solicitar o desregistro.

A Figura 12 mostra as mensagens trocadas entre os recursos e o Diretório de Recursos. No passo 1, o recurso é registrado no diretório e, posteriormente, atualizado periodicamente (passo 2). Isso evita que o Diretório de Recursos mantenha referências de recursos que falharam ou foram removidos, sem conseguir se desregistrar. No passo 3, o recurso envia uma mensagem de desregistro para o Diretório de Recursos, avisando que não estará mais disponível.

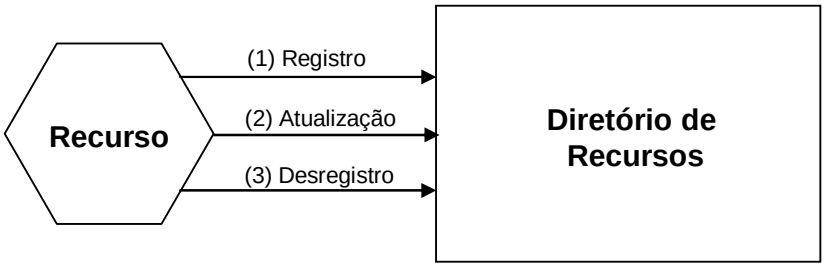


Figura 12 - Mensagens entre Recursos e Diretório de Recursos

O recurso deve utilizar uma interface de registro para anunciar sua presença em seu domínio de operação. Uma interface padronizada para o registro será apresentada posteriormente na Seção 3.3.3.

3.3.2 Arquitetura do Serviço de Descoberta

A Figura 13 apresenta a arquitetura do Serviço de Descoberta composta por três elementos principais: (i) Gerenciador de Consultas; (ii) Diretório de Recursos; e (iii) Serviço de Contexto. Nesta mesma figura são mostrados os passos executados para processar uma consulta de descoberta. Tem-se uma aplicação quer localizar recursos no seu domínio de operação que satisfaçam a certos requisitos de contexto. No passo 1 da Figura 13, ela submete a sua consulta ao Gerenciador que, depois de recebê-la (passo 2, Figura 13), procura, primeiramente, obter do Diretório de Recursos todas as referências de tipos (ou classes) de recursos que atendam a aplicação. Caso a consulta não tenha restrições de contexto, essa lista obtida no passo 2 é imediatamente retornada para a aplicação.

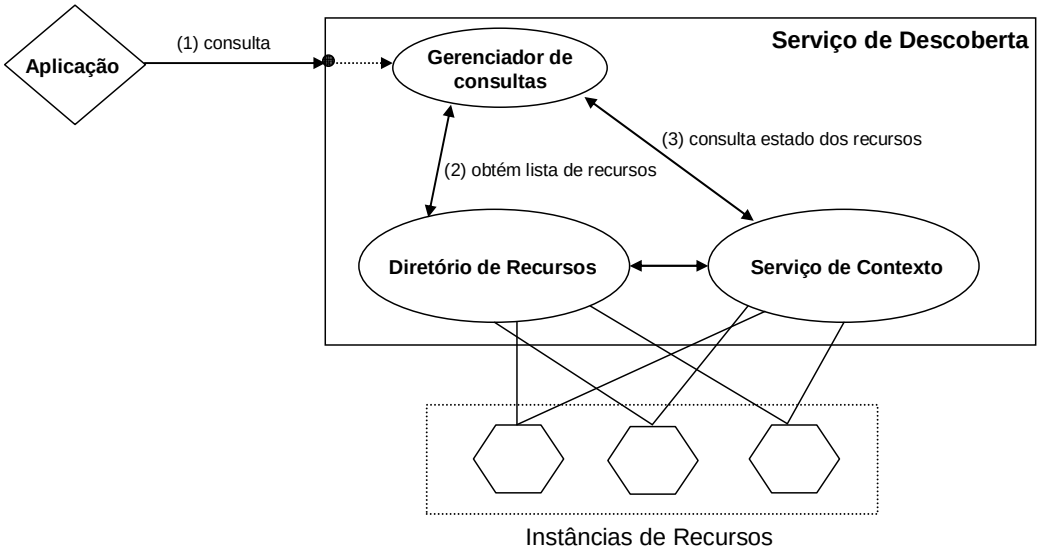


Figura 13 - Arquitetura do Serviço de Descoberta

Nas consultas em que a aplicação quer somente referências a recursos que se encaixem em certas restrições de contexto, o gerenciador deve consultar o Serviço de Contexto (passo 3, Figura 13) para obter as informações de contexto de cada um dos recursos encontrados. Com essas informações, ele poderá classificar os recursos encontrados e filtrar a resposta para a aplicação, retornando somente aqueles que interessam. Seria possível utilizar várias estratégias para aumentar o desempenho do processo de descoberta. Uma das possibilidades seria utilizar um mecanismo de

cache, tanto para as consultas ao diretório de recursos quanto para as do Serviço de Contexto.

3.3.3 Representação do Registro de Recursos

De maneira similar à representação dos recursos, propõe-se um modo de representar os registros de recursos que se tornem disponíveis no ambiente. Através do registro, o recurso informa a sua localização por meio de um identificador URI (*Unique Resource Identifier*) e também os valores de seus atributos. Esses atributos devem obedecer à descrição de seu tipo de recurso já feita e disponível no Diretório de Recursos. Caso não exista uma representação para determinado tipo de recurso, é necessário criar uma representação e armazená-la no diretório.

Na Figura 14 mostra-se o *schema* XML para o documento usado no registro de recurso. Na linha 4 é definido o elemento *ResourceRegister* que possui três elementos simples usados para informar o tipo (*Type*) do recurso, a descrição (*Description)* e o identificador único (URI). *ResourceRegister* também tem um elemento complexo chamado *Attributes* que contém uma ou mais ocorrências do elemento *Attribute*, cujas três propriedades são usadas para informar o nome do atributo (*Name*), seu valor (*Val*) e tipo (*Type*).

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema
3 elementFormDefault="qualified">
4  <xs:element name="ResourceRegister">
5    <xs:complexType>
6      <xs:sequence>
7        <xs:element ref="Type"/>
8        <xs:element ref="Description"/>
9        <xs:element ref="Attributes"/>
10        <xs:element ref="URI"/>
11      </xs:sequence>
12    </xs:complexType>
13  </xs:element>
14  <xs:element name="Type" type="xs:string"/>
15  <xs:element name="Description" type="xs:string"/>
16  <xs:element name="Attributes">
17    <xs:complexType>
18      <xs:sequence>
19        <xs:element maxOccurs="unbounded" ref="Attribute"/>
```

```
20      </xs:sequence>
21    </xs:complexType>
22  </xs:element>
23  <xs:element name="Attribute">
24    <xs:complexType>
25      <xs:attribute name="Name" use="required" type="xs:NCName"/>
26      <xs:attribute name="Val" type="xs:NMTOKEN"/>
27      <xs:attribute name="Type" type="xs:NMTOKEN"/>
27    </xs:complexType>
28  </xs:element>
29  <xs:element name="URI" type="xs:string"/>
30 </xs:schema>
```

Figura 14 - *Schema* XML do Registro de Recurso

A Figura 15 dá um exemplo de representação de um registro de recurso. Nesta figura há uma representação de *display* de vídeo (linha 2, elemento *Type*). As linhas 5-12 especificam as propriedades deste recurso através dos elementos *Attribute* e suas respectivas propriedades. Este recurso, em particular, tem resolução de 800x600 (*resolution*). Também dispõe de informações específicas de implementação, como a versão (*version*), plataforma requerida (*platform*) e tamanho (*size*).

```
1<ResourceRegister>
2      <Type> “VideoDisplay” </Type>
3      <Description> “Java Display Component” </Description>
4      <Attributes>
5            <Attribute Name=”resolution” Val=”800x600” />
6            <Attribute Name=”version” Val=”2.1” />
7            <Attribute Name=”platform” Val=”JVM2” />
8            <Attribute Name=”code” Val=”display800600.jar” />
9            <Attribute Name=”size” Val=”70KB” />
10           <Attribute Name=”cost” Val=”10” />
11           <Attribute Name=”memory” Val=”2” />
12           <Attribute Name=”battery” Val=”4” />
13     </Attributes>
14     <URI> “http://165.165.65.102/display.class” </URI>
15</ResourceRegister>
```

Figura 15 - Representação XML do Registro de Recurso

O elemento URI informa um endereço que é o identificador único da localização do recurso – neste caso, aponta para um endereço HTTP. Em adição as

propriedades básicas dos recursos, podem-se inserir informações usadas para avaliar a qualidade do recurso como o custo, consumo de memória e bateria.

No exemplo da Figura 15, todos os valores dos atributos do recurso são estáticos. Para outros tipos de recursos, há propriedades com valores que variam dinamicamente. Por exemplo, propriedades dinâmicas são as propriedades de consumo de CPU e memória de um recurso de processamento.

```
1<ResourceRegister>
2      <Type> Processing </Type>
3      <Attributes>
4            <Attribute Name=”CPUClock”    Val=”1800MHz” />
5            <Attribute Name=”TotalMemory” Val=”1024MB”  />
6            <Attribute Name=”TotalDisk”   Val=”40GB”    />
7            <Attribute Name=”FreeMemory” Type=”Dynamic”   />
8            <Attribute Name=”FreeDisk”   Type=”Dynamic”   />
9            <Attribute Name=”CPUIdle”    Type=”Dynamic”   />
10     </Attributes>
11     <URI> 192.168.1.102 </URI>
12</ResourceRegister>
```

Figura 16 - Representação XML do registro de um recurso de processamento

A Figura 16 apresenta a descrição XML do registro de um recurso de processamento representado anteriormente pela Figura 1. Os atributos das linhas 4-6 são atributos estáticos e não variam para aquele recurso. Consultas cujas restrições são apenas baseadas em atributos estáticos, podem ser otimizadas uma vez que não existe a necessidade de consultar o Serviço de Contexto. Os atributos das linhas 7-9 são dinâmicos e devem ser consultados por meio do Serviço de Contexto, dinamicamente, sob demanda. O elemento URI informa um endereço que é o identificador único da localização do recurso que neste caso é um endereço IP.

3.3.4 Representação da Consulta de Descoberta

De forma similar à representação dos recursos, as consultas de descoberta também são construídas através de um *schema* XML, cujo formato é mostrado na Figura 17. Na linha 4 define-se o elemento principal *ResourceQuery*, quem contém um elemento simples *MaxResults*, que indica o número máximo de resultados da

resposta e uma ou mais ocorrências do elemento complexo *Constraints*, que especifica as restrições de contexto que os recursos encontrados devem satisfazer. *Constraints* contém uma propriedade (*From*) usada para indicar o tipo de recurso de onde os valores dos atributos, contidos no elemento complexo *Attribute*, devem ser obtidos. Cada restrição é descrita através do elemento *Attribute* que tem propriedades usadas para indicar o nome dos atributos (*Name*), operadores lógicos (*op*) e valores (*Val*).

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema
3 elementFormDefault="qualified">
4  <xs:element name="ResourceQuery">
5    <xs:complexType>
6      <xs:sequence>
7        <xs:element ref="MaxResults"/>
8        <xs:element ref="constraints"/>
9      </xs:sequence>
10      <xs:attribute name="type" use="required" type="xs:NCName"/>
11    </xs:complexType>
12  </xs:element>
13  <xs:element name="MaxResults" type="xs:integer"/>
14  <xs:element name="constraints">
15    <xs:complexType>
16      <xs:sequence>
17        <xs:element maxOccurs="unbounded" ref="Attribute"/>
18      </xs:sequence>
19    </xs:complexType>
20  </xs:element>
21  <xs:element name="Attribute">
22    <xs:complexType>
23      <xs:attribute name="Name" use="required" type="xs:NCName"/>
24      <xs:attribute name="Val" use="required"/>
25      <xs:attribute name="op" use="required"/>
26    </xs:complexType>
27  </xs:element>
28 </xs:schema>
```

Figura 17 - Schema XML de uma Consulta de Descoberta

A Figura 18 mostra uma instância 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 3-12 indicam as

restrições de contexto que devem ser satisfeitas pelos recursos a serem encontrados. Nas linhas 4-7 são indicadas as restrições do tipo *Processing*, informando que o recurso precisa ter *clock* de processamento (*CPUClock*) de, no mínimo, 1.8 Ghz. A linha 6 especifica que o recurso deve ter, no máximo, *50%* de uso de CPU (*CPUIDle*) e a linha 7 informa que se deseja um recurso com sistema operacional (*OSName*) igual a *Windows XP*. Já nas linhas 10-11 são indicadas as restrições do tipo *Transport* do canal 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 *50 ms*.

```
1<ResourceQuery type="VideoServer">
2  <MaxResults>5</MaxResults>
3    <Constraints From="Processing">
4        <Attribute Name="CPUClock"  op=">=" Val="1800MHZ" />
5        <Attribute Name="FreeMemory" op=">=" Val="128" />
6        <Attribute Name="CPUIDle" op=">=" Val="50" />
7        <Attribute Name="OSName"  op="==" Val="Windows XP" />
8    </Constraints>
9    <Constraints From="Transport">
10       <Attribute Name="Bandwidth" op=">=" Val="256" />
11       <Attribute Name="Delay"     op="<=" Val="50" />
12    </Constraints>
13 </ResourceQuery >
```

Figura 18 - Representação XML de uma Consulta de Descoberta

3.3.5 Representação de Resposta do Serviço de Descoberta

Para a representação das respostas geradas pelo Serviço de Descoberta utiliza-se o *schema* XML, definido na Figura 19. A linha 4 estabelece o elemento principal *DiscoveryResponse*, cuja propriedade *Type* indica o tipo de recurso que a resposta está retornando. Por decisão de projeto, na consulta e resposta pode-se especificar somente um tipo de recurso. Dessa forma, se uma aplicação precisa descobrir dois tipos de recursos, ela deve submeter duas consultas independentes ao Serviço de Descoberta. *DiscoveryReponse* contém uma ou mais ocorrências do elemento complexo *ResourceInfo*, usado para localizar os recursos e suas propriedades. Isso porque a resposta pode conter ter vários recursos que satisfaçam as restrições informadas na

consulta. A localização de cada recurso é mostrada pela propriedade URI e IP do elemento *ResourceInfo* que, por sua vez, tem o elemento complexo *Attributes*. Neste, há uma ou mais ocorrências do elemento *Attribute* empregado para representar as propriedades e valores do recurso, respectivamente, através das propriedades *Name* e *Val*. *Attribute* contém uma propriedade (*From)* usada para indicar o tipo de recurso que seus atributos pertencem.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema
3     elementFormDefault="qualified">
4  <xs:element name="DiscoveryResponse">
5    <xs:complexType>
6      <xs:sequence>
7        <xs:element maxOccurs="unbounded" ref="ResourceInfo"/>
8      </xs:sequence>
9      <xs:attribute name="Type" use="required" type="xs:NCName"/>
10   </xs:complexType>
11  </xs:element>
12  <xs:element name="ResourceInfo">
13    <xs:complexType>
14      <xs:sequence>
15        <xs:element ref="Attributes"/>
16      </xs:sequence>
17      <xs:attribute name="IP" use="required" type="xs:NMTOKEN"/>
18      <xs:attribute name="URI" use="required" type="xs:NCName"/>
19    </xs:complexType>
20  </xs:element>
21  <xs:element name="Attributes">
22    <xs:complexType>
23     <xs:sequence>
24        <xs:element maxOccurs="unbounded" ref="Attribute"/>
25      </xs:sequence>
26    </xs:complexType>
27  </xs:element>
28  <xs:element name="Attribute">
29    <xs:complexType>
30      <xs:attribute name="Name" use="required" type="xs:NCName"/>
31      <xs:attribute name="Val" use="required" type="xs:NMTOKEN"/>
32    </xs:complexType>
33  </xs:element>
34 </xs:schema>
```

Figura 19 - Schema XML da Reposta do Serviço de Descoberta

A Figura 20 mostra um exemplo de resposta do Serviço de Descoberta. Na linha 1 está especificado que a resposta contém recursos do tipo *VideoServer*. Nas linhas 2-11 há uma instância de recurso que está localizado no endereço IP *192.168.1.1* e tem propriedades do tipo *Processing*, *CPUIdle* igual a 68.0 (linha 4) e *CPUClock* igual a 1800 MHz (linha 5). Para as propriedades do tipo *Transport* (linhas 7-10), tem-se largura de banda (*Bandwidth*, linha 8) igual a *358 kbps* e atraso (*Delay*, linha 9) igual a *48 ms*.

Nas linhas 12-21 tem-se outra instância de recurso localizado em *192.168.1.2* que tem propriedades, do tipo *Processing*, *CPUIdle* igual a 50.0 (linha 14) e *CPUClock* igual a 600 MHz (linha 15). Para as propriedades do tipo *Transport* (linhas 17-20), tem-se largura de banda (*Bandwidth*, linha 18) igual a *758 kbps* e atraso (*Delay*, linha 19) igual a *50 ms*.

```
1<DiscoveryResponse Type=”VideoServer” >
2  <ResourceInfo URI=”wokstation.ic.uff.br” IP=”192.168.1.1”>
3    <Attributes From=”Processing”>
4      <Attribute Name=”CPUIdle”  Val=”68.0”/>
5      <Attribute Name=”CPUClock”  Val=”1800”/>
6    </Attributes>
7    <Attributes From=”Transport”>
8      <Attribute Name=”Bandwidth”  Val=”358”/>
9      <Attribute Name=”Delay”      Val=”48”/>
10    </Attributes>
11  </ResourceInfo>
12  <ResourceInfo URI=”wokstation1.ic.uff.br” IP=”192.168.1.2”>
13    <Attributes From=”Processing”>
14      <Attribute  Name=”CPUIdle”  Val=”50.0”/>
15      <Attribute  Name=”CPUClock”  Val=”600”/>
16    </Attributes>
17    <Attributes From=”Transport”>
18      <Attribute Name=”Bandwidth”  Val=”758”/>
19      <Attribute Name=”Delay”      Val=”50”/>
20    </Attributes>
21  </ResourceInfo>
22</DiscoveryResponse>
```

Figura 20 - Representação XML de uma resposta do Serviço de Descoberta

3.3.6 Utilização do Serviço de Descoberta

O diagrama de interação da Figura 21 mostra as mensagens trocadas entre a aplicação e os elementos do Serviço de Descoberta. No passo 1, aplicação submete uma consulta de descoberta, de acordo com o schema XML proposto na Seção 3.3.4, informando a classe de recurso requerida e as restrições de contexto. O Gerenciador de Consultas recebe e interpreta a consulta submetida pela aplicação e faz uma requisição ao Diretório de Recursos (passo 2) obtendo às instâncias de recursos da classe requerida (passo 3). Ao obter a lista de recursos o Gerenciador identifica a propriedades de contexto necessárias e submete uma consulta ao Serviço de Contexto (passo 4) e obtém (passo 5) o estado de contexto de cada um dos recursos. Com estas informações o Gerenciador filtra os recursos (passo 6) e retorna uma reposta em XML para aplicação (passo 7).

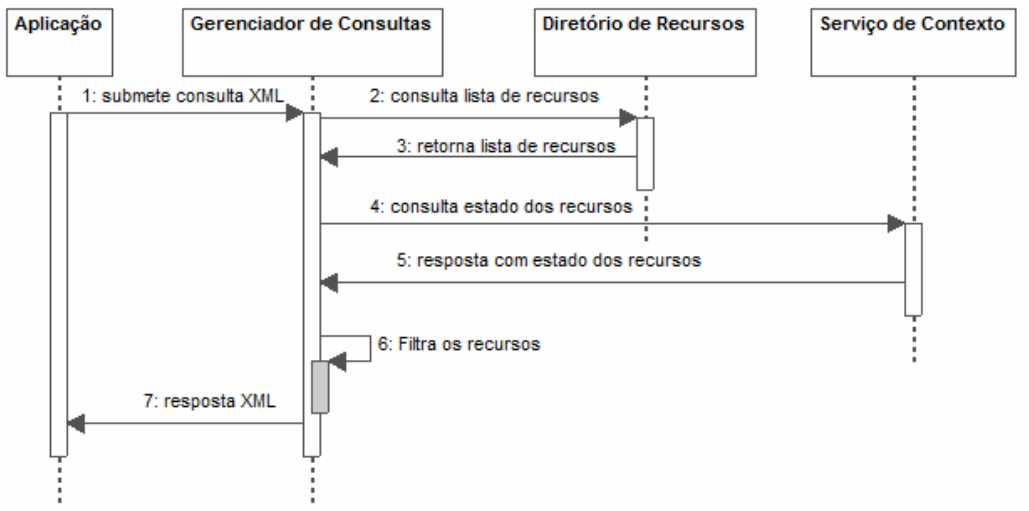


Figura 21 - Diagrama de Interação entre os elementos do Serviço de Descoberta

3.4 Conclusão do Capítulo

Este capítulo dedicou-se à proposta da dissertação. Inicialmente, foi apresentada uma proposta de representação de recursos. Posteriormente introduziu-se o Serviço de Contexto em que se abordaram a sua arquitetura geral e também as representações em XML utilizadas pelo serviço: (i) representação das consultas; e (ii) representação das respostas.

Posteriormente, apresentou-se uma proposta de Serviço de Descoberta que suporta a descoberta, levando em conta restrições de contexto que devem ser satisfeitas pelos recursos. Também foram mostradas as representações para as consultas e respostas deste serviço.

Uma característica que deve ser ressaltada é que as interfaces propostas são independentes de tecnologias de diretório e de instrumentação usada na coleta de informações de contexto.

O próximo capítulo apresenta uma proposta de integração do Serviço de Contexto e do Serviço de Descoberta a uma infra-estrutura de gerenciamento de aplicações sensíveis ao contexto.

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, escolhido por sua flexibilidade em permitir a especificação e suporte tanto de aplicações tradicionais, quanto pervasivas. Isso possibilita a avaliação da proposta considerando ambientes de operação com requisitos distintos. Além dessa flexibilidade proporcionada pelo CR-RIO, este projeto é desenvolvido em nosso grupo de pesquisa e tem uma implementação disponível.

Atualmente não há o suporte à descoberta de recursos no gerenciamento de aplicações realizado pelo CR-RIO, que é feito através de *hotspots* inseridos pelos desenvolvedores das aplicações. Um dos objetivos e contribuições desta dissertação é estender a linguagem de descrição de contratos adicionando o suporte explícito à descoberta de recursos. Além disso, será mostrado como os componentes da infra-estrutura de suporte do CR-RIO interagem com os Serviços de Contexto e de Descoberta. Posteriormente, no Capítulo 5, serão descritos alguns exemplos de aplicação utilizando o CR-RIO com os serviços propostos integrados em sua infra-estrutura de suporte.

4.1 O Framework CR-RIO

A abordagem utilizada pelo *Framework* CR-RIO é centrada 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 (*middleware*) 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 suprir as exigências de contratos; e (iii) prover um conjunto de mecanismos para impor, monitorar e manter os contratos associados à aplicação.

O diferencial da abordagem está na associação direta entre a arquitetura da aplicação como um todo, ou das várias partes que a compõem, e os seus requisitos não-funcionais, descritos por contratos. Por um lado, isso permite que o projetista consiga descrever, com a granulosidade requerida, em que parte da configuração de componentes da arquitetura determinado contrato deve ser imposto. Por outro, sempre que uma adaptação precisa ser efetuada, para manter a qualidade descrita no contrato, ela é realizada através de operações de configuração na arquitetura da aplicação.

Na próxima seção serão apresentados os elementos associados aos contratos e ao sistema de suporte concebido para a implantação dos mesmos.

4.1.1 Contratos

No CR-RIO, um serviço funcional de uma aplicação é considerado uma atividade especializada, definida através da especificação dos componentes arquiteturais e sua topologia de interconexão, geralmente não permitindo negociação [Beugnard et al 1999]. Serviços não-funcionais são definidos pelas restrições às atividades não-especializadas da aplicação e podem admitir alguma negociação envolvendo os recursos utilizados. Um contrato regulando os aspectos não-funcionais da aplicação pode descrever, em tempo de projeto, o uso a ser feito de recursos compartilhados e variações aceitáveis na disponibilidade destes recursos. Este contrato será imposto em tempo de operação por uma infra-estrutura composta por um

conjunto de componentes que implementam a semântica do contrato. Um contrato possui os seguintes elementos:

a) **Categorias:** descrevem as propriedades de recursos ou os aspectos não-funcionais específicos, separadamente dos componentes. Por exemplo, características de processamento, memória ou de comunicação podem estar associadas a uma Categoria. Aspectos menos tangíveis como faixa de preço (“caro” ou “barato”), tolerância a falhas, ou qualidade (“boa”, “média” ou “ruim”) também podem ser descritos. Para cada Categoria devem ser providos componentes ou serviços de suporte correspondentes que vão alocar/reservar e monitorar as respectivas características, utilizando para isso a infra-estrutura disponível.

b) **Perfis:** que quantificam/valoram as propriedades de uma Categoria. A quantificação restringe cada propriedade de acordo com a sua descrição, funcionando como uma instância de valores aceitáveis para determinada Categoria. Componentes ou partes de arquitetura podem definir perfis e restringir seu contexto de operação com a granulosidade desejada.

c) Um conjunto de **serviços**: cada serviço contém um conjunto de restrições que devem ser aplicadas à aplicação, no nível da arquitetura. Isso é feito associando-se um ou mais perfis aos componentes da aplicação, ou à forma de interação dos componentes. Assim, o nível de qualidade desejado/tolerado por um serviço é diferenciado de outro pelo conjunto de propriedades declaradas nos perfis. O conjunto de serviços define os possíveis estados de operação para a aplicação.

d) Uma cláusula de **negociação** (*negotiation*): descreve uma política, definida por uma máquina de estados, que estabelece uma ordem arbitrária para a implantação dos serviços. De acordo com o descrito na cláusula, quando um serviço de maior preferência não puder ser mais mantido, a infra-estrutura de suporte tentará implantar um serviço de menor preferência. O retorno para um serviço de maior preferência também pode ser descrito, permitindo que um serviço de melhor qualidade seja implantado, se os recursos necessários ao mesmo tornarem-se disponíveis.

4.1.2 Descrição de Categorias

Como exemplo, na Figura 22 são descritas as categorias utilizadas na aplicação de videoconferência que será apresentada na Seção 5.2. Cada categoria contém o nome das propriedades de interesse e suas características. A categoria *Processing* (linhas 1-5) representa os recursos de processamento sistema local a serem alocados/monitorados para dar suporte à qualidade esperada pelo usuário. A categoria *Transport* (linhas 7-11) define características de transporte e comunicação, como banda, atraso e *jitter*.

As categorias relacionadas a recursos de áudio e vídeo (linhas 13-25) contêm propriedades que podem ser selecionadas na maioria dos clientes usados na videoconferência. Ao se instanciar um terminal de videoconferência, por exemplo, perfis de áudio e vídeo selecionam os valores específicos das propriedades de áudio (por exemplo, *codec:G711; sampleRate:8000*) e vídeo (digamos, *codec:H261; frame-rate:20*).

```
1 category Processing {
2  utilization:    decreasing numeric %;
3  clockFrequency: increasing  numeric MHz;
4  memReq:         increasing numeric Mbytes;
5 }
6
7 category Transport {
8   bandwidth: increasing numeric Mbps;
9   delay:     decreasing  numeric ms;
10   jitter:    decreasing numeric;
11 }
12
13 category VideoMedia {
14  codec:     enum (H261, H263, MJPEG);
15  quality:   enum (LOW, MEDIUM, HIGH);
16  size:      enum (CIF, QCIF);
17  frameRate: increasing numeric fps;
18 }
19
20 category AudioMedia {
21  codec:        enum (G711,G723, GSM, DVI);
22  sampleLenght: enum (8, 16);
23  sampleRate:   increasing numeric Hz;
```

```
24  channels:     enum (MONO, STEREO);
25 }
```

Figura 22 - Categorias

4.1.3 Infra-estrutura de Suporte

As arquiteturas e contratos descritos em CBabel são mapeados em um modelo de objetos proposto e implementado em [Corradi 2005]. Este modelo é refletido em um repositório de meta-nível, que mantém as informações de configuração da arquitetura, as quais podem ser atualizadas e consultadas durante a vida da aplicação. Este repositório inclui também as representações dos contratos e mantém informações relacionadas a recursos (dispositivos, serviços, aplicações, etc) de interesse das aplicações. Com base nos contratos, que utilizam as informações contidas no repositório, uma infra-estrutura de suporte (*middleware*) é usada para gerenciar as configurações arquiteturais. Esse middleware é composto pelos seguintes componentes: Gerenciador de Contratos (*Contract Manager* - CM), Contratador de QoS (*Contractor*), Agente de Recurso (*Resource Agent*), e Configurador (*Configurator*). A Figura 23 mostra esses componentes e a interação entre eles no processo de gerenciamento de contrato, sendo que cada um possui as seguintes responsabilidades:

Contract Manager: responsável pela implantação e gerência de contratos. O CM interpreta os contratos já mapeados no repositório de meta-nível e extrai dos mesmos as informações dos serviços, respectivos perfis e a máquina de estados de negociação de serviços. Para implantar um serviço o CM pede aos *Contractors* que verifiquem as restrições exigidas pelos perfis do serviço selecionado. Se o CM for informado, por algum *Contractor*, que um perfil não pode ser (mais) atendido, o serviço atual será invalidado. Neste caso, o CM consulta a cláusula de negociação de serviços do contrato visando verificar a existência de candidatos a próximo serviço. Caso exista, ele tenta estabelecer o serviço cujas restrições sejam compatíveis com os atuais níveis de recursos, obedecendo à ordem de preferência descrita na cláusula de negociação de serviços. Se nenhum serviço puder ser provido, o estado *out-of-service* é alcançado e o gerenciamento do contrato para a aplicação é encerrado. O CM pode também iniciar

uma nova negociação quando os recursos para a implantação de um serviço de maior preferência ficarem disponíveis.

Contractor: gerencia a monitoração das propriedades de recursos especificadas nos perfis. Esse gerenciamento consiste no envio de requisições solicitando os valores dessas propriedades aos Agentes de Recurso, além da avaliação das restrições dos serviços em face dos valores monitorados. Caso alguma restrição do serviço seja violada, o Contractor notifica essa ocorrência ao CM. O CM também é notificado pelo Contractor, se as propriedades de recursos por ele monitorados satisfazem as restrições de um serviço com maior prioridade que o serviço corrente.

Resource Agent: Categorias são associadas a Agentes de Recurso (*Resource Agents*) que encapsulam o acesso aos elementos básicos de suporte (mecanismos, recursos ou serviços), provendo interfaces para gerência e monitoração de valores de propriedades requeridas. Os valores monitorados são passados para o *Contractor* quando mudanças são detectadas. O aprimoramento deste componente é um dos focos deste trabalho. Na Seção 4.2 é descrito como a estrutura dos Agentes de Recurso foi modificada para a integração do Serviço de Contexto.

Configurator: elemento responsável por mapear as descrições arquiteturais, em CBabel, para ações do nível de sistema que efetivarão as configurações requeridas na infra-estrutura de suporte da aplicação. O *Configurator* provê duas APIs, configuração e reflexão arquitetural, através das quais as facilidades de configuração são acessadas. A API de configuração permite instanciar, ligar, parar e substituir componentes durante a operação da aplicação. Estas operações são refletidas num repositório persistente de meta-nível, que mantém o estado da aplicação e pode ser consultado através da API de reflexão arquitetural.

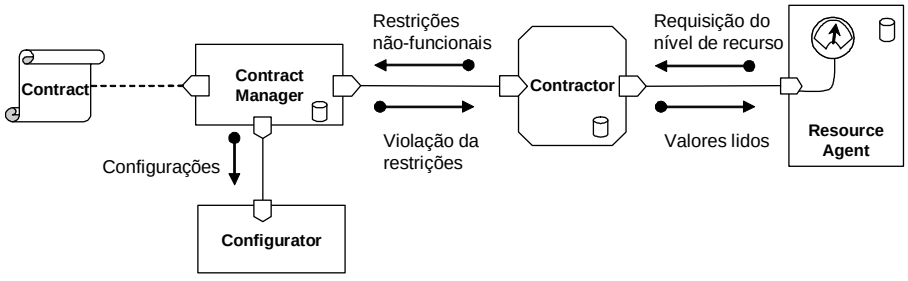


Figura 23 - Elementos do suporte

4.2 Serviço de Contexto x Agentes de Recurso

Atualmente, no CR-RIO utiliza-se o Agente de Recurso básico para monitorar as informações de contexto dos recursos. Assim, o desenvolvedor deve saber especificamente quais Agentes de Recurso ele precisa usar, como contatá-los e também o formato dos dados retornados por cada um deles. As desvantagens deste modelo são discutidas na Seção 3.2.1. Para superar essas limitações, propõe-se utilizar o Serviço de Contexto proposto anteriormente na Seção 3.2. Assim, o projetista não precisa se preocupar como ocorre a coleta dos dados, mas somente informar os dados necessários. O Serviço de Contexto se encarrega da comunicação com os agentes que fornecem estas informações.

A Figura 27 mostra a infra-estrutura de suporte do CR-RIO integrada com o Serviço de Contexto, onde se pode ver o *Contractor* consultando este serviço, com o objetivo de monitorar as propriedades dos perfis no contrato. Posteriormente, na subseção 4.6 serão explicadas, mais detalhadamente, as mensagens trocadas entre os componentes do CR-RIO.

4.3 Utilização de Perfis para a Descoberta de Recursos

No *framework* CR-RIO são usados perfis para informar as restrições e os requisitos de contexto dos recursos necessários à execução dos serviços. Propomos reutilizar estes perfis para informar as restrições de contexto que devem ser satisfeitas durante a descoberta de recursos. Dessa forma, o sistema de descoberta é capaz de filtrar o resultado, retornando apenas os recursos que satisfazem às restrições do serviço. Por exemplo, dado o perfil de processamento da Figura 24, que é baseado na categoria de processamento mostrada pela Figura 22 (linhas 1-5), pode-se passar este perfil ao sistema de descoberta, de modo que ele somente retorne os recursos que tenham *clock* de processamento de, no mínimo, *800 MHz*, no máximo, *50%* de utilização e, no mínimo, *256 MB* de memória disponível.

```
1 profile {
2   Processing.utilization <= 50;
3   Processing.clockFrequency >= 800;
4   Processing.memFree >= 256;
```

```
5 } refProcessing;
```

Figura 24 - Perfil de Processamento

Antes de consultar o Serviço de Descoberta é necessário mapear os perfis descritos nos contratos para propriedades XML, de acordo com o *schema* proposto na Figura 17. Cada restrição descrita no perfil gera uma ocorrência do elemento *Attribute*. Por exemplo, o mapeamento do perfil apresentado na Figura 24 é mostrado na Figura 25. Visando simplificar o exemplo, é mostrado somente uma parte do documento XML resultante.

```
...
5  <Attribute Name=”utilization” op=”<=” Val=”50” />
6  <Attribute Name=”clockFrequency” op=”>=” Val=”800” />
7  <Attribute Name=”memFree”  op=”>=” Val=”256” />
...
```

Figura 25 - Mapeamento Perfil - XML

4.4 Perfil de Seleção

O Serviço de Descoberta pode retornar um conjunto contendo uma ou mais instâncias de recursos que satisfaçam as restrições da aplicação. Dessa forma, o CR-RIO deve ser capaz de inferir qual recurso, dentro do conjunto recebido, melhor satisfaz os requisitos da aplicação. Por exemplo, a consulta da Figura 25 retorna todos os recursos que satisfazem as restrições informadas, no entanto, a aplicação prefere aqueles que tenham a menor utilização CPU. Assim, é desejável ter uma forma disponível para filtrar o resultado, por exemplo através de uma função de seleção ou utilidade que permita, para cada propriedade individual de cada tipo de recurso, se especificar: (i) se ela deve ser maximizada ou minimizada, (ii) o peso para expressar sua importância relativa e (iii) uma ordem de preferência. Com este objetivo propomos o uso de um *perfil de seleção*, que permite descrever no nível da arquitetura um indicativo de preferência em relação aos recursos descobertos. Os parâmetros e sintaxe do perfil de seleção, que seguem o mesmo estilo dos perfis, permitem a especificação de pesos e prioridades para as propriedades relacionadas aos recursos

sendo descobertos. A sintaxe para o perfil de seleção é apresentada a seguir na notação BNF:

```
SELECAO   := selection { CORPO } ID;
CORPO     := DIRETIVA+
DIRETIVA  := NOMECATEGORIA.NOMEPROPRIEDADE : ([ORIENTACAO] PESO) |
                                                (ORDEMPREFERENCIA [PESO]);
ORIENTACAO       := maximize | minimize
ORDEMPREFERENCIA := (Valor > | ORDEMPREFERENCIA) | NULL;
PESO             := weight NUMBER;
ID               := STRING
NOMECATEGORIA    := ID
NOMEPROPRIEDADE  := ID
VALOR            := STRING | NUMBER
```

Um perfil de seleção se inicia com a palavra reservada *selection* e contém um corpo (CORPO) e uma identificação (ID). CORPO pode conter uma ou mais diretivas (DIRETIVA) de seleção de recursos. Uma diretiva pode especificar uma orientação de otimização (ORIENTACAO) ou uma ordem de preferência para os valores das propriedades (NOMEPROPRIEDADE) da categoria (NOMECATEGORIA). Cada diretiva também tem um peso (PESO) para indicar a preferência da aplicação por determinada propriedade.

A Figura 26 (a) (linhas 1-5) mostra um exemplo de perfil de seleção usado na aplicação de videoconferência na Seção 5.2. Na linha 1 tem-se a palavra reservada *selection* usada para indicar que o bloco a seguir contém a especificação de um perfil de seleção. Na linha 2, é especificado que a propriedade *CPUIdle* (CPU livre) da categoria de recurso *Processing* deve ser maximizada e tem o peso com valor 2. Na linha 3, é especificado que a propriedade *MemFree* (memória livre) da categoria de recurso *Processing* deve ser maximizada com peso no valor igual a 1. Não é obrigatório informar a política de otimização para as propriedades. O critério de seleção *default* considera a descrição da categoria para determinar se pretende maximizar ou minimizar o valor de uma propriedade. É assumido que propriedades do tipo *increasing* devem ser maximizadas e aquelas do tipo *decreasing* minimizadas. Com estas informações de peso, pode-se inferir que a aplicação tem preferência pelos

recursos com maior quantidade de processodor livre (*CPUIdle*), isto porque ela tem um peso duas vezes maior que o de memória livre (*MemFree*).

Para as propriedades não numéricas ou enumerações constantes, pode-se estabelecer uma ordem de preferência, como mostrado na função da Figura 26 (b) (linha 7) que é utilizada pela aplicação de vídeo sob demanda pervasiva na Seção 5.3. Neste exemplo, é estabelecida uma ordem de preferência para a propriedade *type* do recurso do tipo *Display*. O operador '>' indica da esquerda para a direita os valores preferenciais para a propriedade. No exemplo em questão, a aplicação informa que os tipos preferenciais, nesta ordem, são: Plasma, Projetor, monitor e em último caso o *palm*.

```
1 selection {
2      Processing.CPUIdle: maximize weight 2;
3      Processing.MemFree: maximize weight 1;                (a)
4 } hqRef;
5
6 selection {
7      Display.type = (Plasma > Projetor > monitor > palm);  (b)
8 } uDisplaySozinho;
```

Figura 26 - Exemplos de perfis de seleção

É importante ressaltar que o perfil de seleção é usado somente como a representação de um indicativo ou política de preferência que é passada como parâmetro para uma função de seleção ou utilidade específica, implementada como um *hot spot* da arquitetura. Por exemplo, o CR-RIO pode fazer a soma ponderada das propriedades e seus respectivos pesos obtendo um valor de utilidade para o recurso em questão. Assim, o recurso que tiver o maior valor numérico de utilidade seria avaliado como o melhor deles. Funções de seleção arbitrariamente sofisticadas, utilizando técnicas de otimização ou inteligência computacional podem ser também integradas a este esquema.

Em nossa proposta de integração com o CR-RIO, o perfil de seleção é utilizado por um novo elemento chamado de *Selector* (definido na Seção 4.7) para filtrar os recursos obtidos do Serviço de Descoberta. A representação da função deve ser inserida no contrato da aplicação, como descrito na seção a seguir. Posteriormente,

no Capítulo 5, serão discutidos alguns exemplos de utilização a partir de aplicações gerenciadas pelo CR-RIO.

4.5 Ligação Estática e Dinâmica de Recursos

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. Esta opção pode não ser aplicável em aplicações dinâmicas, cujas restrições foram discutidas anteriormente nas Seções 2.3 e 3.3. Nestes casos, a descrição arquitetural vai conter referências aos tipos ou classes de componentes e recursos e a ligação destas referências com artefatos de software e recursos reais será realizada dinamicamente em tempo de implantação. Para a ligação dinâmica dos recursos nas aplicações gerenciadas pelo CR-RIO, é necessário informar explicitamente no contrato que se quer este comportamento. Assim, o desenvolvedor deve especificar o contrato considerando os recursos "virtuais" sempre que necessitar que a ligação seja feita dinamicamente, de acordo com as restrições de contexto. Como exemplo, é utilizada uma entrada do contrato descrito na Figura 42.

```
(1)   link term to refUFF by commCon with hqTtermCProf;
```

Na linha acima (1) descreve-se em tempo de projeto, que o módulo *term*, uma instância da classe *Terminal* será ligada à *refUFF*, uma instância específica de um recurso da classe *Reflector*. Um dos problemas dessa abordagem é que, caso o perfil *termCProf* que restringe a ligação entre os dois elementos não seja satisfeito no momento da ligação, o serviço não poderá ser estabelecido. No entanto, poderia haver outras instâncias da classe *Reflector* (ou seja, outros refletores) sendo executadas em outras máquinas, que poderiam satisfazer as restrições do contrato. Surge, então, a necessidade de especificar o contrato em termos de recursos virtuais, que são recursos cujas referências são encontradas dinamicamente pelo Serviço de Descoberta durante a implantação do serviço. A seguir são mostradas as possíveis maneiras de transformar a chamada (1), de forma a utilizar recursos virtuais.

```
(2) link term to ref = Reflector by commCon with hqTermCProf
```

A linha acima (2) informa que o terminal será ligado à variável *ref* que faz referência a um recurso da classe *Reflector,* cuja instância é descoberta dinamicamente, em tempo de implantação. No momento em que o serviço referente a essa linha do contrato tentar ser estabelecido, a infra-estrutura deverá localizar uma instância de recurso da classe *Reflector* que satisfaça às restrições de contexto especificadas em *hqTermCProf*. No entanto, em alguns casos, pode ser preciso restringir o domínio dos recursos a serem descobertos e usados pelas aplicações. A linha a seguir descreve como essa restrição pode ser especificada.

```
(3) link term to ref = Reflector at teleconf by commCon with
                                                    hqTermCProf
```

A chamada anterior (3) informa que a aplicação somente deseja instâncias de recursos da classe *Reflector,* localizadas no domínio *teleconf*. Caso nenhum domínio seja informado, o CR-RIO considera como *default,* o domínio local de operação. Com essa informação de domínio o *framework* submete as consultas diretamente para o Serviço de Descoberta localizado no domínio em questão.

Em alguns casos podem-se ter inúmeras referências da classe de recurso *Reflector,* que satisfazem às restrições de domínio e de contexto requeridas. Assim, é preciso ter uma maneira de especificar um critério de seleção ou utilidade empregado para guiar a classificação das referências obtidas do Serviço de Descoberta. A linha a seguir mostra como o perfil de seleção é usado com este objetivo.

```
(4) link term to ref = Reflector at teleconf select(hqref) by
                                          commCon with hqTermCProf
```

Nesta última chamada (4), além da infra-estrutura realizar as operações descritas para as chamadas (2) e (3), ela vai classificar o conjunto de instâncias de recursos obtido do Serviço de Descoberta, por meio de um perfil de seleção (cuja sintaxe foi discutida na Seção 4.4) representado por *hqRef* (apresentada na Seção 5.2.5) . Assim, pode-se escolher a melhor instância contida no conjunto de recursos.

Nos casos onde não é usado o perfil de seleção, o CR-RIO escolherá, aleatoriamente, uma instância qualquer do conjunto retornado pelo Serviço de Descoberta.

```
(5) link term to ref = Reflector select*(hqref) by
                                          commCon with hqTermCProf
```

A chamada acima, (5) apresenta uma variação no uso de perfis de seleção, *select**. A semântica de *select** difere da semântica de *select* da seguinte maneira: *select** é usado quando a aplicação precisa selecionar continuamente os recursos associados na arquitetura, segundo os critérios do perfil de seleção. De uma forma geral, usando *select**, sempre que um recurso melhor que o corrente tornar-se disponível e for descoberto pelo Serviço de Descoberta, o *Contract Manager* é notificado e reimplanta o serviço corrente com o novo recurso. Para isso, propomos a introdução um elemento especial na infra-estrutura de suporte de CR-RIO que seria encarregado de notificar uma "violação" do serviço corrente por ter obtido a referência de um recurso ainda mais apto que o atual (ao contrário dos *Contractors* que notificam violações quando um perfil não pode ser atendido). Mais detalhes de funcionamento do *select** são discutidos posteriormente na Seção 4.7.

4.6 Interação com os Componentes do Framework

A Subseção 4.1 apresentou a arquitetura geral do *framework* CR-RIO e explicou o funcionamento de seus componentes e a interação entre eles. Esta subseção contém a proposta de integração do Serviço de Descoberta e do Serviço de Contexto na infra-estrutura de suporte do CR-RIO.

A Figura 27 mostra uma versão modificada da Figura 23 indicando como os Serviços de Contexto (*Context Service*) e de Descoberta (*Resource Discovery*) encaixam-se na infra-estrutura de suporte do CR-RIO. O Serviço de Contexto atua como um *proxy* para os Agentes de Recurso. De acordo com a proposta desse trabalho, o Serviço de Contexto deve encobrir todos os detalhes de comunicação com os Agentes de Recurso. Antes de utilizar o Serviço de Contexto, o *Contractor* era obrigado a saber quais agentes contatar e como se comunicar com cada um deles. A partir de agora ele somente precisa consultar os tipos de recurso e suas respectivas

propriedades de contexto das quais ele quer obter os valores. Ao receber a consulta, o Serviço de Contexto encarrega-se de buscar os valores para as propriedades requeridas.

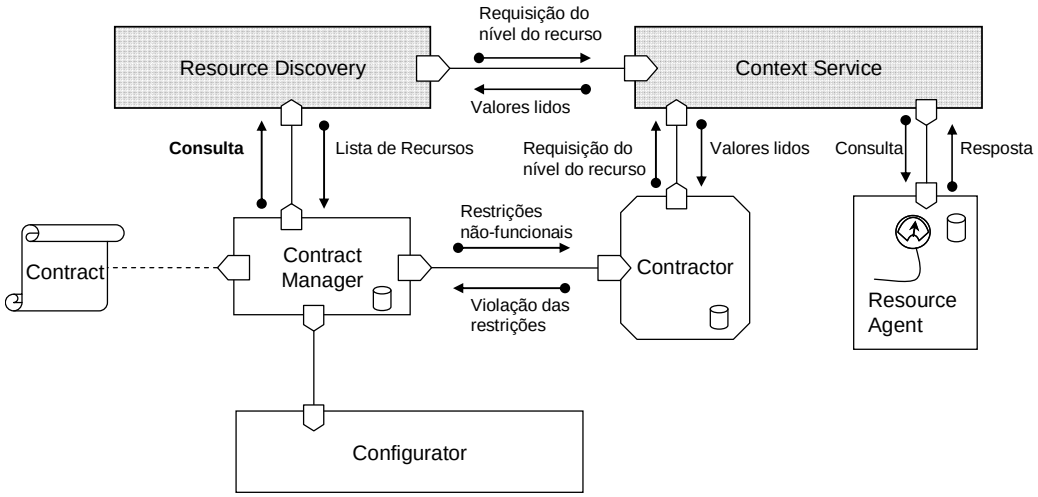


Figura 27 - Interação com os Elementos de Suporte do CR-RIO

A utilização do Serviço de Contexto permite aumentar a abstração do processo de obtenção do estado de contexto dos recursos porque o *Contractor* não interage diretamente com o Agente de Recurso. Agora ele se baseia em um Serviço de Contexto sobre o qual realiza as consultas em termos de tipos de recursos e propriedades de contexto utilizando as referências enviadas pelo *Contract Manager* na etapa de implantação do serviço.

Também na Figura 27, é mostrado o Serviço de Descoberta, responsável pela descoberta dinâmica dos recursos. Depois de interpretar o contrato e selecionar o serviço a ser estabelecido, o Gerenciador de Contratos identifica quais recursos ele precisa localizar e as restrições de contexto que estes devem satisfazer, para depois consultar o Serviço de Descoberta. Como discutido anteriormente, essas restrições de contexto, são as informações contidas nos perfis de cada serviço. Caso seja informado um perfil de seleção, a lista de recursos obtida do serviço de descoberta deve ser classificada com o objetivo de selecionar o melhor deles.

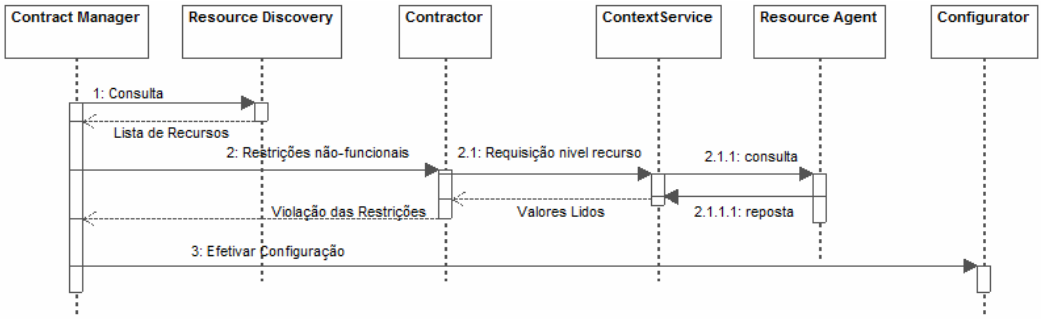


Figura 28 - Diagrama de Iteração entre os componentes do CR-RIO

Para ilustrar a interação entre os componentes do framework utilizaremos a entrada do contrato (2) da Seção 4.5. A Figura 28 mostra o diagrama de seqüência das mensagens trocadas entre os elementos da infra-estrutura entre o início da consulta ao Serviço de Descoberta até a efetivação da configuração do recurso. No passo 1, ao interpretar o contrato e selecionar o serviço de maior prioridade, o *Contract Manager* identifica a classe de recurso (*Reflector*) que deve ser ligado dinamicamente e submete uma consulta ao Serviço de Descoberta (*Resource Discovery*). Este retorna uma lista de recursos (instâncias de *Reflector*) para o *Contract Manager*. Como nesta entrada não é especificado um perfil de seleção, o *Contract Manager* escolhe um recurso aleatoriamente, dentre aqueles retornados pelo Serviço de Descoberta. Posteriormente, no passo 2, repassa-se ao *Contractor* a referência ao recurso a ser efetivamente utilizado e as restrições não-funcionais que devem ser obedecidas no nó sob seu controle. No passo 2.1, o *Contractor* submete uma consulta ao *Context Service* que, por sua vez, consulta os Agentes de Recurso (*Resouce Agents*) apropriados e retorna os valores das propriedades requeridas. Ao receber o estado do recurso, o *Contractor* verifica se ela viola ou não alguma restrição dos perfis. Finalmente, no passo 3, o *Contract Manager* envia uma mensagem para o *Configurator* efetivar a configuração do serviço.

Se o serviço corrente for invalidado, o *Contract Manager* identifica o próximo serviço de acordo com a ordem de prioridade descrita no contrato e repete os passos descritos na Figura 28. A invalidação do serviço corrente acontece quando as restrições dos perfis não são satisfeitas, ou quando o nível de operação do recurso atual ou outro qualquer encontrado pelo Serviço de Descoberta, satisfaz as restrições contidas nos perfis de um serviço de maior prioridade.

4.7 Utilização do Componente Selector

Para os casos onde é utilizado um perfil de seleção ou quando a aplicação precisa monitorar continuamente os recursos associados à arquitetura, é necessário utilizar um novo componente denominado *Selector*. A Figura 29 apresenta a proposta de integração do *Selector* à infra-estrutura de suporte de CR-RIO, em conjunto aos Serviços de Contexto e Descoberta.

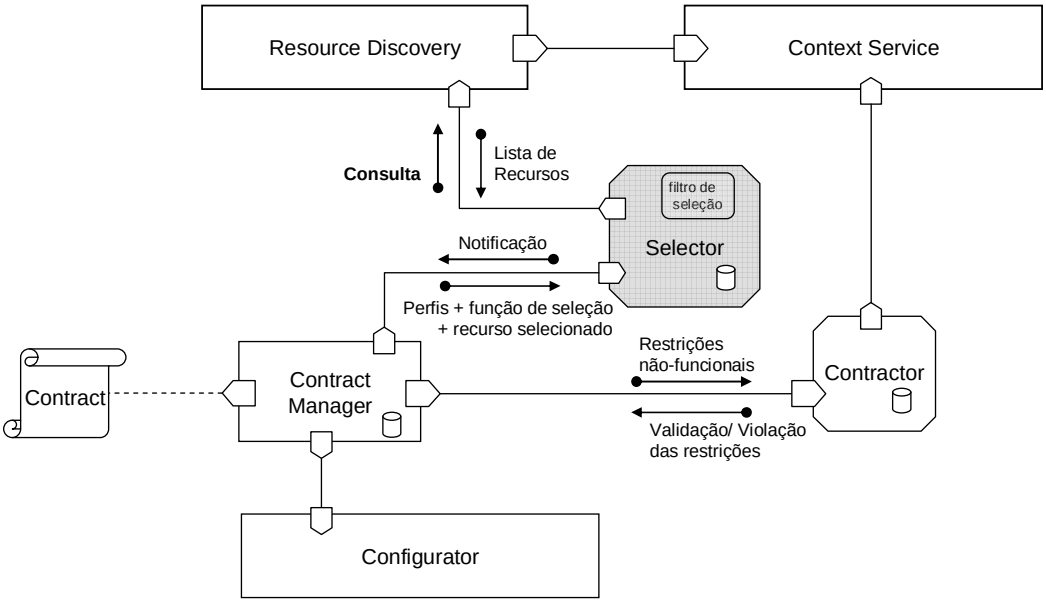


Figura 29 - A integração do Selector ao Serviço de Descobertas

Para exemplificar a integração do *Selector*, utilizaremos a entrada de contrato (5) mostrada na Seção 4.5. Para executar a chamada em questão, os seguintes passos são executados pelo CR-RIO de acordo com o digrama de seqüência da Figura 30:

1. O *Contract Manager* interpreta o contrato com uma cláusula arquitetural contendo *select** e solicita ao *Selector* que descubra um recurso. Para isso ele precisa informar: (i) a classe do recurso (*Reflector*); (ii) os perfis (*hqTermCProf*); e (iii) perfil de seleção (*hqRef*);

2. O *Selector* faz o pedido ao Serviço de Descoberta;

3. *Selector* recebe a lista de recursos que são filtrados e classificados usando uma função de seleção que recebe como parâmetro as informações contidas em *hqRef*.

Com isso, o *Selector* pode inferir qual o "melhor" recurso contido na lista obtida do Serviço de Descoberta.

4. O *Selector* armazena as informações do recurso selecionado usando a função de seleção e as repassa para o *Contract Manager* que contacta os *Contractors* e decide se implanta ou não o serviço, chamando o *Configurator*.

5. Após a implantação do serviço, o *Selector* continua consultando periodicamente o Serviço de Descoberta usando as informações de classe, perfil, e parâmetros de seleção do recurso a ser descoberto.

6. Os passos 2 e 3 são repetidos. Só que, ao invés do passo 4, o *Selector* compara o melhor recurso selecionado desta vez, com os dados do recurso selecionado anteriormente. Para isso, o *Selector* sempre mantém armazenado os dados do recurso sendo usado pelo serviço em execução.

7. Caso o melhor recurso encontrado seja o mesmo que esta sendo usado pelo serviço em execução, nada precisa ser feito. Caso contrário, o *Selector* notifica o *Contract Manager* que o serviço atual será invalidado, visto que existe um recurso melhor que o atualmente utilizado, e junto com a notificação repassa a referência ao novo recurso.

8. *Contract Manager* tenta reimplantar o serviço atual usando o novo recurso encontrado nos passos 6 e 7.

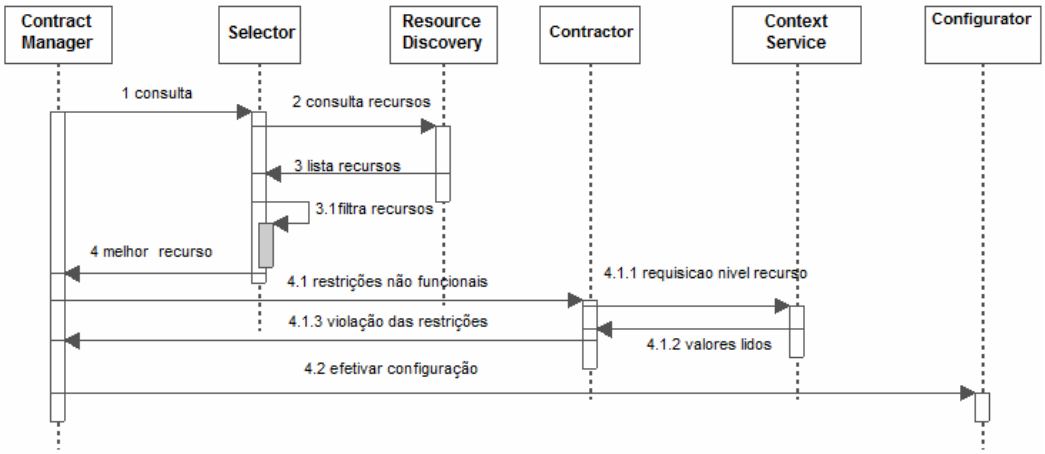


Figura 30 - Diagrama de Iteração entre os componentes do CR-RIO e o Selector

Observa-se que o *Selector* tem papel semelhante ao do *Contractor* no sentido de verificar continuamente propriedades dos recursos utilizados. As consultas do

Selector são realizadas de forma semelhante as do *Contract Manager*, com as mesmas informações (tipo do recurso a ser descoberto e propriedades extraídas dos perfis). O mecanismo de notificação dos *Contractors* para o *Contract Manager* pode ser reusado para o *Selector*. Ao receber uma notificação do *Selector*, o *Contract Manager* interrompe o serviço corrente e reimplanta o mesmo, configurado com o novo recurso descoberto.

Esta funcionalidade atende a um domínio de aplicações em que existem vários recursos disponíveis do mesmo tipo, em que os mesmos apresentam grande variação de em suas propriedades. Por exemplo, uma aplicação pervasiva em que PDAs, precisam estar continuamente conectados a uma base de dados através de redes WiFi infra-estruturadas (com pontos de acesso). Na medida em que o usuário do PDA se desloca pelo ambiente, os sinais de vários pontos de acesso são capturados, cada um deles com um nível diferente. Um serviço contendo a seleção contínua (*select**) poderia garantir que o PDA estaria sempre conectado ao ponto de acesso que oferecesse melhores características de conectividade (melhor sinal, menor taxa de erros, maior velocidade, etc.).

4.8 Reorganização do Repositório de meta-nível

A organização do repositório de meta-nível do CR-RIO foi estudada e modificada para a integração do Serviço de Descoberta. Até então, como não havia o suporte explícito à descoberta de recursos no *framework* CR-RIO, este repositório era responsável por manter todas as informações dos recursos (dispositivos, serviços, etc) de interesse das aplicações. Com a inserção do Serviço de Descoberta, o repositório de meta-nível não precisa armazenzar as informações relacionadas aos recursos. Assim, ele passa a ser responsável somente pelas informações arquiteturais da aplicação (interface, ligações entre módulos, e referências).

A partir dessa reorganização do repositório de meta-nível, tem-se uma melhor separação de interesses oferencendo maior flexibilidade e reuso das informações associadas aos recursos. Isto porque o Diretório de Recursos do Serviço de Descoberta é desaclopado ao CR-RIO. Dessa forma, as informações de recursos podem ser compartilhadas, por exemplo, com outros *frameworks* de gerenciamento de aplicações sensíveis ao contexto.

4.9 Conclusão do Capítulo

Inicialmente foram apresentados detalhes do funcionamento do *framework* CR-RIO, que utiliza uma abordagem baseada no conceito de arquitetura de software para especificar e gerenciar aplicações com requisitos não-funcionais. O CR-RIO permite descrever contratos de qualidade de serviço, que especificam as restrições não-funcionais desejadas pelos serviços de uma aplicação, através da linguagem CBabel. Para suportar o gerenciamento desses contratos, o *framework* possui uma infra-estrutura de suporte composta por componentes que, através da interpretação de um contrato, gerenciam as restrições de qualidade requeridas por uma aplicação.

Posteriormente, mostrou-se a proposta de como o Serviço de Contexto e o Serviço de Descoberta podem ser integrados na infra-estrutura de suporte do CR-RIO. Foi proposta uma forma de descrever o contrato das aplicações em termos de recursos virtuais, que devem ser descobertos dinamicamente, e também como a infra-estrutura de suporte do CR-RIO faz essa descoberta. O suporte à descoberta de recurso permite ao desenvolvedor especificar restrições de domínio dos recursos e utilizar perfis de seleção usados para escolher o melhor recurso a ser utilizado.

O próximo capítulo apresenta exemplos de aplicação que ajudam aprofundar alguns detalhes da integração dos Serviços de Descoberta e Contexto com o *framework* CR-RIO e permitem validar a proposta.

5 Exemplos de Aplicação

Com o objetivo de validar nossa proposta, este capítulo apresenta exemplos de aplicações com restrições de contexto e demonstra como os Serviços de Contexto e de Descoberta, integrados ao *framework* CR-RIO, são utilizados para gerenciá-las, de acordo com as políticas de qualidade especificadas nos contratos definidos para essas aplicações.

São descritos três aplicações: (i) vídeo sob demanda com servidores replicados, (ii) videoconferência, e (iii) vídeo sob demanda em ambiente pervasivo. Estes exemplos foram selecionados com o objetivo de explorar as diversas características dos Serviços de Contexto e de Descoberta como elementos importantes

no gerenciamento de aplicações “tradicionais” e pervasivas, que possuem tipos distintos de restrições de contexto.

5.1 Aplicação de Vídeo sob Demanda (VoD) Dinâmica

O cenário ilustrado neste exemplo é baseado na aplicação de VoD apresentado em [Corradi 2005], que foi estendida para suportar a descoberta dinâmica do servidor de vídeo. A arquitetura da aplicação é constituída de réplicas de servidores multimídia os quais contém uma coleção de vídeos armazenados no formato MPEG-2 e de clientes que exibem fluxos de vídeo recebido dos servidores. Os clientes podem ser de diferentes plataformas, desde dispositivos portáteis até estações de trabalho, criando um cenário heterogêneo no qual são oferecidos diferentes níveis de disponibilidade de recursos, tais como CPU e largura de banda. Neste contexto, é necessário um mecanismo para transformar os vídeos MPEG-2 em formatos alternativos (e.g., H.261, H263) que exijam menos recursos de processamento e de rede do cliente. Este último também deve ser capaz de descobrir dinamicamente os servidores que oferecem os melhores níveis de serviço em termos de perfis especificados no contrato.

5.1.1 Cenário Base

A Figura 31 mostra parte do cenário da aplicação em mais detalhes. Existe um cliente que quer visualizar um vídeo que está armazenado em três servidores replicados (*VideoServer1, VideoServer2, VideoServer3*) os quais pertencem a diferentes domínios administrativos (UFF e UERJ). O primeiro problema que o cliente deve resolver é a escolha do melhor servidor levando em consideração alguma métrica. A primeira idéia seria utilizar o servidor com maior largura de banda e menor atraso. No entanto, pode ser que o cliente não tenha permissão de acesso a todos os servidores em todos os domínios administrativos. Por exemplo, suponha que ele somente possa acessar os servidores localizados no domínio UFF. Isso faz com que a busca do melhor servidor fique restrita aos recursos contidos neste domínio.

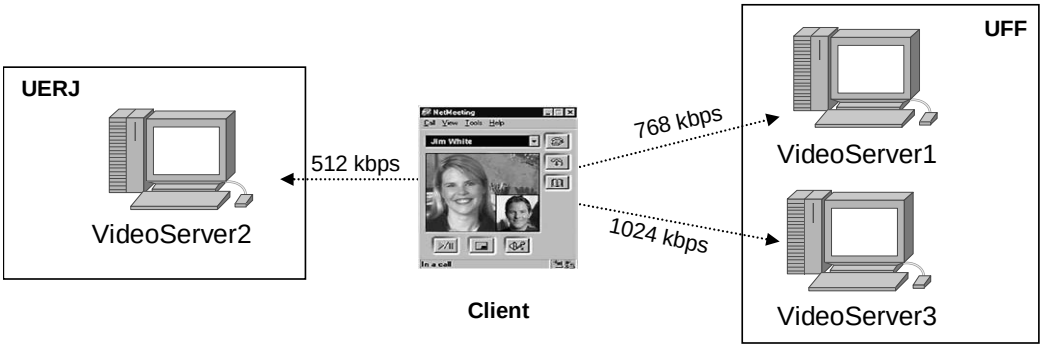


Figura 31 - Cenário da aplicação de vídeo sob demanda dinâmica

Também há o caso em que o cliente possui pouca disponibilidade de recursos de largura de banda ou de processamento local. Nesse caso, a arquitetura deve ser adaptada a este contexto, utilizando-se, por exemplo, um transcodificador que converta o vídeo, no nó do servidor, antes de ser transportado pela rede, para uma codificação com qualidade de vídeo inferior diminuindo as restrições de processamento e de rede, possibilitando sua visualização pelo cliente.

Para esta aplicação foi estabelecido que um fluxo de vídeo deve ser transmitido preferencialmente no formato MJPEG (*Motion* JPEG), que possui melhor qualidade, porém requer mais recursos do canal de comunicação para a sua transmissão e dos clientes para efetuar o seu processamento na exibição. Caso os recursos sejam insuficientes para esse formato, o vídeo pode ser transcodificado, por exemplo, para o formato H.261, que possui qualidade inferior ao MJPEG e exige menos recursos para sua transmissão e processamento.

Um dos objetivos deste estudo de caso é mostrar como as restrições citadas anteriormente podem ser descritas no contrato da aplicação e como a infra-estrutura de suporte do CR-RIO trata essas situações.

5.1.1.1 Arquitetura da Aplicação

A arquitetura básica da aplicação é apresentada na Figura 32, onde são identificados dois tipos diferentes de clientes: (a) com alta disponibilidade de processamento acessando os servidores através de uma LAN (*Local Area Network*), e (b) com disponibilidade de processamento regular, acessando os servidores através de um modem e uma linha telefônica convencional. Para este último caso, um conector

responsável pela transcodificação do formato MPEG-2 para o H.261 é inserido entre o cliente e o servidor.

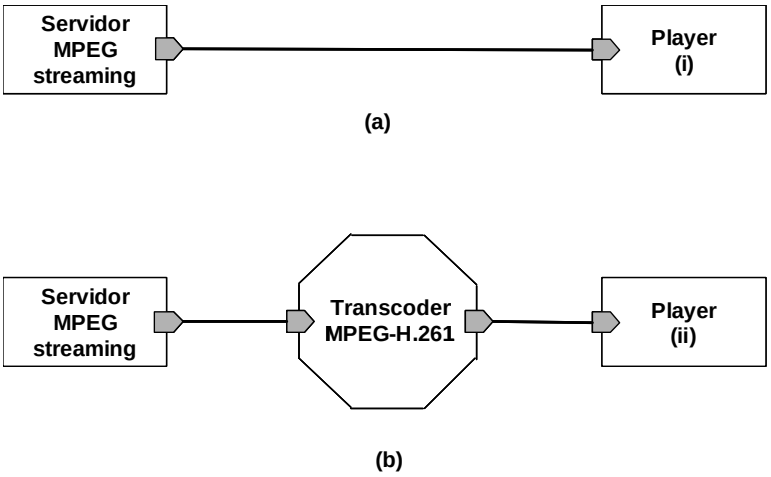


Figura 32 - Arquitetura da aplicação vod

A Figura 33 descreve a arquitetura dessa aplicação através da ADL CBabel. Nela, são descritos os módulos *Client* e *VideoServer* (linhas 3 a 8) e o conector responsável pela transcodificação do formato de vídeo (linhas 9 a 12). Na linha 13, o módulo servidor e o conector H261 são instanciados em um determinado nó (*HostServidor*).

```
1  module Cliente_Servidor {
2      port provide, request;
3      module Client {
4            out port request;
5      } player;
6      module VideoServer {
7            in port provide;
8      } videoServer;
9      connector Transcoder {
10           in port request;
11           out port provide;
12     } H261;
13     instantiate videoServer, H261 at serverHost;
14  } vod;
```

Figura 33 - Descrição da arquitetura da aplicação VoD

A partir dessa descrição, um ambiente de suporte a configuração pode implantar o servidor no nó desejado. Depois disso, o servidor multimídia estaria pronto para receber requisições por vídeos dos clientes da aplicação.

5.1.2 Categorias e Contratos de QoS

Para especificar os serviços da aplicação VoD e seus respectivos níveis de qualidade requeridos, o próximo passo é a descrição do contrato de qualidade de serviço. A Figura 34 descreve esse contrato, orientando o *framework* CR-RIO no gerenciamento dessa aplicação, de acordo com a qualidade requisitada para seus serviços.

O contrato Vod (linhas 1 a 18) descreve dois serviços. O primeiro, *sMPEG_video* (linhas 2 a 6), instancia um cliente de vídeo (linha 3), caso seu recurso de processamento satisfaça as restrições especificadas no perfil *cpu_mpeg* (linhas 20 a 23), e o conecta a um servidor de vídeo (linha 4) localizado dinamicamente no domínio UFF, de forma que o canal de comunicação entre o cliente e o servidor satisfaça as restrições declaradas no perfil *transport_mpeg* (linhas 25 a 28). Se essas restrições forem satisfeitas o serviço será estabelecido e o cliente receberá o vídeo no formato preferencial (MPEG-2). O serviço *sH261_video* (linhas 8 a 12) requer menos recursos, pois, se estabelecido, transmitirá o vídeo no formato inferior H.261. Para que o vídeo seja transmitido nesse formato, o cliente é conectado ao servidor pelo conector H261 (linha 10), cuja função é transcodificar o vídeo do formato de maior para o de menor qualidade. As restrições referentes aos recursos de processamento e transporte para esse segundo serviço são descritas nos perfis *cpu_h261* e *transport_h261* (linhas 30 a 38). Os serviços *sMPEG_video* e *sH261_video* não contém perfis de seleção para escolher o melhor servidor de vídeo. Assim, será usado qualquer servidor retornado pelo Serviço de Descoberta e que satisfaça as restrições de qualidade especificadas nos perfis de cada um dos serviços.

Na cláusula de negociação de serviços do contrato Vod (linhas 14 a 17), o serviço *sMPEG_video* é descrito como o de maior prioridade (linhas 15), sendo que o outro serviço (*sH261_video*) só poderá ser estabelecido caso as restrições do primeiro não possam ser atendidas (*not*). Já na regra de negociação para o serviço *sH261_video*

(linha 16), é definido que, se ele estiver estabelecido e em um determinado momento o serviço *sMPEG_video* puder ser provido, ele será, pois é o de maior preferência.

```
1 contract {
2     service {
3           instantiate Player as client at HostCliente with cpu_mpeg;
4           link client to ref = VideoServer at uff with
5                                           transport_mpeg;
6     } sMPEG_video;
7
8     service {
9           instantiate Player as client at HostCliente with cpu_h261;
10          link client to ref = VideoServer at uff by H261 with
11                                          transport_h261;
12    } sH261_video;
13
14    negotiation {
15          not sMPEG_video -> sH261_video;
16          sH261_video -> sMPEG_video;
17    };
18 } Vod;
19
20 profile {
21    Processing.clockFrequence >= 700 MHz;
22    Processing.utilization <= 50 %;
23 } cpu_mpeg;
24
25 profile {
26    Transport.delay <= 50 ms;
27    Transport.bandwidth >= 1 Mbps;
28 } transport_mpeg;
29
30 profile {
31    Processing.clockFrequence >= 266 MHz;
32    Processing.utilization <= 70 %;
33 } cpu_h261;
34
35 profile {
36    Transport.delay <= 200 ms;
37    Transport.bandwidth >= 0,056 Mbps; // 56 Kbps
38 } transport_h261;
```

Figura 34 - Contrato de QoS para a aplicação VoD

5.1.3 Gerenciamento do Contrato

Para que o contrato possa ser gerenciado pelo CR-RIO, é preciso implementar (caso eles não existam) os Agentes de Recurso específicos aos recursos de processamento e transporte requeridos pela aplicação e registrá-los no Serviço de Contexto. Também e necessário codificar o *Contractor* que gerenciará a monitoração feita por meio do Serviço de Contexto, verificando se as propriedades monitoradas validam as restrições descritas no contrato. O Serviço de Descoberta provê a infra-estrutura básica para localizar um dentre os vários servidores replicados de forma a obter o vídeo daquele que tenha melhor disponibilidade. Mesmo quando não há vários servidores replicados, o sistema de descoberta evita a configuração estática do endereço do servidor. Isso pode tornar mais fácil a manutenibilidade do sistema. A Figura 35 ilustra a configuração dos componentes da infra-estrutura de suporte do CR-RIO para o gerenciamento do contrato para a aplicação em questão.

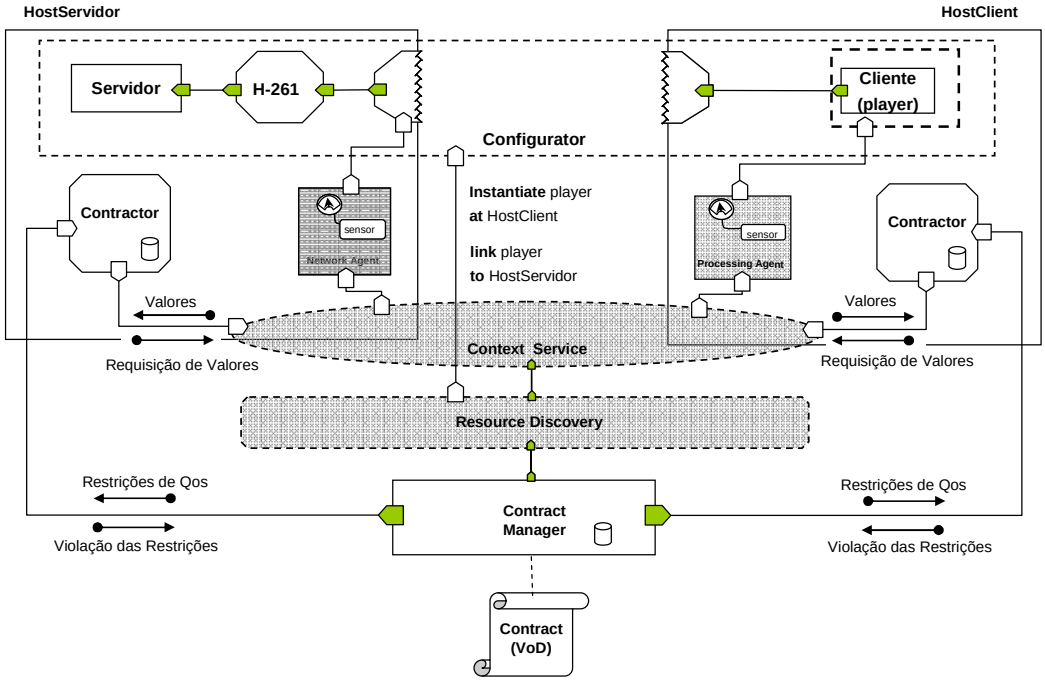


Figura 35 - Configuração dos componentes do CR-RIO para o gerenciamento do contrato para a aplicação VoD.

No gerenciamento desse contrato (*Vod*), o primeiro passo é a interpretação de sua descrição pelo *Contract Manager* (CM). Obtida as informações do contrato, o CM identifica na cláusula de negociação de serviços que *sMPEG_video* é o serviço de

maior preferência. Logo em seguida o CM deverá identificar as máquinas que hospedarão o cliente e o servidor de vídeo. Para este exemplo, a localização do cliente é definida estaticamente, e para o servidor de vídeo, o CM repassa as informações de classe de recurso, perfis e perfis de seleção para o *Selector* que tentará descobrir um servidor que fique dentro do perfil de transporte necessário para estabelecer o serviço em questão. A linha 4 do código da Figura 34 é convertida para uma chamada ao Serviço de Descoberta do domínio alvo, que no exemplo é UFF.

A consulta submetida ao Serviço de Descoberta é mostrada na Figura 36 (a) e na linha 7 é informada a propriedade *from* que especifica o recurso a partir do qual as medidas de rede devem ser feitas, que no exemplo em questão é o endereço do cliente de vídeo. Essa chamada deve retornar um servidor de vídeo que fique dentro do perfil de transporte repassado. Um exemplo de resposta é mostrada na Figura 36 (b). Caso não seja encontrado pelo menos um servidor, o CM deve tentar estabelecer o próximo serviço no contrato. Após identificar os recursos que farão parte da instância da aplicação, o CM deverá monitorar as restrições conjuntas para esse serviço (perfis *cpu_mpeg* e *transport_mpeg*). Para isso o CM envia essas restrições para os *Contractors* responsáveis pela verificação em cada um desses recursos. Para essa aplicação, há um *Contractor* na máquina do cliente, responsável pela verificação das propriedades do recurso processamento, e outro na máquina do servidor, para verificação das propriedades do recurso transporte. A verificação das propriedades do recurso transporte poderia ser feita tanto no cliente quanto no servidor. Esses *Contractors* recebem, então, essas restrições e solicitam ao Serviço de Contexto, os valores atuais das propriedades dos recursos, verificando se eles satisfazem as restrições para o serviço *sMPEG_video*. Caso as restrições sejam satisfeitas, o CM é notificado que esse serviço pode ser estabelecido e para isso o CM envia a configuração arquitetural desse serviço (linhas 3 e 4 da Figura 34) ao *Configurator*, que se encarrega de estabelecer a configuração exigida para o provimento do serviço.

```
1<ResourceQuery type=”VideoServer”>
2  <MaxResults>5</MaxResults>
3    <Constraints From=”Transport”>
5       <Attribute Name=”delay” op=”<=” Val=”50”/>                    (a)
6       <Attribute Name=”bandwidth” op=”>=” Val=”1000”/>
7    </Constraints>
8 </ResourceQuery >
```

```
9
10<DiscoveryResponse Type=”VideoServer” >
11  <ResourceInfo URI=”wokstation.ic.uff.br” IP=”192.168.1.1”>
12    <Attributes From=”Transport”>
13      <Attribute Name=”Bandwidth”  Val=”1250”/>                    (b)
14      <Attribute Name=”Delay”      Val=”48”/>
15    </Attributes>
16  </ResourceInfo>
```

Figura 36 – Consulta e resposta de descoberta dos servidores de vídeo do serviço sMPEG_video

Depois que o serviço estiver estabelecido, as propriedades dos recursos continuam sendo periodicamente monitoradas pelo *Contractor* e, caso o valor de uma das propriedades viole as restrições para o serviço corrente, o *Contractor* notifica isso ao CM, que tenta o estabelecimento do próximo serviço (*sH261_video*). Por exemplo, se o serviço *sMPEG_video* está sendo provido e, em um determinado momento, a utilização da CPU do cliente atinja o valor de 60%, violando assim a restrição de qualidade para esse serviço (linha 20 da Figura 34), o serviço corrente não poderá ser mantido. Esta situação faz com que o CR-RIO tente estabelecer outro serviço, de acordo com as cláusulas de negociação. De acordo com o contrato, o serviço *sH261_video* seria estabelecido, já que o valor da propriedade de processamento do recurso não viola a restrição especificada pelo perfil (linha 30 da Figura 34). Caso essa propriedade volte a um valor aceitável pelo serviço preferencial, este será restabelecido, desde que as outras restrições associadas a ele também sejam satisfeitas. Observa-se que para este exemplo, para todos os serviços descritos no contrato Vod, é necessário utilizar o Serviço de Descoberta. A Figura 37 (a) e (b) mostra a consulta (a) e resposta (b) do Serviço de Descoberta para o serviço *sH261_video*.

```
1<ResourceQuery type=”VideoServer”>
2  <MaxResults>5</MaxResults>
3    <Constraints From=”Transport”>
4       <Attribute Name=”delay” op=”<=” Val=”200” />              (a)
5       <Attribute Name=”bandwidth” op=”>=” Val=”560” />
6    </Constraints>
7 </ResourceQuery>
8
9 <DiscoveryResponse Type=”VideoServer” >
10  <ResourceInfo URI=”wokstation2.ic.uff.br” IP=”192.168.1.1”>
```

```
11    <Attributes From="Transport">
12      <Attribute Name="Bandwidth"  Val="756"/>                    (b)
13      <Attribute Name="Delay"      Val="170"/>
14    </Attributes>
15  </ResourceInfo>
```

Figura 37 - Consulta e resposta de Descoberta dos Servidores de Vídeo do serviço sH261_video

5.2 Aplicação de Videoconferência

Esta seção apresenta a aplicação de videoconferência que expõe mais detalhes da abordagem proposta. Inicialmente, são identificados os requisitos básicos desta aplicação e apresentado um contrato que descreve tais requisitos. Em seguida, discute-se a mesma aplicação com requisitos mais dinâmicos.

A aplicação considerada contém os elementos usuais: (i) um serviço de diretório/sessão, que faz o registro e o controle de sessões (salas) de videoconferência, tal como o *Gatekeeper* do padrão H323 [Toga 1999]; (ii) um elemento de redistribuição de fluxos de áudio e vídeo, equivalente ao MCU do padrão H.323 (sem a função de transcodificação), ou aos refletores de sistemas como o VRVS [Adamczyk et al 2003]; e (iii) terminais de usuário que fazem a captura, exibição e transmissão de mídias de áudio e vídeo.

5.2.1 Cenário de Base

O cenário base para a aplicação de videoconferência é constituído de usuários distribuídos em uma rede como a Internet, sendo uma rede *overlay* usada para prover comunicação multiponto, através de canais ponto-a-ponto. A arquitetura da rede *overlay* é formada por refletores, interligados por conectores de comunicação, que fazem o encaminhamento de fluxos de áudio e vídeo.

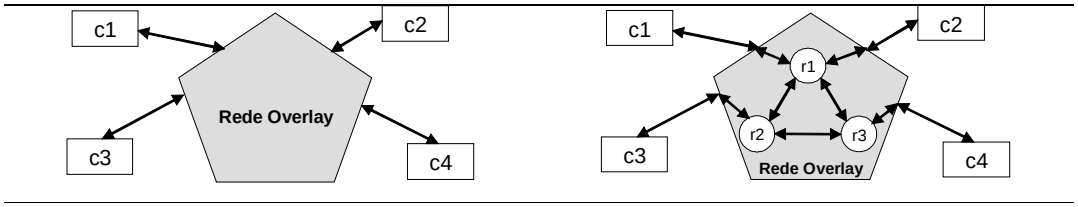


Figura 38 - (a) Arquitetura da aplicação; (b) Rede *overlay* detalhada

A Figura 38 (a) ilustra a arquitetura da aplicação, com usuários (terminais) que desejam participar de uma sessão de videoconferência utilizando a rede *overlay*. A rede *overlay* é "enxergada" como uma entidade arquitetural única, permitindo diferentes formas de configuração e adaptação, tais como: (i) para um conjunto estático de clientes participantes de uma sessão, selecionar um conjunto de refletores visando minimizar o retardo da comunicação entre os participantes; (ii) para um conjunto dinâmico de clientes, na admissão de um novo cliente, o suporte pode selecionar, do conjunto de refletores disponíveis, aquele que, baseado em alguma política, otimize parâmetros de desempenho, e.g., atraso e vazão de rede. A Figura 38 (b) mostra mais detalhes da arquitetura da rede *overlay*, constituída de três refletores *r1*, *r2* e *r3* instanciados em diferentes domínios administrativos. Os clientes *c1* e *c2* estão ligados à rede *overlay* pelo refletor *r1* e os clientes *c3* e *c4* através dos refletores *r2* e *r3*, respectivamente.

Além dos aspectos funcionais da aplicação (componentes, topologia de interconexão, divisão de funcionalidade, etc.) existe interesse em aspectos não-funcionais de contexto, relacionados a requisitos operacionais, tais como as qualidades dos fluxos de vídeo e áudio requeridas pelo usuário, e a capacidade da rede *overlay* encaminhar fluxos de várias sessões. Neste contexto, tais requisitos devem ser especificados, impostos pelo ambiente de suporte, e também monitorados, conforme discutido na Seção 2.4. No *framework* CR-RIO isso é obtido com o estabelecimento de contratos associados a diferentes partes da arquitetura, especificamente: a rede *overlay*, o canal de comunicação entre o terminal e a rede *overlay*, e os recursos locais onde os terminais são executados.

5.2.2 Contrato para a Rede *Overlay*

Ao se planejar a rede *overlay* um contrato deve ser definido. Este contrato especifica os recursos mínimos necessários para a execução dos refletores, (CPU e memória) e os recursos mínimos para os canais de comunicação interligando os mesmos. No caso de uma rede *overlay* implantada com o objetivo de prover um serviço robusto, os refletores seriam instalados em máquinas dedicadas, com recursos suficientes. Neste caso, a especificação dos requisitos relacionados a recursos locais funcionaria como um controle de admissão, para assegurar que os recursos

necessários existem. No caso mais geral, em que nas máquinas existem outros sistemas concorrendo com o refletor, os estados destes recursos também são monitorados para verificar se estão dentro dos limites especificados. Este contrato é independente das sessões de videoconferência, bem como dos terminais individuais ligados à rede *overlay* sob seu controle.

```
1 contract {
2   service {
3    // instâncias dos refletores em 3 nós distribuídos
4    instantiate Reflector as refUFF at caueira.uff.br with refLProf;
5    instantiate Reflector as refUERJ at blackcat.uerj.br with refLProf;
6    instantiate Reflector as refLAMPADA at lampada.uerj.br with refLProf;
7    // ligação de pares de refletores
8    link refUFF  to refUERJ    by comSock with refCProf;
9    link refUERJ to refLAMPADA by comSock with refCProf;
10   } viaInternet;
11   negotiation {
12     not viaInternet -> out-of-service;
13   }
12 } cViaInternet;
```

Figura 39 - Contrato para a implantação da rede overlay

A Figura 39 apresenta um contrato para uma rede *overlay*. O serviço *viaInternet* (linha 2-10) apresenta os componentes da arquitetura e os requisitos não-funcionais associados: (i) instanciação dos refletores em máquinas específicas, obedecendo aos requisitos locais de cada refletor (no caso, o perfil *refLProf* – linhas 4-6); (ii) a ligação desses refletores, ponto-a-ponto, também de acordo com os requisitos de transporte necessários para prover os serviços (perfil *refCProf*, linhas 8-9), estabelecendo assim uma topologia de encaminhamento de dados através dos enlaces apropriados. A cláusula de negociação neste caso é bem simples (linha 11). Se o serviço *viaInternet* não puder ser implantado ou mantido, o mesmo torna-se indisponível. Num cenário mais robusto, poder-se-ia especificar no contrato configurações parciais que permitam manter o fornecimento do serviço, mesmo no caso de falhas de refletores.

A demanda por recursos locais para cada refletor é definida em termos de características de processamento e memória, especificadas no perfil *refLProf*, Figura 40 linhas 1-05. Neste perfil estão definidos valores para algumas características da

categoria *Processing*. Assim, por exemplo, ao se instanciar o refletor *refUFF* (linha 4, Figura 39) estas características são previamente verificadas. As características desejadas para os canais de comunicação, que interligam cada par de refletores, são definidas no perfil *refCProf* (linhas 7-10). O requisito de atraso (*delay*) é configurado para 150 ms que é o valor máximo de atraso em que se pode ter uma comunicação VoIP aceitável [NetPredict 2006]. Os requisitos de banda são configurados proporcionalmente à quantidade máxima de usuários e sessões simultâneas e o tipo de codificação de áudio e vídeo usados. No exemplo, definimos, a partir de uma estimativa, as características deste perfil somando a banda necessária para transportar os dados de áudio e vídeo, e multiplicando o resultado pelo número médio de usuários previsto.

```
1 profile {
2      Processing.utilization <= 50;
3      Processing.clockFrequency >= 2800;
4      Processing.memReq >= 512;
5 } refLProf;
6
7 profile{
8      Transport.delay <= 150; //ms
9      Transport.bandwith >= 16000;// Kbps
10 } refCProf;
```

Figura 40 - Perfil de processamento e comunicação para os refletores

Observa-se que nem sempre é possível fazer reserva de recursos. Na maioria dos sistemas é possível fazer apenas a alocação inicial dos recursos que podem então ser compartilhados, ou mesmo re-alocados, para aplicações de maior prioridade. Assim sendo, durante a operação, as características definidas podem ser monitoradas e, em caso de violação, mecanismos de adaptação dinâmica podem ser acionados, e.g., para introduzir um novo refletor na rede visando redistribuir a carga.

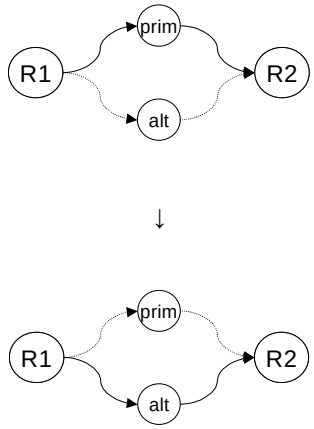
5.2.3 Tolerância a Falhas na Rede Overlay

No caso de saturação ou falha de um enlace ligando dois refletores, a ligação poderia ser re-configurada utilizando um enlace alternativo (mantido em *cold* ou *hot standby*) - Figura 41 (b). Este aspecto pode ser contemplado em um contrato

específico, que descreve as possíveis configurações de enlace entre dois refletores, como mostra a Figura 41 (a), para a rede *overlay* provida pelo serviço *viaInternet* (Figura 39 – linhas 2-10).

```
1  service {
2    link refUFF  to refUERJ  by
3           comSock1 with refCProf;
4    } primeLink;
5
6  service {
7    link refUFF  to refUERJ  by
8           comSock2 with refCProf;
9    } altLink;
10
11  negotiation {
12    not primeLink -> altLink;
13    altLink -> primeLink;
14    not altLink -> out-of-service; }
15 } cAltOvl;
```

(a)



(b)

Figura 41 - Contrato para a rede de *overlay* prevendo auto-reparo

O contrato da Figura 41 (a) descreve dois serviços de enlace. Através de conectores de comunicação diferentes (*comSock1* e *comSock2*) especifica-se que rotas independentes são utilizadas em cada serviço. A cláusula de negociação indica que o serviço preferencial é *primeLink* (linhas 1-4) que pode ser um enlace menos robusto e de baixo custo, como por exemplo, a internet. Se este serviço não estiver disponível, a arquitetura é reconfigurada para usar o serviço *altLink* (linhas 6-10) que utiliza um enlace dedicado de alta velocidade, porém com custo elevado. Se o serviço *primeLink* voltar a ser disponível, ele volta a ser estabelecido (linha 13). Se nenhum serviço puder ser implantado, esta parte passa a ser inoperante (linha 14), o que vai ser também refletido na rede *overlay*, como um todo. Na Seção 5.2.6 é decrito um experimento envolvendo o contrato da Figura 41, onde o serviço *primeLink* usa um enlance comum via internet e o serviço a*ltLink* usa um enlace de alta velocidade disponibilizado pelo projeto GIGA [Giga 2006].

Uma vez implantado e disponível o serviço de *overlay*, os terminais de usuário podem se conectar a esta rede para participar de uma sessão. O acesso de terminais é tratado em uma outra visão arquitetura.

5.2.4 Contrato Estático de Terminal

A instanciação de um terminal é regida pelo contrato descrito na Figura 42. Neste primeiro exemplo, o serviço *sSimpleTerm* contém os requisitos de recursos locais e de mídia, vinculados à instância *term* do módulo *Terminal*, descritos nos perfis *termLProf*, *termVProf* e *termAProf* (linha 3); as restrições para a comunicação estão descritas no perfil *termCProf*. Estes perfis são valorações das propriedades das categorias *Processing*, *Transport*, *VideoMedia* e *AudioMedia*, definidas previamente na Figura 22. Observa-se também que está definido que se o serviço *sSimpleTerm* não (mais) puder ser estabelecido, o terminal sairá de operação (linha 7).

```
1 contract {
2   service {
3     instantiate Terminal as term with termLProf, termVProf, termAProf;
4     link term to refUFF by commCon with termCProf;
5   } sSimpleTerm;
6   negotiation {
7     not sSimpleTerm -> out-of-service;
8   }
9 } cSimpleTerm;
```

Figura 42 - Contrato para um Terminal

A especificação de cada perfil é mostrada na Figura 43. Por exemplo, no perfil *termVProf* (linhas 13-18) está definido que o *codec* de vídeo a ser utilizado é o H261, a resolução desejada é QCIF, e o *frame rate* mínimo tolerado é de 14 fps. Assim, a implantação do serviço *sSimpleTerm*, que requer a instanciação do *Terminal* (linha 2, Figura 42) só será realizada se estas características puderem ser impostas.

```
1  profile {
2   Processing.cpuSlice >= 30;
3   Processing.utilization <= 40;
4   Processing.clockFrequency>= 400;
5   Processing.memReq >= 128;
6  } termLProf;
7
8  profile{
9   Transport.delay <= 80; //ms
10   Transport.bandwith >=128; //kbps
11  } termCProf;
```

```
12
13  profile {
14    VideoMedia.codec = H261;
15    VideoMedia.resolution = QCIF;
16    VideoMedia.frameRate: >= 14;
17    VideoMedia.quality = MEDIUM;
18  } termVProf;
19
20  profile {
21    AudioMedia.codec = DVI;
22    AudioMedia.sampleLenght = 8;
23    AudioMedia.sampleRate = 800;
24    AudioMedia.channels = MONO;
25  } termAProf;
```

Figura 43 - Perfis para um Terminal

5.2.5 Contrato Dinâmico de Terminal

Observa-se que no contrato anterior, para conectar o terminal à rede *overlay*, um dos três refletores disponíveis foi estaticamente selecionado (Figura 42, linha 4). Contudo, o usuário pode desempenhar um papel pró-ativo, quanto ao acesso ao serviço de videoconferência, usando um perfil de seleção como descrito nas Seções 4.4 e 4.5. Assim, a seleção do refletor de acesso pode ser feita através de políticas definidas nos perfis de seleção. Assim, ponderações sobre conjuntos de propriedades (e.g., *clockFrequency* e *utilization* da categoria *Processing*, *delay* e *bandwidth* da categoria *Transport*) orientariam a escolha do melhor refletor.

(b) Após a seleção inicial, durante a operação, o enlace ligando o *Terminal* ao refletor começa a apresentar um atraso maior do que o tolerado, ou uma banda menor do que a necessária. Prevendo este caso, o contrato pode incluir um serviço de menor qualidade que consome menos recursos (H261, por exemplo), viabilizando a continuidade da interação.

```
1  service {
2    instantiate Terminal as term with hqTermLProf, hqTermVProf,
3                                                      hqTermAProf;
4    link term to ref = Reflector at teleconf select(hqref) by commCon
5                                                        with hqTermCProf;
6  } sHQTerm;
```

```
7
8  service {
9    instantiate Terminal as term with lqTermLProf, lqTermVProf,
10                                                      lqTermAProf;
11    link term to ref by commCon with lqTermCProf;
12  } sLQTerm;
13
14  negotiation {
15    not sHQTerm -> sLQTerm;
16    not sLQTerm -> out-of-service;
17  }
```

Figura 44 - Serviços prevendo adaptação dinâmica de Terminal

Para esta nova versão do contrato, dois serviços são especificados (Figura 44):

sHQTerm: serviço preferencial, que permite a utilização de áudio e vídeo de alta-qualidade. A instanciação do módulo *Terminal* (linha 2) e a ligação do *Terminal* ao refletor (linha 3) são restritas por perfis dimensionados adequadamente.

sLQTerm: serviço de menor qualidade, a ser utilizado em último caso. Equivalente ao serviço *sHQTerm*, sendo que os perfis (*lqTermVProf, lqTermAProf*) contêm as restrições de propriedades com valores apenas suficientes para dar suporte a este serviço de qualidade inferior.

```
profile {
  Processing.utilization <= 30;
  Processing.clockFrequency >= 1600;
  Processing.memReq >= 512;
} hqTermLProf;

profile {
  VideoMedia.codec = MJPEG;
  VideoMedia.resolution = CIF;
  VideoMedia.frameRate >= 24;
  VideoMedia.quality = HIGH;
} hqTermVProf;

profile {
  AudioMedia.codec = G711;
  AudioMedia.sampleLenght = 16;
  AudioMedia.sampleRate = 32000;
  AudioMedia.channels = MONO;
} hqTermAProf;
```

```
profile {
  Processing.utilization <= 40;
  Processing.clockFrequency >= 400;
  Processing.memReq >= 128;
} hqTermLProf;

profile {
  VideoMedia.codec = H261;
  VideoMedia.resolution = QCIF;
  VideoMedia.frameRate = 14;
  VideoMedia.quality = MEDIUM;
} lqTermVProf;

profile {
   AudioMedia.codec = DVI;
   AudioMedia.sampleLenght = 8;
   AudioMedia.sampleRate = 800;
   AudioMedia.channels = MONO
} lqTermAProf
```

```
profile{
  Transport.delay <= 80;
  Transport.bandwith >= 1600;
} hqTermCProf;
```

(a)

```
profile{
  Transport.delay <= 80;
  Transport.bandwith >= 128;
} lqTermCProf;
```

(b)

Figura 45 - Perfis dos serviços (a) sHQTerm e (b) sLQTerm

Os perfis utilizados para indicar os requisitos dos serviços *sHQTerm* e *sLQTerm* são apresentados na Figura 45 (a) e (b), respectivamente. Os valores para cada propriedade foram selecionados de modo consistente com o serviço ao qual está associado. Na cláusula de negociação, Figura 44, o serviço *sHQTerm* (linha 15) é inicialmente negociado. Caso não seja possível manter este serviço (considerado que neste exemplo será possível implantá-lo inicialmente), o CM negociará a implantação do serviço *sLQTerm* (linha 16), de menor qualidade. Se nenhum serviço puder ser implantado a aplicação é terminada.

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

Figura 46 - Perfil de seleção hqRef

Em tempo de implantação do contrato, o CM deve selecionar um refletor específico. Assim, na descrição da ligação da referência *term* (instanciada na linha 2), no serviço preferencial (Figura 44, linha 4) é usada a referência virtual *ref*, que vai conter a referência ao refletor a ser efetivamente usado na ligação. Essa referência é resolvida, quando da implantação do serviço inicial, pela descoberta dinâmica de recurso feita pelo *Selector* usando o Serviço de Descoberta. Essa descoberta leva em consideração a classe de recurso a ser descoberta (*Reflector*) o domínio do recurso (*teleconf*) e as restrições de contexto a serem satisfeitas (*hqTermCProf*). Ao obter a lista de recursos, o *Selector* usa o perfil de seleção *hqRef*, mostrado na Figura 46, aos perfis de cada refletor da lista obtida do Serviço de Descoberta, sendo os resultados classificados de modo a determinar qual o refletor mais adequado. A função *hqRef* informa que as propriedades *MemFree* e *CPUIdle* devem ser maximizadas, sendo que esta última tem maior importância, visto que o seu peso é duas vezes maior que o de *MemFree*. A referência da instância específica escolhida é então retornada para o

Contract Manager e utilizada nas operações de configuração. A chamada enviada pelo *Selector* ao Serviço de Descoberta é mostrada na Figura 47.

```
01<ResourceQuery type=”Reflector”>
02  <MaxResults>5</MaxResults>
03    <Constraints From=”Transport”>
05       <Attribute Name=”delay” op=”<=” Val=”80” />
06       <Attribute Name=”bandwidth” op=”>=” Val=”1600” />
07    </Constraints>
08 </ResourceQuery>
```

Figura 47 - Consulta de Descoberta dos Refletores do serviço sHQ_Term

5.2.6 Implementação

Utilizando o *framework* CR-RIO, a aplicação de videoconferência foi implementada com o objetivo de avaliar a abordagem de contratos e realizar experimentos relacionados ao chaveamento entre serviços. Especializações do *Contractor* foram implementadas, de forma tratar as restrições específicas aos serviços dos contratos relacionados a essa aplicação. Da mesma forma, Agentes de Recurso foram especializados para oferecer acesso às métricas associadas a cada contrato. Especificamente, foram desenvolvidos três Agentes de Recurso:

- Processamento: realiza a monitoração de recursos locais como CPU e memória; implementado utilizando a ferramenta Ganglia [Massie et al 2004].
- Rede: provê medidas de banda passante e atraso para os enlaces de comunicação; baseado na ferramenta Abing [Navratil & Cottrell 2003].
- *Real Time Protocol*: encarregado da monitoração de propriedades dos fluxos de mídia. Desenvolvido a partir da biblioteca *commons* distribuída com o programa VIC [Mbone 2006]. Na versão atual, monitora a perda de pacotes RTP.

Para os três tipos de agentes foram utilizadas algumas ferramentas (ex. Ganglia, Abing) específicas para coletar os valores das propriedades de cada tipo de recurso. Entretanto, os agentes são todos desenvolvidos baseados em um modelo de objetos que adota um padrão de projeto que permite trocar a instrumentação de baixo nível sem necessitar mudanças no código da aplicação. Detalhes do modelo proposto para implementar Agentes de Recurso podem ser encontrados no Apêndice A.

O refletor utiliza como base o código descrito em [Hodson et al 1998], que faz apenas o repasse de pacotes UDP. A versão original foi ampliada com uma API que permite fazer o gerenciamento dos participantes e a criação dinâmica de sessões. Para o experimento, configuramos cenários com fluxos de áudio e vídeo pré-gravados. Foram utilizadas máquinas (Pentium 4 de 2.80 GHz/ 512 MB de RAM) distribuídas entre a UFF e a UERJ, conectadas através da RedeRio, "estrangulada" com um HUB de 10 Mbps (para viabilizar a injeção de fluxos de *background*, sem impactar os nós sob observação), com uma conexão redundante através da rede GIGA, utilizando *switches* de 100/1000 Mbps.

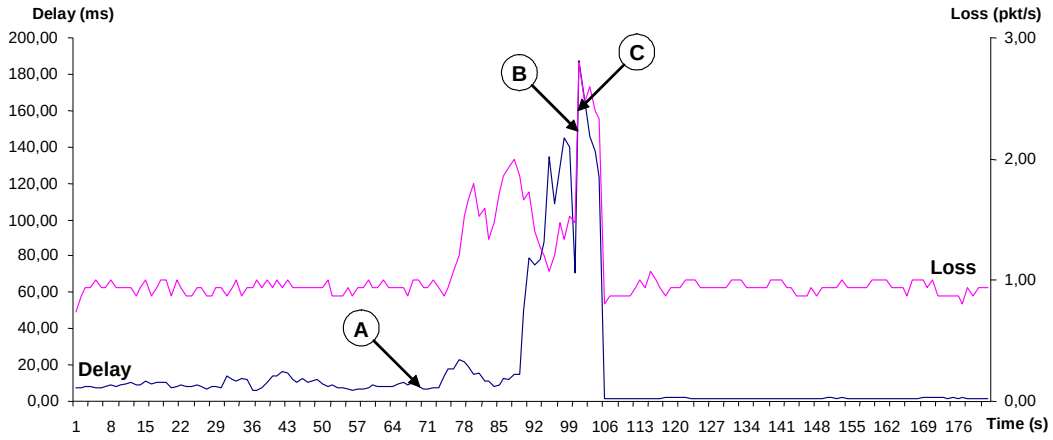


Figura 48 - Médias (média móvel) de atraso e perdas de pacote RTP

A Figura 48 apresenta o resultado de um experimento associado ao contrato de comunicação entre dois refletores, *refUFF* e *refUERJ*; serviço *primLink*, descrito na Figura 41. Um fluxo de áudio foi transmitido de *refUFF* para *refUERJ* e um fluxo UDP injetado como *background*. A medida de atraso (*delay*) foi utilizada para identificar a necessidade de adaptação. Também foi monitorada a perda de pacotes RTP em *refUERJ*, para avaliar a qualidade do fluxo de áudio sob teste. O ponto A mostra o início da introdução do fluxo UDP. O ponto B indica o instante em que o Agente de Recurso no *refUERJ* monitora um valor para o atraso (*delay*) acima do estabelecido no perfil (150 ms), fato reportado ao CM através do *Contractor*. O ponto C aponta o instante em que o serviço alternativo (*altLink*) é implantado (entre B e C, a máquina de estados é avaliada, e um novo serviço é selecionado e implantado). O Agente de Recurso continua a reportar perdas de pacotes, que são descartadas pelo *Contractor*, até que o serviço alternativo tenha tempo suficiente para entrar em regime

permanente; verifica-se que a partir deste ponto o valor do atraso retorna ao patamar desejado.

De modo a evitar instabilidades, provocadas por medidas fora do padrão temporário do fluxo monitorado, que poderiam causar falsas transições entre serviços, foi usado um filtro de média móvel. No experimento da Figura 48, foi adotado um período de amostragem de 3 segundos, sendo a média móvel calculada sobre os 5 últimos valores. Esses valores de amostragem e janela da média móvel nos permitiram identificar tendências nas medidas e foram obtidos empiricamente através de experimentos práticos. Por outro lado, segundo testes executados em [Corradi 2005], o tempo médio gasto entre o recebimento de uma notificação de violação de perfil e o estabelecimento de um próximo serviço é cerca de 2,3 ms. Adicionalmente, o tempo gasto para trocar/adicionar um componente (o que pode ser necessário para efetivar uma adaptação, e.g., trocar um *codec*), utilizando um configurador integrado ao *middleware* [Santos 2006], varia entre 10,4 ms (melhor caso: execução local/objeto pré-instanciado) e 28,7 ms (pior caso: execução remota/com criação de objeto). Essas últimas medidas demonstram que o gargalo, que limita o tempo total de adaptação, é explicitamente ligado ao intervalo necessário para identificar de forma consistente a violação dos perfis, associados ao serviço em atividade. Desta forma, fica constatada a necessidade de desenvolver técnicas mais sofisticadas, que permitam disparar, rápida e consistentemente, o chaveamento entre serviços; [Bhatti et al 1999] apresenta uma investigação mais detalhada deste tópico.

5.3 Aplicação de Vídeo sob Demanda Pervasiva

A aplicação de vídeo sob demanda pervasiva (VodP) apresenta um cenário onde o usuário transita por vários ambientes, sendo que cada ambiente apresenta sua própria composição em termos de recursos disponíveis e presença de outros usuários. Especificamente para esta aplicação, têm-se os ambientes representados por salas em um prédio. Para tornar possível sua localização, cada usuário possui um sensor que informa para um serviço de localização de usuários em que sala ele se encontra.

Este é um exemplo clássico de “aplicação móvel sensível ao contexto” (*context-aware mobile application*) [Dey 2000] que explora o contexto dinâmico dos seus usuários, provocado pela mobilidade e constante mudança no ambiente e que

fazem uso de novos dispositivos portáteis multifuncionais, como os PDAs e telefones celulares.

5.3.1 Cenário Base

Nesta aplicação um fluxo de vídeo deve ser reproduzido em um dos dispositivos disponíveis na sala em que o usuário se encontra, onde ele pode estar sozinho ou acompanhado. O conjunto de dispositivos de exibição de vídeo em cada ambiente pode ser usado por outros usuários para outras finalidades. Neste exemplo, para fins de orientação da adaptação da aplicação (reprodução do vídeo), são estabelecidas ordens de preferência do usuário por certos dispositivos, que varia de acordo com o contexto do ambiente que ele se encontra. Por exemplo, sempre que o usuário estiver em uma sala e outras pessoas estiverem presentes, o vídeo deve ser reproduzido no *display* pessoal do usuário. Para esta aplicação, são considerados os seguintes dispositivos: Monitor de Computador, PDA, Televisão de Plasma, Projetor e outros possíveis *displays* que não contam desta lista.

Algumas tarefas que devem ser feitas no gerenciamento e execução da aplicação incluem: (i) saber a localização atual de um usuário, (ii) identificar se o usuário esta ou não em sua própria sala, (iii) listar os dispositivos disponíveis em determinada sala. Além dessas tarefas, também se deve utilizar um perfil de seleção para definir o “melhor” dispositivo usado para visualizar o vídeo. Esta definição depende do contexto do usuário no momento e leva em conta se ele está ou não em sua sala e também se ele está acompanhado de outras pessoas.

Nas subseções a seguir é mostrado como essas restrições podem ser descritas por meio de contratos de qualidade de serviço usados no *framework* CR-RIO.

5.3.2 Definição das Categorias

Para esta aplicação foram identificados três tipos de recursos: (i) Usuário (*User*), (ii) *Display*, e (iii) Sala (*Room*). A Figura 49 mostra as categorias utilizadas na representação de cada um destes recursos. A categoria *User* (Figura 49, linhas 1 - 6) representa um usuário, onde a propriedade *id* especifica sua identificação, *userRoom*

informa a sala pertencente ao usuário, *currentRoom* mostra a localização corrente, e *envChange* indica se o usuário esta mudando de sala.

```
1 QoSCategory User {
2   id: numeric in;
3   userRoom: numeric in;
4   currentRoom: numeric in;
5   envChange: boolean in;
6 }
7
8 QoSCategory Room {
9   nUsers: numeric in;
10   number: numeric in;
11 }
12
13 QosCategory Display {
14   location: numeric in;
15   type: enum (TvPlasma, Projetor, PCMonitor, PDA, Outro) out;
16   mobile: boolean in;
17 }
```

Figura 49 - Categorias da Aplicação Vod Pervasiva

A categoria *Room* (Figura 49, linhas 8 – 11) representa o recurso sala. As propriedades *nUsers*, e *number* são utilizadas, respectivamente, para informar o número de usuários presentes e o número da sala, que no contrato será usado como sua identificação. Também foi criada a categoria *Display* que contém as propriedades *location*, *type* e *mobile*, usadas respectivamente para informar a localização do *display*, seus possíveis tipos e se ele é um dispositivo móvel.

A Figura 50 mostra como essas categorias devem ser representadas de acordo com o *schema* XML do Serviço de Descoberta e armazenadas no Diretório de Recursos.

```
1<Resource>
2     <Type> User </Type>
3     <Description> User Location Service </Description>
4     <Attributes>
5           <Attribute Name="id"          Type="integer" />        (a)
7           <Attribute Name="userRoom"    Type="integer" />
8           <Attribute Name="currentRoom" Type="integer" />
9           <Attribute Name="envChange"   Type="boolean" />
```

```
10    </Attributes>
11</Resource>

1<Resource>
2     <Type> Room </Type>
3     <Description> Room Information Service </Description>
4     <Attributes>
5           <Attribute Name=”nUsers”  Type=”integer” />          (b)
6           <Attribute Name=”number”  Type=”integer” />
7     </Attributes>
8</Resource>

1<Resource>
2     <Type> Display </Type>
3     <Description> Display Information Service </Description>
4     <Attributes>
5           <Attribute Name=”location” Type=”integer” />         (c)
7           <Attribute Name=”type”      Type=”string” />
8           <Attribute Name=”mobile”    Type=”boolean” />
9     </Attributes>
10</Resource>
```

Figura 50 - Representação XML dos recursos. (a) Usuário, (b) Sala e (c) *Display*

5.3.3 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 está 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. Pode-se fazer isso através do suporte a contratos do CR-RIO utilizando serviços e perfis. A Figura 51 mostra uma possível solução de contrato para a aplicação VodP e a Figura 52 mostra os perfis utilizados neste contrato.

```
1 contract (userId, userRoom){
2  service {
3     link serverVoD to disp = Display select(uDisplayPropriaSala) with
4                                                PropriaSala, ExisteDisplay;
5  } sPropriaSala;
6
7  service {
```

```
8    link serverVoD to disp = Display select (uDisplaySozinho) with
9                                OutraSala, Sozinho, ExisteDisplay;
10  } sOutraSalaSozinho;
11
12  service {
13  link serverVoD to disp = Display select (uDisplayAcompanhado) with
14                            OutraSala, Acompanhado, ExisteDisplay;
15  } sOutraSalaAcompanhado;
16
17  service {
18     block severVod;
19  } sMudancaContexto;
20
21  negotiation {
22     not sPropriaSala -> (sOutraSalaSozinho || sOutraSalaAcompanhado ||
23     sMudancaContexto);
24
25     sOutraSalaSozinho -> (sPropriaSala || sOutraSalaAcompanhado ||
26     MudancaContexto);
27
28     sOutraSalaAcompanhado  -> (sPropriaSala || sOutraSalaSozinho ||
29     sMudancaContexto);
30
31     sMudancaContexto -> (sPropriaSala || sOutraSalaSozinho ||
32     sOutraSalaAcompanhado);
33  }
34 } VodP;
```

Figura 51 - Contrato Vod pervasivo

A linha 1 do contrato *VodP* indica que ele recebe parâmetros no momento de sua instanciação. Estes parâmetros são chamados de variáveis de contexto e são utilizados pelos perfis de cada serviço. No exemplo em questão, tem-se duas variáveis *userId* e *userRoom* usadas respectivamente para indicar a identificação e número da sala do usuário. Essas variáveis são obrigatórias e devem ser repassadas ao contrato para que seja possível instanciá-lo.

No contrato da Figura 51 são definidos quatro serviços. No primeiro serviço *sPropriaSala* (linhas 2-5), o usuário esta em sua própria sala, situação definida pelo perfil *PropriaSala* (Figura 52, linhas 1-5). Neste perfil são utilizadas duas variáveis de contexto (linhas 2-3) que indicam a identificação e número da sala do usuário que executa a instância do contrato. O caractere ‘%’ é usado para acessar os valores das variáveis de contexto do contrato. O perfil *ExisteDisplay* (Figura 52, linhas 47 - 51)

define que devem ser localizados todos os possíveis *Displays* pertencentes a sala onde o usuário se encontra. O perfil de seleção *uDisplayPropriaSala* (linha 28 da Figura 52) especifica a ordem de preferência dos *Displays*. Este perfil de seleção é usado para classificar a lista de *Displays* obtida do Serviço de Descoberta e indica que o monitor do computador é o *Display* preferencial.

No segundo serviço *sOutraSalaSozinho* (Figura 51, linhas 7-10), o usuário muda de sala e se encontra sozinho, situação em que o vídeo deve ser mostrado em algum *Display* disponível e de preferência o de maior qualidade, de acordo com a preferência do perfil de seleção *uDisplaySozinho* (Figura 52, linhas 26-28). Neste perfil tem-se a televisão de plasma como *Display* preferencial e depois um projetor. O *framework* deve tentar utilizar um tipo específico de *Display* de acordo com a ordem descrita no perfil. Os perfis *OutraSala* (linhas 15-20) e *Sozinho* (linhas 10 - 14) definem, respectivamente quando o usuário não esta em sua sala e se esta acompanhado.

No terceiro serviço *sOutraSalaAcompanhado* (Figura 51, linhas 12 - 15), o usuário está em outra sala, porém na companhia de outras pessoas. O perfil *Acompanhado* (Figura 52, linhas 21-25) define se o usuário esta acompanhado de outras pessoas. Neste caso, por questões de privacidade ou para não ocupar o *display* compartilhado, o vídeo deve ser transmitido para o PDA do usuário de acordo com o perfil de seleção *uDisplayAcompanhado*. (Figura 52, linhas 37-42)

Por fim, o quarto serviço *sMudancaContexto* (Figura 51, linhas 17 - 19), é usado quando o usuário não é localizado ou quando não é encontrado um *Display* apropriado. Nestes casos, a transmissão do vídeo deve ser interrompida ate que outro serviço do contrato possa ser estabelecido.

```
1 profile {
2   User.id = %userId;
3   User.currentRoom = %userRoom;
4   User.envChange = false;
5 } PropriaSala;

10 profile {
11   User.id = %userId;
12   Room.number = User.currentRoom;
13   Room.nUsers = 1;
```

```
15 profile {
16   User.id = %userId;
17   User.currentRoom != %userRoom;
19   User.envChange = false;
20 } OutraSala;

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

```
14 } Sozinho;

26 Selection {
28  Display.type = (PCMonitor > palm);
29 } uDisplayPropriaSala;

30 Selection {
34    Display.type = (Plasma >
35          Projetor > monitor > palm);
36 } uDisplaySozinho;

43 profile {
44    User.id = %userId;
45    User.envChange = true;
46 } MudancaContexto;

25 } Acompanhado;

37 Selection {
41     Display.type = (palm);
42 } uDisplayAcompanhado;

47 profile {
48   User.id = %userId;
49   Display.location =
50              User.currentRoom;
51 } ExisteDisplay;
```

Figura 52 - Perfis da aplicação Vod pervasivo .

5.3.4 Gerenciamento do Contrato

Para que o contrato possa ser gerenciado pelo CR-RIO, é preciso implementar (caso eles não existam) os Agentes de Recurso específicos que obtém as informações do usuário e dos *Displays*, e registrá-los no Serviço de Contexto. Também e necessário codificar o *Contractor* que gerenciará a monitoração feita por meio do Serviço de Contexto, verificando se as propriedades monitoradas validam as restrições descritas nos perfis contrato. O Serviço de Descoberta provê a infra-estrutura básica para localizar um dentre os vários *Displays* de forma a obter o uma instância que atenda as restrições dos perfis.

No gerenciamento desse contrato (*VodP*), o primeiro passo é a interpretação de sua descrição pelo *Contract Manager* (CM). Obtida as informações do contrato, o CM identifica na cláusula de negociação de serviços (Figura 51, linhas 21-33) que *sPropriaSala* é o serviço preferencial. Neste serviço existe apenas uma chamada *link* para ligar o servidor de vídeo (instanciado estaticamente e referenciado por *serverVod)* ao Display. Os *Displays* já foram instanciados e suas informações de registro se encontram do Diretório de Recursos. O Serviço de Descoberta é usado para obter a lista de *Displays* disponíveis no ambiente corrente do usuário. Assim, o CR-RIO tenta localizar um *Display* que satisfaça às restrições descritas nos perfis *PropriaSala* e *ExisteDisplay* que usam variáveis de contexto para obter a

indentificação do usuário (*userId*) e a sua sala (*userRoom*). 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 *uDisplayPropriaSala* (que estabelece um ordem de preferência) e repassa para o CM que conclui que o usuário está em sua própria sala. Após identificar os recursos que farão parte da instância da aplicação, o CM deverá monitorar as restrições conjuntas para esse serviço (perfis *PropriaSala* e *ExisteDisplay*). Para isso o CM envia essas restrições para o *Contractor* responsável pela verificação das propriedades contidas nos perfis. Para essa aplicação, há um *Contractor* global, responsável pela verificação das propriedades da localização do usuário e da disponibilidade dos *Displays*. Esses *Contractor* recebe, então, essas restrições e solicita ao Serviço de Contexto, os valores atuais das propriedades dos recursos, verificando se estes satisfazem as restrições para o serviço *sPropriaSala*. Caso as restrições sejam satisfeitas, o CM é notificado que esse serviço pode ser estabelecido e para isso o CM envia a configuração arquitetural desse serviço (linha 3 da Figura 51) ao *Configurator*, que se encarrega de estabelecer a configuração exigida para o provimento do serviço.

Depois que o serviço estiver estabelecido, as propriedades dos recursos continuam sendo monitoradas periodicamente pelo *Contractor* e, caso o valor de uma das propriedades viole as restrições para o serviço corrente, ele notifica o CM, que tenta o estabelecimento do próximo serviço (*sOutraSalaSozinho*). Por exemplo, se o serviço *sPropriaSala* está sendo provido e, em um determinado momento, o usuário muda de sala, violando assim a restrição para esse serviço (linha 3 da Figura 52), o serviço corrente não poderá ser mantido. Esta situação faz com que o CR-RIO tente estabelecer outro serviço, de acordo com as cláusulas de negociação. De acordo com o contrato, ele tentará estabelecer o serviço *sOutraSalaSozinho* e, para isso, ele executa todos os passos descritos anteriormente para o serviço *sPropriaSala*.

5.4 Conclusão do Capítulo

Este capítulo apresentou exemplos de aplicações que possuem restrições de contexto em seus serviços e que precisam da monitoração e descoberta dinâmica de recursos. Esses exemplos permitiram demonstrar a funcionalidade do Serviço de Contexto e do Serviço de Descoberta, além do *framework* CR-RIO no gerenciamento

de aplicações sensíveis ao contexto. Assim, foi possível validar os serviços propostos e a integração com o CR-RIO.

O primeiro exemplo mostrado foi de uma aplicação de vídeo sob demanda dinâmica constituída de réplicas de servidores multimídia que transmitem vídeos a clientes. Foi exemplificado o uso do Serviço de Contexto e do Serviço de Descoberta, além da utilização de contratos na descrição de serviços que definem a qualidade do vídeo transmitido. Estes serviços foram associados a restrições, dos recursos de processamento do cliente e da disponibilidade de banda passante no canal de comunicação entre o cliente e os servidores, que especificam o que é necessário, em termos de recursos, para a transmissão do vídeo em uma determinada qualidade. Foi mostrado como o CR-RIO monitora os níveis desses recursos por meio do Serviço de Contexto, permitindo identificar a adequabilidade de estabelecimento de um serviço ou de mantê-lo em execução. Também foi apresentado como o Serviço de Descoberta pode ser usado para localizar servidores que satisfaçam as restrições descritas no contrato da aplicação. Através do gerenciamento da aplicação pelo CR-RIO, um vídeo sendo transmitido a um cliente pode ter a sua qualidade degradada de forma automática, contando com um mecanismo de transcodificação de formato, para se adequar à escassez de recursos, em um determinado momento da transmissão.

O segundo exemplo descreveu o cenário de uma aplicação de videoconferência composta por usuários com diferentes características, o que exige configurações e qualidade distintas para os serviços por eles definidos através dos contratos de qualidade de serviço. Trata-se de um cenário mais complexo que o primeiro, onde foi possível exemplificar o uso do *framework* CR-RIO para gerenciar, de forma distribuída, os segmentos desta aplicação. Foi mostrado como os usuários dessa aplicação poderiam ter os seus contratos gerenciados pelos componentes do CR-RIO de forma distribuída e independente. A utilização do Serviço de Descoberta possibilitou realizar a descoberta dinâmica do refletor com o qual o cliente se comunica ao entrar em uma sessão de videoconferência.

O terceiro exemplo demonstrou uma aplicação de vídeo sob-demanda em ambiente pervasivo. Foi exemplificado como descrever diferentes serviços da aplicação, através de um contrato, que define as políticas de adaptação de acordo com o estado do contexto em que a aplicação se encontra. Este estado de contexto inclui a localização corrente do usuário e também o número de pessoas presentes na sala.

Além disso, foi mostrado o uso de perfil de seleção para filtrar os recursos de acordo com a preferência descrita no contrato. Neste exemplo, foi usado um filtro para se estabelecer a ordem de preferência por determinado *Display* de acordo com o contexto do usuário da aplicação. Através deste exemplo, foi possível demonstrar a capacidade do *framework* CR-RIO na especificação e gerência de aplicações pervasivas sensíveis ao contexto.

A partir das aplicações mostradas neste capítulo, demonstrou-se a importância do Serviço de Contexto e do Serviço de Descoberta. Sem estes serviços o desenvolvimento destas aplicações só seria possível com implementações *adhoc* destes serviços ou através de um trabalho específico de integração de ferramentas como as apresentadas no Capítulo 2. O emprego dos serviços propostos facilita o desenvolvimento destas aplicações mantendo-se as características de separação de interesses e reusabilidade.

6 Conclusão

Aplicações de computação ubíqua e pervasiva, sistemas autônomos, e aplicações com requisitos não-funcionais, de uma forma geral, são sensíveis ao contexto de operação. Por isso, tais aplicações demandam um suporte específico para a especificação, projeto e suporte relacionado a estes requisitos não-funcionais. Nesta linha, nosso trabalho contempla uma proposta para disponibilizar, facilitar a integração e o uso de elementos de suporte importantes: (i) instrumentação para coletar informações de contexto e do ambiente de operação, e (ii) uma infra-estrutura para a descoberta dinâmica de recursos. Esses dois elementos devem funcionar em conjunto, provendo o suporte básico a ser utilizado na construção das aplicações mencionadas para permitir as adaptações necessárias em face de mudanças no contexto de operação e na disponibilidade de recursos.

Inicialmente foram apresentados os conceitos básicos usados nesta dissertação. Isto incluiu uma revisão sobre Agentes de Recurso, onde foram identificados seus principais requisitos e funcionalidades, possibilitando estabelecer um conjunto de atividades básicas que os agentes devem realizar. Uma abordagem similar foi utilizada para o Serviço de Descoberta, onde foram apresentados seus pontos chaves e identificado os principais componentes que compõem este serviço. Foram mostradas possíveis formas para a representação de recurso e discutido como ontologias podem ser usadas para este fim. Em seguida, realizamos um estudo sobre as principais questões envolvidas no gerenciamento de aplicações sensíveis ao contexto, o que permitiu identificar os requisitos necessários para esse gerenciamento e descrita uma infra-estrutura geral de suporte formada por elementos – Especificação, Monitoramento, Gerência e Configuração – responsáveis pelas ações requeridas nesse processo.

Com base nas atividades de investigação mencionadas, apresentamos, então, nossa proposta. Primeiramente, discutimos uma forma padronizada para a descrição/representação e registro de recursos usando a linguagem XML. A partir da descrição dos recursos, foi apresentada a proposta de Serviço de Contexto, que provê uma camada de abstração sobre os Agentes de Recurso, possibilitando a monitoração das informações de contexto a partir de uma interface padronizada de alto nível e

oferecendo um serviço padronizado para consultas a estas informações. A interface entre o Serviço de Contexto e os Agentes de Recurso possibilita a mudança da instrumentação de baixo nível usada na coleta das informações de contexto sem implicar na modificação das aplicações. Além disso, a arquitetura proposta para o Serviço de Contexto absorve a sobrecarga de processamento e de comunicação, gerada na coleta das informações de contexto. Isso possibilita o desenvolvimento de aplicações pervasivas que executam em dispositivos com capacidade limitada de processamento e de bateria.

A partir da proposta de representação de recursos e a infra-estrutura disponibilizada pelo Serviço de Contexto, foi proposto o Serviço de Descoberta. Este serviço que suporta consultas contendo a especificação de restrições não-funcionais relacionadas ao contexto de operação dos recursos a serem descobertos. Desta forma, uma consulta ao Serviço de Descoberta pode retornar uma lista de recursos que atendem as restrições informadas na consulta. Adicionalmente, a interface padronizada para o registro/desregistro dos recursos e para as consultas e respostas do Serviço de Descoberta, possibilitou a obtenção de uma arquitetura genérica independente de tecnologia.

Na seqüência, apresentamos um trabalho de integração dos Serviços de Contexto e de Descoberta ao *framework* CR-RIO, discutindo como os serviços propostos interagem com os elementos da infra-estrutura de suporte deste *framework*. A infra-estrutura do CR-RIO foi adaptada para utilizar os novos serviços. Mais especificamente, o *Contract Manager* foi adaptado para consultar o Serviço de Descobertas e o *Contractor* adaptado para consultar diretamente o Serviço de Contexto. A descrição dos contratos foi estendida, visando prover suporte explícito à especificação de recursos virtuais, que devem ser descobertos dinamicamente em tempo de execução por meio do Serviço de Descoberta. Com isso, ligamos o nível arquitetural aos novos serviços disponíveis na infra-estrutura.

Um ponto importante da atividade de integração é a introdução de funções de seleção. Em princípio, uma consulta ao Serviço de Descoberta pode retornar uma lista de recursos, todos aptos a atender as restrições não-funcionais requeridas. Uma função de seleção aplicada a esta lista pode classificar os recursos obtidos de acordo com uma política definida por perfis de seleção. Assim, propusemos introduzir esta responsabilidade a um novo elemento denominado *Selector*, que aplicaria a função de

seleção, que recebe como parâmetro o perfil de seleção definido no contrato, após uma consulta ao Serviço de Descoberta. Para isso, indicamos uma forma para a especificação de perfil de seleção, utilizando elementos já existentes no contrato e um mecanismo para informar o uso dos perfis de seleção ainda no nível do contrato, ligando também esta especificação aos elementos da infra-estrutura.

Com o objetivo de validar a proposta, foram apresentadas três aplicações com restrições de contexto: (i) de vídeo sob demanda com descoberta dinâmica do servidor de vídeo, (ii) de videoconferência com descoberta dinâmica dos refletores de encaminhando de mídia, e (iii) de vídeo sob demanda em ambiente pervasivo. Através destes exemplos foi possível demonstrar a viabilidade da proposta e a flexibilidade do *framework* CR-RIO no atendimento das necessidades de aplicações "tradicionais" e pervasivas. Este trabalho expande o domínio de aplicação do CR-RIO, ao possibilitar a especificação e o gerenciamento de aplicações que se beneficiam da gerência de recursos com informação de contexto e com a descoberta dinâmica de recursos.

Outros *frameworks* também poderiam ser estendidos, por meio das facilidades proporcionadas pelos serviços propostos, com o propósito de ampliar os tipos de aplicações suportados. Apesar de esta dissertação ter usado o *framework* CR-RIO para validar a utilização dos serviços propostos, estes foram concebidos para serem genéricos, reusáveis, e independentes de tecnologia e podem ser integrados a outros *frameworks* de gerenciamento.

6.1 Trabalhos Futuros

As propostas apresentadas neste trabalho envolvem diversas técnicas e mecanismos, abrangendo assuntos relacionados a diversas áreas da computação, tais como a monitoração, representação e descoberta de recursos, *frameworks* de gerenciamento de contratos e sistemas pervasivos. O objetivo principal era apresentar uma forma consistente para tratar estes assuntos, tanto individualmente com de forma integrada, em alto-nível de abstração. Mais especificamente objetivamos integrar o Serviço de Contexto e o Serviço de Descoberta com uma linguagem de contratos e uma infra-estrutura para gerência de arquiteturas de software, o CR-RIO. As atividades de investigação realizadas durante o trabalho nos levaram a identificar

possibilidades de aprimoramento de nossa proposta e o prosseguimento em trabalhos futuros:

- Implementação dos Serviços de Contexto e de Descoberta para avaliar o desempenho da arquitetura dos serviços propostos, que apesar de diminuírem o *overhead* da comunicação em rede, impõem uma sobrecarga extra para fazer a interpretação das mensagens XML. Neste contexto também é importante propor um protocolo comum de comunicação entre o Serviço de Contexto e os Agentes de Recursos para facilitar a inserção de novos tipos de agentes no serviço.
- Avaliar sistemas banco de dados nativos-XML para o armazenamento das descrições dos recursos, o que facilita a consulta de suas informações. Um banco desse tipo é útil para armazenar informações sobre os recursos distribuídos disponíveis em uma rede, permitindo consultas para localizá-los no processo de descoberta de recursos. Também é importante avaliar técnicas para a federação de Serviços de Descoberta pertencentes a diferentes domínios administrativos.
- Estudar a viabilidade de adicionar suporte a especificação de diretivas de otimização na consulta de descoberta, possibilitando a seleção automática do recurso. Isso permite deixar a carga da classificação dos resultados por conta do Serviço de Descoberta. No entanto, isso requer o estudo para uma forma mais expressiva de representação de perfis de seleção sofisticados, além de aumentar a complexidade da implementação do Serviço de Descoberta.
- Efetuar testes com aplicações pervasivas em cenários mais complexos. Embora a aplicação exposta neste trabalho tenha permitido validar a utilização dos serviços propostos neste domínio de aplicação, seria interessante a avaliação sobre outros cenários. Um exemplo é o de ambiente inteligente (*smart space*) onde existe uma diversidade maior de dispositivos que interagem entre si de forma coordenada.
- A proposta apresentada para a representação de recursos contém elementos que poderiam ser utilizados em um sistema de ontologias. Assim, poderia ser investigado como introduzir a representação semântica à forma de

descrição e ao serviço de localização de recursos. Dentre as soluções existentes destacam-se o RDF, RDFS [W3C 2006] e as diversas variações da DAML, como a DAML-S [DAMLS 2006] ou DAML-OIL [DAMLO 2006].

De modo a tornar efetivo e facilitar o uso dos serviços propostos, espera-se que uma separação explicita seja adotada futuramente, no projeto dos artefatos comumente usados no suporte a requisitos não-funcionais e de contexto. Fabricantes de componentes de software e provedores recursos poderiam fornecer uma descrição das características de seus produtos de uma forma padronizada, o que facilitaria, por exemplo, o registro no Serviço de Descoberta. Assim, as descrições de recursos poderiam ser fornecidas como arquivos XML, o que facilitaria a capacidade de interoperabilidade entre diversos sistemas que necessitam das informações contidas nessas descrições, tais como o Serviço de Contexto e o Serviço de Descoberta. Além disso, Agentes de Recurso, com APIs padronizadas, também poderiam ser providos para facilitar gerência e monitoramento de seus mecanismo internos. Isto facilitará a integração com o nível alto de gerenciamento, permitindo mais flexibilidade para impor políticas adequadas para atender as exigências das aplicações.

Bibliografia

[Aagedal & Ecklund 2002] Aagedal, J. Ø.; Ecklund, E. F. **Modelling QoS: Towards a UML Profile**. <<UML>> 2002, Dresden, Alemanha, setembro, 2002.

[Adamczyk et al 2003] Adamczyk, D., Collados, D., Denis, G., Fernandes, J. et al., **VRVS 3 - Global Platform for Rich Media Conferencing and Collaboration**, *Third Annual Workshop on Advanced Collaborative Environments*, Seattle, Washington, junho, 2003.

[Ansaloni 2003] Ansaloni, S. **Proposta de Padrão Arquitetural para Descrição e Implementação de Contratos de QoS**. Dissertação (Mestrado) — Universidade Federal Fluminense, 2003.

[Arnold 1999] ARNOLD, Ken. **The Jini Architecture: Dynamic Services in a Flexible Network**. *Sun Microsystems*, Inc. *1 Network Drive. Communications of the* ACM, 1999.

[Aydt et al 2001] Aydt, R. Smith, W. Swany, M. Taylor, V. Tierney, B. Wolski, R. **A Grid Monitoring Architecture**. julho, 2001.

[Batista 2003] Batista, L. **Uma Proposta para Integração de Protocolos de Localização de Serviços em Computação Ubíqua e Pervasiva**. Dissertação de Mestrado, Instituto de Computação - Universidade Federal Fluminense (IC/UFF), 2003

[Becker & Geihs 1997] Becker, C.; Geihs, K. **MAQS - Management for Adaptive QoS-enabled Services.** IEEE *Workshop on Middleware for Distributed Real-Time Systems and Services*, São Francisco, EUA, 1997.

[Beugnard et al 1999] Beugnard, A., Jézéquel, J.-M., Plouzeau, N. e Watkins, D., **Making Components Contract Aware**, IEEE *Computer*, Vol. 32, No.7, p. 38-45, julho 1999.

[Bhatti et al 1999] Bhatti, S. N. e Knight, G., **Enabling QoS adaptation decisions for Internet applications, Computer Networks**, Vol. 31, No. 7, pp. 669-692, março, 1999.

[Broens 2004] Broens, T., **Context-aware, Ontology based, Semantic Service Discovery** *Thesis for a Master of Science degree in Telematics from the University of Twente, Enschede, The Netherlands*, julho, 2004.

[Capra et al 2005] Capra. L, Zachariadis, S and Mascolo. C, **Q-CAD: QoS and Context Aware Discovery Framework for Mobile Systems.** *In Proc. of International Conference on Pervasive Services* (ICPS'05), Santorini, Grécia, julho, 2005.

[Cardoso et al 2006] CARDOSO, Leonardo Teixeira; SZTAJNBERG, Alexandre ; LOQUES FILHO, Orlando Gomes . **Self-adaptive applications using ADL contracts**. In: *Second* IEEE *International Workshop on Self-Managed Networks, Systems & Services*, 2006, *Dublin. Lecture Notes in Computing Science*, Alexander Keller and Jean-Philippe Martin-Flatin (Eds.), 2006. Vol. 3996. p. 87-101.

[Cardoso et al 2006b] CARDOSO, Leonardo Xavier Teixeira; SZTAJNBERG, Alexandre; LOQUES FILHO, Orlando Gomes. **Provendo aplicações com requisitos não-funcionais dinâmicos através de contratos**. In: 24o Simpósio Brasileiro de Redes de Computadores, 2006, Curitiba. Anais do 24o Simpósio Brasileiro de Redes de Computadores, 2006. Vol. 1. p. 671-685.

[Carvalho 2001] Carvalho, S. **Um Design Pattern para a Configuração de Arquiteturas de Software**. Dissertação de Mestrado, Instituto de Computação – Universidade Federal Fluminense (IC/UFF), janeiro, 2001.

[Cheng et al. 2002] Cheng, S.; Garlan, D.; Schmerl, B.; Sousa, J. P.; Spitznagel, B.; Steenkiste, P. **Using Architectural Style as a Basis for Self-repair.** *Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture*, Montreal, Canadá, agosto, 2002.

[Cheshire & Krochmal 2005] Cheshire, S. & Krochmal, M. **DNS-Based Service Discovery**, IETF *Internet draft, work in progress*, junho 2005.

[Cohen et al 2000] Cohen, J. et al. **General Event Notification Architecture Base: Client to Arbiter (GENA)**. INTERNET DRAFT. setembro de 2000. 14p. Disponível em http://www.upnp.org/download/draft-cohen-gena-client-01.txt

[Corradi 2005] Corradi, A. **Um *Framework* de Suporte a Requisitos Não-Funcionais para Serviços de Nível Alto**. Dissertação de Mestrado, Instituto de Computação - Universidade Federal Fluminense (IC/UFF), agosto, 2005.

[Curty 2002] Curty, R. **Uma Proposta para Descrição e Implementação de Contratos para Serviços com Qualidade Diferenciada**. Dissertação de Mestrado, Instituto de Computação - Universidade Federal Fluminense (IC/UFF), novembro, 2002.

[DAMLS 2006] OWL-based Web Service Ontology. Disponível em http://www.daml.org/services/owl-s/, Visitado: outubro 2006

[DAMLO 2006] DAML+OIL. Disponível em http://www.daml.org/2001/03/daml+oil-index.html, Visitado: outubro 2006

[Davies et al 2003] Davies , J., Fensel, D., Harmelen, F. van., **Towards The Semantic Web: Ontology-Driven Knowledge Management**, John Wiley & Sons Ltd, 2003.

[Dey 2000] A. Dey, **Providing Architectural Support for Context-Aware applications**, *Georgia Institute of Technology*, PHD *Thesis*, novembro 2000

[Dinda et al 2001] Dinda, P. A.; Gross, T.; Karrer, R.; Lowekamp, B.; Miller, N.; Steenkiste, P.; Sutherland, D. **The Architecture of the Remos System**. 10th *IEEE Symposium on High Performance Distributed Computing*, São Francisco, Califórnia, EUA, agosto, 2001.

[Eugene & Zhang 2002] Eugene, T. S.; Zhang, H. **Predicting Internet Network Distance with Coordinates-Based Approaches.** *Proceedings of the IEEE INFOCOM 2002*, New York, New York, EUA, junho, 2002.

[Exist 2006] **Exist - Open Source XML Database**. Disponível na Internet. http://exist.sourceforge.net/ em 28 de setembro de 2006.

[Fen et al 2005] Fen Zhu, Matt W. Mutkaand, Lionel M. Ni, **Service Discovery in Pervasive Computing Environments**, IEEE *Pervasive Computing*, Vol 4, issn 1536-1268, 2005, p. 81-90.

[Frolund & Koistinen 1998] Frolund, S.; Koistinen, J**. Quality of Service Specification in Distributed Object Systems Design**. *Proceedings of the 4th USENIX Conference on Object-Oriented Technologies and Systems* (COOTS), Santa Fé, Novo México, abril, 1998.

[Freitas 2005] Freitas, G. **Um Experimento de Uso de um F*ramework* de Suporte a Requisitos Não-Funcionais de Qualidade**. Dissertação de Mestrado, Instituto de Computação - Universidade Federal Fluminense (IC/UFF), novembro, 2005.

[Freitas et al 2005] Freitas, G., Cardoso, L., Santos, A., Loques, O. **Otimizando a Utilização de Servidores através de Contratos Arquiteturais**. . In XXIII SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, 2005, Fortaleza. XXIII SBRC, Workshop de Sistemas de Tempo Real, SBRC WTR 2005.

[Gaia 2006] **Gaia – Active Spaces for Ubiquitous Computing**. Disponível em http://gaia.cs.uiuc.edu/, Visitado: outubro 2006.

[Gamma et al 1995] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. **Design Patterns – Elements of Reusable Object-Oriented Software**, *Addison Wesley Publishing*, EUA, 1995.

[Ganek & Corbi 2003] Ganek, A. G.& Corbi, T. A. **The Dawning of the Autonomic Computing Era**. IBM Syst. J., IBM Corp., Riverton, NJ, USA, Vol. 42, No. 1, p. 5–18, 2003.

[Garlan et al 2004] Garlan, D.; Cheng, S.-W.; Huang, A.-C.; Schmerl, B. e Steenkiste, P. **Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure**. *Computer*, IEEE *Computer* Society Press, Los Alamitos, CA, USA, Vol 37, No. 10, p. 46–54, 2004.

[Giga 2006] Projeto GIGA. Disponível em www.giga.org.br, Visitado: fevereiro 2006.

[Guarino 1998] Guarino, N. **Formal Ontology and Information Systems**. In: *Proceedings of the First Int. Conference on Formal Ontology in Information Systems*, Trento, Italy, junho 1998.

[Guttman et al 1999] Guttman, E. Perkins, C. Veizades, J. Day, M. **Service Location Protocol**, Version 2. RFC2608, junho 1999.

[Guttman et al. 1999a] Guttman, E. et al. **Service Templates and Service: Schemes**. RFC 2609. Junho de 1999. Disponível em http://www.ietf.org/rfc/rfc2609.txt

[Hodes et al 2002] Hodes, T. et al., **An Architecture for Secure Wide-Area Service Discovery**, ACM *Wireless Networks* J.,Vol.8, Nos 2/3, 2002, pp. 21-30.

[Hodson et al 1998] Hodson, O., Varakliotis, O., S. & Hardman, V., **A software platform for multi-way audio distribution over the Internet, Audio and music technology: the challenge of creative DSP**, IEE Colloquium, London, novembro 1998.

[Huang & Steenkiste 2003] Huang,A.-C. & Steenkiste, P. **Network Sensitive Service Discovery**. In: USENIX *Symposium on Internet Technologies and Systems*, 2003.

[Jain et al 2002] Jain, M., Dovrolis, C. Pathload, **A measurement tool for end-to-end available bandwidth**, *Proceedings of Passive and Active Measurements* (PAM) *Workshop*, março, 2002.

[Judd & Steenkiste 2003] Judd, G. & Steenkiste, P. **Providing contextual information to pervasive computing applications**. In 1st IEEE *Conference on Pervasive Computing and Communications* (PerCom), p. 133--142, Fort Worth, março 2003.

[KindBerg & Fox 2002] Kindberg ,T. & Fox, A.**. System software for ubiquitous computing**. *Pervasive Computing Magazine*, 2002.

[Kyungmin et al 2006] Kyungmin Lee, Dongman Lee, Yang Woo Ko, Jaeik Lee, Yoo Chul Chung, **An Objectified Naming System for Provinding Context Transparent to Context-Aware applications** *Proceedings of the Fourth Workshop on Software Technologies for Future Embedded and Ubiquitous Systems*, 2006.

[Lisbôa 2003] Lisbôa, J. **Utilização do design pattern architecture configurator em um ambiente de suporte à configuração de arquiteturas**. Dissertação de Mestrado, Instituto de Computação – Universidade Federal Fluminense (IC/UFF), abril, 2003.

[Loques & Sztajnberg 2004] Loques, O. & Sztajnberg, **A. Customizing Component-Based Architectures by Contract.** *Proceedings of Second International Working Conference on Component Deployment*, p. 18-34, Edimburgo, Reino Unido, Maio, 2004.

[Loyall et al 1998] Loyall, J. P.; Schantz, R. E.; Zinky, J. A.; Bakken, D. E. **Specifying and Measuring Quality of Service in Distributed Object Systems**. *Proceedings of The 1st IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, Kyoto, Japão, p.43, abril, 1998.

[Massie et al 2004] Massie, M. L.;Chun, B.N.e Culler,D.E. **The Ganglia Distributed Monitoring System: Design, Implementation, and Experience.** *Parallel Computing*, v.30, n. 7, 2004.

[MBone 2006] UCL Network and Multimedia Research Group, **Mbone Conferencing Applications**, Disponível em http://www-mice.cs.ucl.ac.uk/multimedia/software. Visitado: outubro 2006

[Mendoza 2003] D. Rios Mendoza, **Using ontologies in Context-Aware Services Platforms**, *Masters Thesis*, University of Twente, novembro 2003.

[Microsoft 2000] MICROSOFT. **Understanding Universal Plug and Play White Paper**. Microsoft Corporation. junho de 2000. Disponível em http://www.upnp.org/download/UPNP_UnderstandingUPNP.doc

[Moore et al 2003] Andrew Moore, James Hall, Christian Kreibich, Euan Harris and Ian Pratt, **Architecture of a Network Monitor**, in **Proceedings of the Passive and Active Measurement Workshop**, La Jolla, California, abril 6-8, 2003.

[Motta 2005] Motta, Paulo. **Sinapse: Um Arcabouço para suporte de Aplicações Publish/Subscribe baseado em configuração arquitetural**. Dissertação de Mestrado, Instituto de Computação – Universidade Federal Fluminense (IC/UFF), dezembro, 2005.

[Nagios 2006] Nagios Project. Website http://nagios.org/, Visitado: novembro 2006.

[Nahrstedt et al 2001] Nahrstedt, K.; Xu, D.; Wichadakul, D.; Li, B. **QoS-Aware Middleware for Ubiquitous and Heterogeneous Environments**. *IEEE Communications Magazine*, Vol. 39, No. 11, p. 140-148, novembro, 2001.

[Navratil & Cottrell 2003] Navratil, J. e Cottrell, R. L., **A Practical Approach to Available Bandwidth Estimation, Passive / Active Measurement** Wks. p. 1-11, La Jolla, CA, abril, 2003.

[Newman 2003] Newman H.B, Legrand I.C, Galvez P, Voicu R, Cirstoiu C, **MonALISA: A Distributed Monitoring Service Architecture**. CHEP 2003, La Jola, California, março, 2003.

[NTop 2006] Ntop Tool. Website http://www.ntop.org/, Visitado: novembro 2006.

[NetPredict 2006] NetPredict.com. **Assess the Ability of Your Network to Handle VoIP before You Commit**. Disponível em http://www.netpredict.com/ pdfs_all/WhitePaper-VoIP.pdf

[Pias & Wilbur 2001] Pias, M. & Wilbur, S. *EdgeMeter: Distributed Network Metering*, In Proceedings of the IEEE Openarch 2001 conference, short paper session, Anchorage, Alaska, abril, 2001.

[Pokraev 2003] Pokraev, S, **Context Aware services**, WAP/D3.0, novembro 2003

[Rangangathan & Campbell 2003] Rangangathan, A. & Campbell, R.H. **A Middleware for Context Aware Agents in Ubiquitous Computing Environments**, *Middleware* 2003, p. 143-161.

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

[Salutation 1999] Salutation Consortium, **Salutation Architecture Specification**. *Version* 2.1 1999.

[Santos 2006] Santos, A. L. G., **Técnicas de Configuração Dinâmica de Arquiteturas de Software**, Dissertação de Mestrado, Inst. de Computação, UFF, 2006.

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

[Schilit 1994] B. Schilit et al., **Context-Aware Computing Applications**, IEEE *Workshop on Mobile Computing Systems and Applications*, Santa Cruz, California, USA, 1994.

[Soap 2000]. **Simple Object Access Protocol** (SOAP) 1.1. W3C Note 08 May 2000. Disponível em http://www.w3.org/TR/SOAP, Visitado: outubro 2006.

[Staab et al 2001]. S. Staab, R. Studer, H.P. Schnurr, Y.Sure, **Knowledge Processes and Ontologies**, IEEE *Intelligent Systems*, Vol. 16, No. 1, fevereiro, 2001.

[Sztajnberg et al 2005] Sztajnberg, A., Corradi, A. M., Santos, A. L., Barros, F. A., Cardoso, L. X. T. e Loques, O. G**., Especificação e Suporte de Requisitos Não-Funcionais para Serviços de Nível Alto**, Minicursos do 23o SBRC, p. 223-279, Fortaleza, CE, 2005.

[Toga 1999] Toga, J. e Ott, J., **ITU-T standardization activities for interactive multimedia communications on packet-based networks: H.323 and related recommendations**, *Computer Networks*, Vol. 31, No. 3, p. 205-223, fevereiro, 1999.

[Wang et al 2001] Wang, N.; Schmidt, D. C.; Kircher, M.; Parameswaran, K. **Adaptive and Reflective Middleware for QoS-Enabled CCM Applications**. IEEE *Distributed Systems Online*, Vol. 2, No. 5, julho, 2001.

[Watson et al 2004] David Watson, G. Rober Malan and Farnam Jahanian, **An extensible probe architecture for network protocol performance measurement**, *Software Practice and Experience*, Vol. 34, 2004.

[Weide 2001] Weide, P. van der. **Information Retrieval**, *Masters Thesis*, *Nijmegen Institute for Computing and Information Sciences*, abril, 2001.

[Weiser 1991] Weiser, M. **The computer for the 21st century**. *Scientific American*, setembro, 1991.

[William et al 1999] William Adjie-Winoto, Elliot Schwartz, Hari Bvalakrishnan, and Jeremy Lilley. **The design and implementation of an intentional naming system**. 17 Th ACM *Symposium on Operating Systems Principles*, 1999.

[Wolski et al 1999] 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, p. 757-768, 1999.

[W3C 2006] W3C, "RDF", Website http://www.w3c.org/RDF/, Visitado: julho 2006.

[W3C 2006a] W3C, OWL Website, http://www.w3c.org/2001/sw/WebOnt/, Visitado: julho 2006.

[W3C 2006b] W3C, XSLT Website, http://www.w3.org/TR/xslt, Visitado: novembro 2006.

[W3C 2006d] **W3C XML *Schema***. Website, http://www.w3.org/XML/Schema, Visitado: julho 2006.

[Xindice 2006] The Apache Software Foundation. **The Apache XML Project**. Disponível na Internet. http://xml.apache.org/xindice/ em 28 de setembro de 2006.

[Yeong et al 1995] Yeong, W., Howes, T., and Kille, S. **Lightweight Directory Access Protocol**. RFC 1777, março 1995.

Apêndice A

Cada tipo de recurso deve ter um Agente de Recurso associado que fornece as informações contidas na sua representação. Com a proposta de representação descrita na Seção 3.1, uma aplicação pode saber, para cada tipo de recurso, suas propriedades e respectivos tipos de dados. Assim, o agente deve disponibilizar essas informações de acordo com os tipos descritos para cada propriedade de recurso. Idealmente, a interface do agente não deve se prender a nenhuma tecnologia particular usada no para coletar as informações. Nas seções a seguir é descrita uma proposta orientada a objetos para os Agentes de Recursos que pode ser usada na implementação do Serviço de Contexto descrito na Seção 3.2.

A.1 Interface para Agentes de Recurso

Na Seção 2.2.2 foram citados alguns exemplos de ferramentas usadas para de monitoração usadas para coletar os valores das propriedades dos recursos. Cada ambiente de operação pode utilizar uma dessas ferramentas que têm interfaces incompatíveis. Por exemplo, o Ganglia é indicado para ambientes de computação baseados em *Clusters* de computadores interligados à redes locais de alto desempenho, enquanto que o NWS já é usado em Grids de computadores espalhados ao redor do globo, visto que ele se encaixa no especificação do *Grid Monitoring Archicteture*. Assim, caso uma aplicação seja codificada especificamente para utilizar diretamente a interface de uma dessas ferramentas, sua portabilidade para outros ambientes torna-se dificultada necessitando alterações em seu código. Com esse problema em mente, é proposta uma interface padronizada para Agentes de Recurso que é independente da ferramenta utilizada no baixo nível.

A.1.1 Padrão de Projeto

Na criação da interface é utilizado o padrão de projeto *Strategy* [Gamma et al 1995] que já é bem conhecido pela literatura. Basicamente, este padrão segue dois princípios: (i) encapsular diferentes estratégias que realizam uma mesma tarefa; e (ii)

desenvolver baseado em uma interface e não em uma implementação particular. Em [Gamma et al 1995] os autores definem o padrão *Strategy* da seguinte maneira:

- Defina uma família de algoritmos, encapsule cada um deles, e faça-os intercambiáveis. O padrão *Strategy* permite trocar os algoritmos sem afetar os clientes que utilizam à interface.

Em nosso contexto para a coleta de informações dos recursos, adaptamos essa descrição para o padrão:

- Defina um conjunto de técnicas para a coleta de informações, encapsule cada uma delas, e faça as intercambiáveis. O padrão *Strategy* permite trocar a maneira como os dados são coletados sem afetar as aplicações que utilizam a interface.

Este padrão permite o desenvolvedor criar sua aplicação como uma coleção de partes intercambiáveis e fracamente acopladas. Em nosso contexto de interface para coleta de informações dos recursos, tem-se uma interface padronizada que pode utilizar diferentes estratégias para a coleta de dados. Assim as aplicações são desenvolvidas utilizando a interface padronizada sem depender de uma estratégia específica.

Para exemplificar a metodologia, Figura 53 mostra o diagrama UML das classes envolvidas na criação de uma interface padronizada para a coleta de informações sobre recursos do tipo *Processing*:

ResourceState é a superclasse das especializações que definirão os atributos referentes às propriedades de cada tipo de recurso, que são descritas de acordo com a representação XML mostrada na Seção 3.1. Assim, para cada tipo de recurso é preciso criar uma especialização de *ResourceState*, que conterá os atributos correspondentes às propriedades descritas na representação do recurso. Por exemplo, na Figura 53 é mostrada a classe *ProcessingState* (que herda *ResourceState*) usada para armazenas as propriedades dos recursos de processamento.

ResourceAgent descreve a interface básica que todo Agente de Recurso deve implementar e que é utilizada pelas aplicações. *AbstractResourceAgent* é uma classe abstrata que implementa alguns métodos comuns a todas as especializações de Agentes de Recurso. Para cada tipo de recurso (por exemplo, processamento e

transporte) deve ser criada uma especialização da classe *AbstractResourceAgent*, através do mecanismo de herança. Ao herdar *AbstractResourceAgent* a classe especializada também deve implementar todos os métodos da interface *ResourceAgent*. Essa implementação é responsável, na classe concreta, por efetuar a leitura dos valores das propriedades do recurso ao qual a especialização se refere. Cada especialização de *AbstractResourceAgent* para um determinado recurso, possui um *ResourceState* associado, que guardará em seus atributos os valores das propriedades do recurso obtidos dos sensores específicos. A Figura 53 mostra algumas especializações de Agentes de Recurso que coletam informações de recursos do tipo *Processing*. São mostrados três classes – *SNMPProcessinAgent*, *GangliaProcessingAgent* e *NWSProcessingAgent* - que encapsulam três ferramentas usadas na coleta dos valores das propriedades do recurso. Todas as especializações retornam os valores das propriedades armazenando-as em uma instância da classe *ProcessingState*. Assim, a aplicação é capaz de obter os dados oriundos de diferentes ferramentas, orientada a uma interface padronizada orientada a objetos.

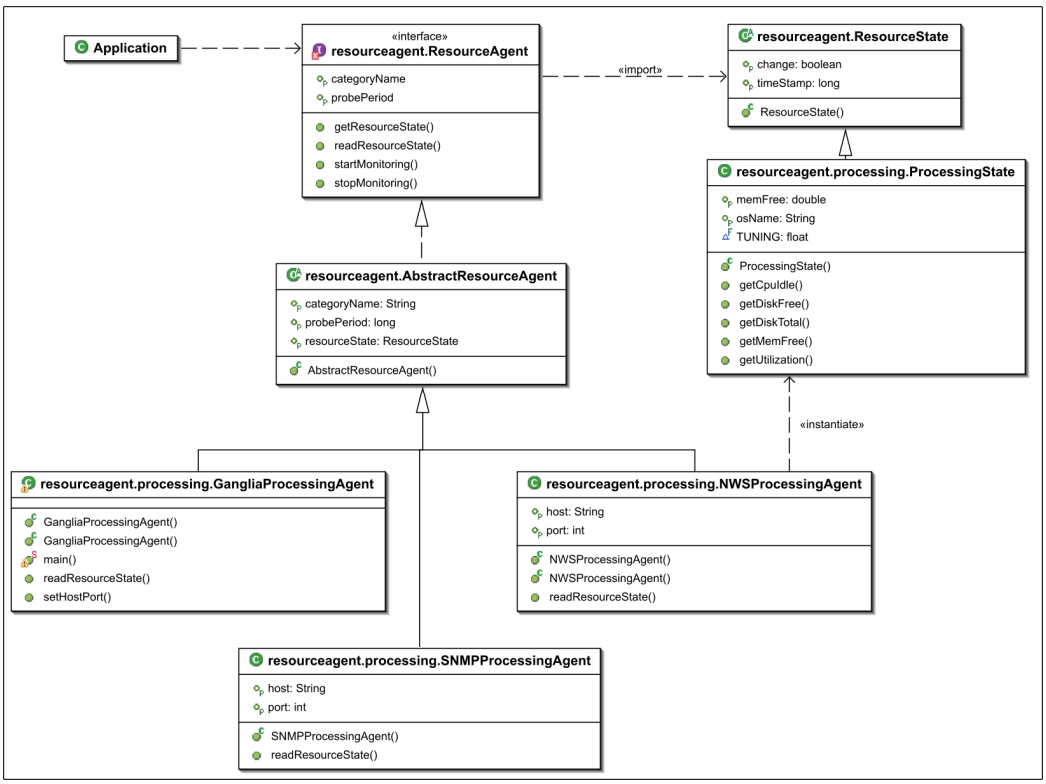


Figura 53 - Diagrama UML da interface de Processamento e suas estratégias.

A classe *Application* representa a aplicação que utiliza a interface *ResourceAgent*. Para isso, ela mantém uma referência para um objeto do tipo *AbstractResourceAgent* que é uma classe abstrata que implementa a interface *ResourceAgent*. Assim, *ResourceAgent* pode conter uma instância concreta de qualquer agente que implemente sua interface.

A.1.2 Criando uma nova estratégia

A criação de uma nova estratégia é simples e se limita a estender *AbstractResourceAgent* sobrescrevendo seus métodos abstratos e implementado os métodos da interface *ResourceAgent*. Todos os agentes associados a um tipo de recurso devem utilizar a mesma especialização de *ResourceState* que contém o estado do recurso. Por exemplo, todos os agentes da categoria *Processing* devem coletar os dados e retorná-los em uma instância da classe *ProcessingState*.

```
1 public class ProcessingState extends ResourceState
2 {
3   public float diskFree;
4   public float memFree;
5   public float cpuIdle;
6 }
7
8 public class ProcessingAgent extends AbstractResourceAgent
9 {
10   public ProcessingAgent()
11   {
12      Super(“Processing”);
13   }
14
15   public ResourceState readResourceState()
16   {
17     ProcessingState processingState = new ProcessingState();
18      processingState.diskFree = 20000.0;
19      processingState.memFree = 256.0;
20      processingState.cpuIdle = 60.0;
21
22      this.setResourceState(processingState);
23      return processingState
24   }
25 }
```

Figura 54 - Exemplo de implementação de uma estratégia

A Figura 54 mostra de forma simplificada a implementação de um agente para recursos do tipo *Processing*. Nas linhas 1-6, é definida a classe *ProcessingState* que estende *ResourceState*. *ProcessingState* contém propriedades especificas para todos os recursos do tipo *Processing*. Seria possível gerar automaticamente esta classe usando a descrição do recurso que tem o nome e o tipo de cada propriedade. Nas linhas 8-25 é mostrado a implementação do agente que retorna os valores das propriedades.

```
1 public class App{
2   public static void main(String args[])
3   {
4      ProcessingState state;
5       ResourceAgent[] agents = { new GangliaProcessingAgent("localhost"),
6                                  new NWSProcessingAgent("localhost", 8869),
7                                  new SNMPProcessingAgent("localhost") };
8
9      for(int i=0; i < agents.length; ++i) {
10        state = (ProcessingState) agents[i].readResourceState();
11        if(state != null) {
12           System.out.println("CPU IDLE: " +  state.cpuIdle );
13           System.out.println("Mem Free: " +  state.memFree );
14           System.out.println("Disk Free: " + state.diskFree );
15        }
16     }
17   }
18 }
```

Figura 55 - Exemplo de utilização da interface de Agentes de Recurso

A Figura 55 mostra um exemplo básico de aplicação que utiliza a interface proposta, para coletar os valores das propriedades de um recurso do tipo *Processing*. Na linha 4 é declarado um objeto do tipo *ProcessingState* (definido na Figura 54, linhas 1-2) que é usado para armazenar os valores das propriedades do recurso. Nas linhas 5-7 é declarado um array de agentes que contém instancias de três especializações de *AbstractResourceAgent:* (i) *GangliaProcessingAgen;*, (ii) *NWSProcessingAgent;* e (iii) *SNMPProcessingAgent* que fazem a coleta dos dados, respectivamente, usando a infra-estrutura disponibilizada pelo Ganglia, NWS e Simple Network Management Protocol - SNMP. Num cenário mais prático, seria possível utilizar alguma metodologia, como o padrão *Factory* [Gamma et al 1995],

que escolheria automaticamente para a aplicação, a instância mais apropriada de agente a ser utilizada para o nó alvo da monitoração.