Universidade do Estado do Rio de Janeiro
Centro de Tecnologia e Ciências
Faculdade de Engenharia
Programa de Pós-Graduação em Engenharia Eletrônica

André Luiz Barbosa Rodrigues

Uma infra-estrutura para monitoramento de sistemas cientes do contexto

Rio de Janeiro
2009

André Luiz Barbosa Rodrigues

Uma infra-estrutura para monitoramento de sistemas cientes do contexto

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre em Ciências em Engenharia Eletrônica, ao Programa de Pós-Graduação em Engenharia Eletrônica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Redes de Telecomunicações e Sistemas Distribuídos.

Orientador: Prof. Dr. Alexandre Sztajnberg

Rio de Janeiro
2009

CATALOGAÇÃO NA FONTE
UERJ/REDE SIRIUS/BIBLIOTECA CTC/B

M395 Rodrigues, André Luiz Barbosa.
Uma infra-estrutura para monitoramento de sistemas cientes do contexto. / André Luiz Barbosa Rodrigues – 2009.
132 f. : il.

Orientador: Alexandre Sztajnberg.
Dissertação (mestrado) – Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia.
Bibliografia: f.132

1. Redes de computação. 2. Sistemas cientes do contexto 3. Sistemas ubíquos e pervasivos (monitoramento) - Dissertações. 4. Engenharia eletrônica – Dissertações. I. Sztajnberg, Alexandre. II. Universidade do Estado do Rio de Janeiro. Faculdade de Engenharia. III. Título.

CDU 004.72.057.4

André Luiz Barbosa Rodrigues

Uma infra-estrutura para monitoramento de sistemas cientes do contexto

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre em Ciências em Engenharia Eletrônica, ao Programa de Pós-Graduação em Engenharia Eletrônica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Redes de Telecomunicações e Sistemas Distribuídos.

Aprovada em: _____

Banca examinadora:

_____
Prof. Dr. Alexandre Sztajnberg (Orientador)
Universidade do Estado do Rio de Janeiro

_____
Prof. Dr. Marcelo Gonçalves Rubinstein
Universidade do Estado do Rio de Janeiro

_____
Prof. Dr. Orlando Gomes Loques Filho
Universidade Federal Fluminense

_____
Prof. Dr. Renato Fontoura de Gusmão Cerqueira
Pontifícia Universidade Católica do Rio de Janeiro

Rio de Janeiro
2009

AGRADECIMENTOS

Agradecemos o apoio parcial da Faperj para a realização deste trabalho.

Agradecemos também o apoio da CAPES, através do PROAP, para nossa a participação no SBRC 2008.

Gostaríamos de agradecer os colegas da UERJ e da UFF, participantes do projeto FAPERJ pela infra-estrutura desenvolvida para a aplicação de tele-saúde.

RESUMO

RODRIGUES, André Luiz Barbosa. *Uma Infra-Estrutura para Monitoramento de Sistemas Cientes do Contexto*. 130 f. Dissertação (Mestrado em Engenharia Eletrônica) - Programa de Pós-Graduação em Engenharia Eletrônica, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2009.

Aplicações ubíquas e pervasivas são cientes do contexto dos recursos utilizados no que diz respeito à disponibilidade e qualidade. Esta classe de aplicações pode se beneficiar de mecanismos para descobrir recursos que atendam aos requisitos não-funcionais desejados, e mecanismos para monitorar a qualidade destes recursos. Neste trabalho é proposta uma arquitetura para dois serviços que deveriam ser incluídos na infra-estrutura de suporte a ser utilizada pelas aplicações mencionadas: um Serviço de Contexto, que provê acesso a informações de contexto, e um Serviço de Descoberta, que permite a descoberta dinâmica de recursos, levando em conta restrições de contexto a serem satisfeitas. Estes serviços se apóiam em Agentes de Recursos, que efetivamente monitoram os recursos e sensores. Uma implementação de referência foi desenvolvida, oferecendo os serviços mencionados na forma de Serviços Web e implementando os Agentes de Recursos empregando um padrão de projeto simples. Para avaliar os serviços estes foram utilizados como infra-estrutura para o desenvolvimento de um sistema tolerante a falhas e uma aplicação de assistência domiciliar remota (tele-saúde). O desempenho dos serviços também foi avaliado.

Palavras-chave: Computação ubíqua. Aplicações cientes de contexto. Serviços de monitoramento e descoberta.

ABSTRACT

Ubiquitous and pervasive applications are aware of the context of the used resources, regarding their availability and quality. This class of application can benefit from mechanisms to discover resources that meet their non-functional requirements and mechanisms to monitor the quality of those resources. We proposed architecture for two services that should be included in the supporting infrastructure used by the mentioned applications: a Context Service that provides access to context information; and a Discovery Service, which allows the dynamic discovery of resources, considering context constraints to be satisfied. These services rely on Resource Agents, which monitor the actual resources and sensors. A reference implementation was developed, providing the mentioned services as Web Services and implementing the Resource Agents using a simple design pattern. To evaluate these services were employed them as the infrastructure to design a fault tolerant system and a remote assisted living application. The performance of the services was also evaluated.

Keywords: Ubiquitous computing. Context-aware application. Discovery and monitoring services.

LISTA DE LISTAGENS

LISTA DE FIGURAS

SUMÁRIO

INTRODUÇÃO

Weiser (Weiser, 1991) define a computação ubíqua (ou pervasiva) como um modelo de interação homem-máquina que visa melhorar o uso do computador através da disponibilização de vários dispositivos capazes de interagir uns com os outros de forma a se tornarem quase imperceptíveis aos usuários finais. Os dispositivos passam a fazer parte do ambiente físico oferecendo serviços com a máxima transparência possível aos usuários finais que interagem com estes dispositivos e serviços para executar tarefas de diferentes tipos. Como exemplo de aplicações desenvolvidas com estas características de projeto pode-se citar as casas inteligentes, ambientes de gerenciamento hospitalar (Rialle et al., 2002; Stefanov et al., 2004) e sistemas de publicação de conteúdo (Mühl et al., 2004).

Aplicações ubíquas são cientes de seus contextos de execução, pois levam em consideração, não apenas os requisitos funcionais (domínio do problema) que as mesmas têm que prover, mas também o contexto em que o usuário e (ou) a infra-estrutura de hardware e software se encontram.

Dey (Dey, 2000) define contexto como "*qualquer informação que pode ser usada para caracterizar a situação de uma entidade. Uma entidade é uma pessoa, lugar ou objeto que é considerado relevante para a interação entre um usuário e uma aplicação, incluindo os próprios usuários e aplicações.*" De todas as definições encontradas na literatura (Schilit et al., 1994; Pascoe, 1998; Chen e Kotz, 2000; Schmidt et al., 1999), esta é a definição das mais genéricas, deixando a cargo do projetista da aplicação decidir o que faz parte ou não do contexto de acordo com sua relevância face aos requisitos que as aplicações devem atender. Portanto, informações de contexto podem ser o percentual de uso de uma CPU, a latência de uma rede, os batimentos cardíacos de um paciente ou até mesmo a localização de uma pessoa.

A disponibilidade e qualidade das aplicações cientes do contexto são afetadas pelo estado dos recursos presentes neste tipo de ambiente computacional. Nessa classe de aplicações, a heterogeneidade de serviços e dispositivos fornecidos por diferentes fabricantes constitui um desafio à construção de aplicações cientes do contexto, pois dispositivos de uma mesma classe podem exigir diferentes tecnologias e protocolos de comunicação. Além disso, novos serviços e dispositivos podem ser adicionados e removidos a qualquer instante, bem como a qualidade dos mecanismos de comunicação utilizados pelos dispositivos pode sofrer variações ao longo da execução das aplicações. Estes fatores exigem que estas aplicações

tenham a capacidade de se adaptar ao contexto atual de execução de forma a poderem continuar provendo o nível de qualidade de serviço esperado por seus usuários.

Para lidar com a adição e remoção de dispositivos e serviços e com as variações nos níveis de qualidade dos mesmos, aplicações cientes do contexto podem requerer adaptações motivadas por mudanças nos requisitos não-funcionais das aplicações, por meio de regras pré-definidas ou por políticas de operação, de desde que cada um desses fatores leve em consideração o estado dos recursos presentes no ambiente de execução. No entanto, para que estas mudanças no contexto de execução das aplicações possam ser identificadas, e recursos com melhor qualidade de serviço possam ser localizados e utilizados em substituição aos atuais, faz-se necessário a existência de uma infra-estrutura de software que permita o monitoramento e a descoberta de recursos neste tipo de ambiente a partir de informações contextuais.

Atividades de descoberta e monitoramento de recursos são recorrentes em sistemas cientes do contexto em especial em sistemas ubíquos e pervasivos. É desejável que estas atividades possam ser tratadas com alto nível de abstração, para que as especificidades existentes entre diferentes fornecedores de dispositivos e serviços possam ser tratadas sem mudanças no código fonte das aplicações que os utilizam. Além disso, a implantação e gerenciamento dos serviços têm que ser realizados de forma uniforme através de um modelo consistente para que possa ser possível não apenas trocar um recurso que se encontra abaixo do nível de qualidade desejado, mas também selecionar um que tenha o melhor nível de qualidade possível naquele momento.

Para que estas atividades de descoberta e monitoramento de serviços e recursos possam ser realizadas é necessário prover alguns serviços básicos e determinar alguma padronização na forma como estas tarefas são executadas. Por exemplo, para que os recursos e serviços possam ser descobertos, monitorados e utilizados, os mesmos precisam ser descritos e então publicados ou registrados em um serviço de diretórios. Sem este passo os clientes não seriam capazes de expressar o que desejam descobrir, nem capazes de identificar onde encontrar o que desejam.

A partir do trabalho de (Cardoso, 2006a) apresentamos um modelo de três serviços para o suporte de aplicações cientes do contexto: um Serviço de Contexto, que provê acesso às informações de contexto; um Serviço de Descoberta, que permite a descoberta de recursos dinâmicos considerando restrições de contexto a serem satisfeitas; e um Serviço de Diretório onde os recursos e serviços registram suas características de maneira padronizada. Estes

serviços contam com Agentes de Recursos, os quais monitoram os recursos coletando dados dos sensores associados aos recursos sendo monitorados. Cada classe de recurso está associada a uma descrição XML, em meta-nível, permitindo a inclusão de qualquer novo tipo de recurso na infra-estrutura.

A arquitetura de software dos serviços foi projetada como uma infra-estrutura (um *framework*) de objetos que se comunicam através de um protocolo neutro de linguagem, baseado na troca de mensagens XML (W3C1, 2008). Uma implementação de referência provê os serviços mencionados como Serviços Web (W3C2, 2008) e a implementação dos Agentes de Recursos faz uso de padrões de projetos (Gamma et al., 1995). Esta implementação de referência foi avaliada através do desenvolvimento de Agentes de Recursos para CPU, memória, temperatura e dispositivos de tele-medicina que foram utilizados em algumas aplicações de computação ubíqua e cientes de contexto. Além disso, medidas de desempenho da implementação de referência foram coletadas com intuito de analisar a escalabilidade da mesma.

Organização do texto

O trabalho está estruturado da seguinte forma: no Capítulo 1 são apresentados conceitos básicos relacionados com o desenvolvimento de aplicações cientes do contexto tais como as diferentes formas de representação do contexto e de recursos, questões relacionas ao monitoramento, descoberta e gerenciamento e adaptação de recursos. No Capítulo 2 são listados alguns trabalhos relacionados com o monitoramento de recursos e infra-estruturas para o desenvolvimento de aplicações ubíquas efetuando algumas comparações entre os mesmos. No Capítulo 3 é apresentada a nossa proposta, são abordados os serviços, as interações existentes entre eles e como os dados do contexto são modelados. No Capítulo 4 é apresentada a implementação de referência e detalhes da API e das extensões possíveis para a utilização dos serviços. Também é feita uma discussão sobre o desempenho da implementação de referência. No Capítulo 5 são apresentados exemplos de aplicação utilizando os serviços para validar nossa proposta: um cenário de tolerância a falta e um cenário de tele-saúde. No Capítulo 6 são apresentadas as conclusões e trabalhos futuros.

CAPÍTULO 1 CONCEITOS BÁSICOS

Este capítulo aborda conceitos básicos relacionados às aplicações cientes do contexto. São apresentadas diferentes maneiras de representar os atributos de contexto e dos recursos sendo monitorados. Questões referentes à como efetuar o monitoramento dos recursos são levantadas apresentando as vantagens e desvantagens de cada possível abordagem. Os elementos necessários à descoberta de recursos bem como as técnicas que se pode utilizar para localizar recursos em ambientes heterogêneos e questões relacionadas à adaptação das aplicações cientes do contexto também são abordadas neste capítulo.

1.1 Representação dos atributos de contexto e de recursos

A representação de um recurso e seus atributos está diretamente associada à descoberta e monitoração do contexto no qual se encontra o recurso. É necessário disponibilizar uma representação das informações sobre seu tipo, atributos e características funcionais para que o recurso possa ser utilizado pelos clientes da infra-estrutura. De acordo com (Strang, 2004) e (Balakrishnan, 2005) os modelos de dados relativos ao contexto podem ser agrupadas nos seguintes grupos:

- **Modelos baseados em pares objeto-valor**. Neste modelo um conjunto de pares objeto-valor descreve os atributos contextuais e as consultas são realizadas casando-se os padrões presentes nos recursos.
- **Modelos gráficos.** Neste modelo pode-se utilizar a UML (*Unified Modeling Language*) (UML, 2007) como ferramenta para representar as informações de contexto de forma gráfica, pois foi concebida para ser genérica e vem se mostrando apta a expressar diversos tipos de modelos. Como exemplo deste tipo de abordagem se pode citar ContextUML (Sheng e Benatallah, 2005). Através desta linguagem é possível descrever diferentes tipos e fontes de informação contextuais, suas restrições e dependências entre as mesmas.
- **Modelos baseados em lógica.** Neste tipo de abordagem utiliza-se um sistema baseado em regras onde os dados de contexto são expressos como fatos. Assim, através de inferência, são derivados novos fatos que auxiliam no processo futuro de descoberta de recursos. Como o casamento de padrões é realizado por meio de prova, o mesmo acaba sendo lento e complexo, dificultando a utilização desta técnica.

- **Modelos orientados a objeto.** Neste tipo de modelo utilizam-se os mesmos conceitos do paradigma de orientação a objetos. Assim os detalhes inerentes à implementação do recurso tais como o processamento e a representação dos atributos não ficam visíveis a outros componentes. As consultas aos recursos são realizadas através de interfaces previamente conhecidas pelos seus clientes.
- **Modelos de esquemas de marcação.** Neste tipo de abordagem são utilizadas linguagens de marcação para descrever de forma hierárquica os recursos e seus atributos através de tags. RDF/S (*Resource Description Framework Schema*) (W3C4, 2008) pode ser utilizado nesta abordagem para codificar modelos nesta abordagem.
- **Modelos baseados em ontologias.** Esta abordagem utiliza ontologias para descrever os atributos dos recursos bem como os relacionamentos existentes entre os mesmos. Devido ao seu poder de expressividade, os recursos presentes em um ambiente pervasivo podem ser classificados de forma eficaz. O principal entrave na utilização desta técnica é a definição de uma ontologia que seja consistente com o domínio da aplicação. O trabalho de (Strang e Linnhoff-Popien, 2004) mostra que modelos baseados em ontologias são os mais expressivos e que melhor descrevem as propriedades de sistemas cientes do contexto.

Neste trabalho são utilizados elementos (i) do modelo de pares objeto-valor, (ii) um sistema baseado nos tipos dos recursos, onde são utilizados alguns conceitos (iii) da modelagem orientada a objetos como herança e composição. Além disso, o modelo proposto pode ser estendido para que uma especificação da ontologia utilizada para descrever os recursos possa ser empregada, ficando a cargo de o projetista fazer uso ou não de um serviço de descoberta baseado em ontologias (uma breve discussão sobre este assunto é desenvolvida na Seção 4.4 e, depois, no Capítulo 6, concluindo o trabalho).

1.2 Monitoramento de recursos

O monitoramento do contexto de execução e dos recursos utilizados é uma atividade fundamental para a operação de aplicações cientes do contexto, tais como aquelas do domínio de computação ubíqua. (Baldauf et al., 2007) e (Chen e Kotz, 2000) destacam três formas de monitoramento e coleta de dados dos recursos: uma através de acesso direto aos sensores, outra baseada em *middleware* e uma terceira baseada em servidores de contexto.

Na primeira forma os dados são coletados diretamente dos sensores de um recurso, o que traz um alto nível de acoplamento entre os recursos e as aplicações que necessitam ter acesso aos seus dados diminuindo sensivelmente o reuso dos componentes. Nesta abordagem, se o cliente necessitar utilizar um conjunto de recursos construídos com diferentes tecnologias terá de lidar com as especificidades de cada conjunto de sensores que compõem o recurso. Isso acaba trazendo maior complexidade para a arquitetura do cliente dificultando a implementação do mesmo.

Na segunda técnica é utilizada uma arquitetura em camadas padronizadas que são interpostas entre os sensores dos recursos e seus clientes. Desta forma esconde-se dos clientes a complexidade e os detalhes técnicos envolvidos na comunicação com cada tipo de sensor. Com isso a complexidade envolvida na construção dos clientes é diminuída e o espectro de utilização dos recursos é ampliado. Quando uma aplicação precisar acessar um novo recurso, de determinado tipo, e que possui um conjunto de sensores diferente dos que foram utilizados pela aplicação no passado, bastará que a equipe responsável pela manutenção do *middleware* desenvolva a camada responsável pela comunicação com este novo conjunto de sensores. Como a aplicação interage com uma camada do *middleware* que a isola do sensor propriamente dito, nenhuma modificação no código da aplicação será necessária para que a mesma possa obter dados do novo recurso.

Finalmente, na terceira técnica um ou mais servidores distribuídos são utilizados para prover acesso aos dados de contexto. Esta abordagem estende a segunda técnica introduzindo um elemento (o servidor de contexto) que disponibiliza os dados de contexto lidando com a complexidade de múltiplos acessos aos múltiplos recursos presentes no ambiente. Além disso, coletar dados de contexto provenientes de múltiplos recursos, lidar com os possíveis diferentes protocolos de comunicação utilizados pelos recursos, assim como lidar com restrições de qualidade de serviço pode ter um custo computacional elevado sem a ajuda de um servidor de contexto. Considerando-se operações de contexto realizadas em dispositivos móveis, com limitações de recursos, a utilização de um servidor de contexto é ainda mais atraente.

1.3 Descoberta de recursos

A descoberta de recursos é um aspecto importante em um ambiente de computação ubíquo, pois recursos podem ser adicionados e removidos a qualquer instante, ou podem

passar a apresentar um estado cuja qualidade de serviço não seja a requerida pelos utilizadores dos recursos. Sendo assim, se faz necessária a existência de uma infra-estrutura que permita a localização de recursos segundo variados critérios. Novos recursos podem ser localizados em resposta a remoção de um recurso previamente existente. Um novo recurso pode ser selecionado em substituição a outro, já em utilização, por apresentar melhores níveis de qualidade de serviço. A forma como a descoberta de recursos será realizada depende da forma como o contexto é modelado (vide Seção 1.1).

Para que os recursos possam ser utilizados é necessário que os mesmos estejam catalogados e registrados em algum repositório que poderá ser consultado por clientes que precisam encontra algum recurso para utilização.

Uma abordagem distribuída para o repositório é possível. Por exemplo, cada recurso seria responsável por monitorar um canal de comunicação de grupo (*multicast*) para receber requisições dos clientes. Ao receber a requisição cada recurso verificaria se o mesmo tem as características desejadas e enviaria para o cliente a sua referência. O cliente seria responsável por filtrar e selecionar uma das respostas e, assim, poderia passar a utilizar o recurso selecionado. Embora os procedimentos sejam realizados por cada recurso, a infra-estrutura de comunicação de grupo precisa ser implantada e a referência ao canal precisa ser conhecida. Além disso, o gerenciamento de canais para comunicação de grupo pode ser complexo e dificultar o uso de repositórios administrados por entidades diferentes.

De uma forma geral uma arquitetura centralizada de repositório é utilizada, aliada a mecanismos de federação, quando necessário. Na arquitetura centralizada os recursos são registrados no repositório antes de se tornarem disponíveis e o endereço do repositório é previamente conhecido pelos recursos e pelos clientes. Sendo assim, é necessário que uma rotina de registro seja executada para que os recursos, assim como seus tipos, sejam registrados nesse repositório. Uma possível abordagem é delegar a um administrador de sistemas a realização deste registro. A outra abordagem é a de os próprios recursos realizarem seu registro, como se dá em sistemas de middleware como CORBA e Jini no registro de seus componentes. Isso levanta algumas questões quanto à segurança. Na primeira abordagem, o administrador pode definir qual serviço poderá ser registrado além de determinar quem poderá utilizá-lo, assim como quando a utilização poderá ser feita. Na segunda abordagem, pode-se utilizar ou não comunicação segura através de certificados digitais dependendo do nível de segurança que se deseja manterá (MyProxy, 2009). Independente da entidade ou como se efetua o registro, é necessário o conhecimento prévio da localização deste repositório

em ambas as abordagens para que o registro seja feito e, posteriormente, uma referência dos recursos seja obtida para que os mesmos possam ser utilizados. Um tratamento profundo sobre aspectos de segurança não será objeto desta dissertação.

Cabe uma observação sobre o registro dos tipos de recursos. Ao se registrar um tipo de recurso no repositório a "fôrma" geral para todos os recursos daquele tipo passa a ser facilmente recuperada. Assim, é possível adicionar mais um nível de segurança para os serviços que se utilizam do repositório permitindo o registro de recursos apenas para aqueles cujas informações de registro obedecem às restrições do tipo associado.

As consultas efetuadas no processo de descoberta de recursos podem ser feitas baseando-se em valores fixos previamente conhecidos e que, a princípio não mudam durante a execução de um recurso (como, por exemplo, à memória de um computador) ou então baseadas em informações de contexto, tais como a localização de uma pessoa ou o nível de utilização de uma CPU. Além disso, é necessário que as consultas possam ser realizadas através de uma interface uniforme.

O modelo adotado neste trabalho separa a função do repositório de recursos da função de descoberta. O repositório é centralizado, nomeado Serviço de Diretório de Recursos (SDR), e tem o objetivo de manter o registro dos tipos de recursos catalogados e a referência aos recursos efetivamente registrados. Um Serviço de Descoberta (SD) é utilizado para identificar a disponibilidade e localizar recursos. Este serviço expõe uma interface uniforme de consulta para os clientes da infra-estrutura e utiliza o SDR para localizar recursos.

1.4 Adaptação de aplicações cientes do contexto

Uma vez que (i) a forma de representação do contexto esteja definida, (ii) os mesmos possam ser localizados para posterior utilização e (iii) que os recursos presentes no ambiente sejam passíveis de monitoramento, as aplicações cientes do contexto têm à disposição a infra-estrutura básica para realizar adaptações necessárias em resposta às mudanças de contexto ao longo do tempo. Estas adaptações deverão levar em consideração as restrições de qualidade de serviço e as políticas de adaptação definidas para a aplicação de tal forma que a mesma possa continuar provendo seus serviços dentro dos critérios de qualidade e operação especificados pelo usuário. É desejável que o projeto das aplicações que precisam realizar adaptações possa contar com conceitos e mecanismos em alto nível implementar tais características.

Uma forma de especificar as políticas de adaptação é através de contratos de qualidade de serviços (Curty, 2002; Loyall, 1998). Os serviços que implementam os requisitos funcionais da aplicação podem ser descritos através de contratos e então efetuar a associação dos mesmos às restrições de qualidade. Os contratos e propriedade de qualidade de serviço podem ser descritos utilizando-se linguagens específicas para a descrição de níveis de qualidade tais como QDL (*Quality Description Languages*) (Pal et al., 2000) e QML (Frølund, 1998).

Outra forma de descrever políticas de adaptação é descrever as restrições de qualidade como invariantes e associar a elas as adaptações necessárias quando as invariantes não são respeitadas (Garlan, 2004).

Os conceitos e modelo de serviços apresentados neste trabalho foram propostos de forma a facilitar integração com aplicações e subsistemas de gerenciamento de adaptações. A separação de interesses e a facilidade de especialização dos elementos do modelo para permitir a integração com diferentes mecanismos e protocolos de interação são discutidas no Capítulo 3 e 4. No Capítulo 4 é apresentado um exemplo de integração da infra-estrutura abordada nesta dissertação e o *framework* CR-RIO (Curty, 2002) que é uma infra-estrutura capaz de realizar adaptações baseadas em contratos.

CAPÍTULO 2 TRABALHOS RELACIONADOS

Neste capítulo são apresentados trabalhos e propostas relacionadas. Dividimos a apresentação discutindo (i) ferramentas e protocolos padronizados relacionados com monitoramento, registro e descoberta de recursos em ambientes distribuídos – diretamente associadas à nossa proposta e (ii) infra-estruturas para desenvolvimento de sistemas cientes do contexto, discutindo aspectos relevantes das diferentes abordagens adotadas por estas infra-estruturas e como as mesmas integram e utilizam serviços de suporte a contexto.

2.1 Ferramentas para registro, descoberta e monitoramento de recursos

Nesta seção destacamos propostas e ferramentas disponíveis para gerenciar o registro, a descoberta e o monitoramento de recursos. Discutimos as principais abstrações de cada uma, seus pontos positivos, e posteriormente como nossa proposta se encaixa entre as mesmas.

2.1.1 SLP

O SLP (*Service Location Protocol*) (Veizades et al., 2009) é um protocolo de descoberta de serviços padronizado pelo IETF (*Internet Engineering Task Force*). Através deste protocolo é possível a descoberta de serviços disponíveis em rede de forma automatizada além de se poder obter a localização dos serviços e as propriedades dos mesmos.

O protocolo estabelece a existência de três elementos: *User Agents*, *Services Agents* e *Directory Agents*. Os *User Agents* são os clientes dos serviços e que efetuam o processo de descoberta dos serviços. Os *Services Agents* são os serviços em si que anunciam através do protocolo suas localizações bem como seus tipos e outros atributos específicos dos serviços. Os *Directory Agents* consistem de agentes que servem como um ou mais serviços de diretórios onde os *Services Agents* se registram e onde os *User Agents* podem efetuar consultas com o intuito de localizar os serviços com as características desejadas.

SLP especifica que a presença de *Directory Agents* é opcional. Ou seja, é um protocolo onde a presença de um ou mais serviço de registro centralizado é opcional. Quando

há a presença destes elementos na rede o processo de descoberta é chamado de detecção passiva e quando estes elementos não estão presentes o processo é chamado de detecção ativa.

No processo de detecção ativa os *User Agents* enviam periodicamente requisições *multicast* com as propriedades desejadas dos serviços. Estas requisições são interpretadas por aqueles *Services Agents* cujas propriedades correspondem àquelas desejadas pelos *User Agents* e respondem através de comunicação *unicast* aos *User Agents*.

No processo de detecção passiva os *Directory Agents* coletam de forma indefinida as informações anunciadas pelos *Services Agents* de forma a compor um diretório de serviços. Os *User Agents* podem então acessar os *Directory Agents* através de comunicação *unicast* para localizar os serviços registrados.

Os *User Agents* podem consultar um servidor DHCP (Droms, 1997) para localizar os *Directory Agents* ou podem ouvir os anúncios *multicast* que são efetuados pelos mesmos periodicamente.

Um serviço de diretório e de descoberta de recursos poderia ser construído tendo como base o SLP. No entanto, apesar do formato das mensagens que descrevem as propriedades dos *Services Agents* ser padronizada (Guttman et al., 1999), as propriedades são todas constituídas de seqüências de caracteres não possuindo, portanto, um sistema de tipos forte. O modelo de contexto que mais se adéqua a este tipo de abordagem é o de pares chave/valor. Caso fosse necessária a realização de consultas mais sofisticadas os recursos presentes no SLP não seriam suficientes recaindo sobre o serviço de diretório e descoberta de recursos a responsabilidade de adaptar estas operações de pesquisa de mais alto-nível para as primitivas do SLP.

2.1.2 Jini

Jini (Jini, 2009; Rodrigues, 2003) é uma especificação criada pela Sun Microsystems centrada na tecnologia Java cujo objetivo é prover uma infra-estrutura para a construção de sistemas facilmente plugáveis ao ambiente de execução de um sistema computacional. Originalmente a idéia por trás desta infra-estrutura era trazer para o ambiente de rede o conceito de *plug and play* presente nos sistemas operacionais modernos, onde dispositivos tais como impressoras, scanners, etc. são automaticamente reconhecidos e cujos *drivers* são

automaticamente instalados pelos sistemas operacionais. Através da infra-estrutura proposta, dispositivos seriam capazes de serem utilizados sem que houvesse intervenção manual por parte dos usuários. No entanto, o conjunto de especificações definido acabou não alcançando o sucesso previsto quando de sua criação devido a pouca adesão de fabricantes de hardware na especificação das interfaces padrões. Isso se deveu principalmente ao fato de ser centrada em Java (todos os componentes da infra-estrutura têm que ser implementados nesta plataforma) exigindo a padronização de interfaces a partir das quais os clientes se comunicariam com os dispositivos e serviços. Posteriormente Jini passou a ser utilizado com sucesso em *grids* (Foster, 2001; Baker e Smith, 2001; Furmento, et al., 2002) onde o *Service Lookup* é utilizado como serviço de diretório.

A infra-estrutura é centrada em um serviço de nomes denominado *Lookup Service* no qual os serviços se registram formando comunidades de serviços.

Clientes e serviços podem localizar o *Lookup Service* através de anúncios *multicast* periódicos feitos pelo mesmo ou através de requisições *multicast* feitas por clientes e serviços. É possível também a comunicação direta com o *Lookup Service* através de requisições *unicast* desde que a localização do mesmo seja previamente conhecida. Esta característica é importante, pois mensagens *multicast* podem ser bloqueadas pelos roteadores presentes na internet o que poderia impedir a comunicação com *Lookup Services* localizados em sub-redes diferentes. Através do processo de comunicação *unicast* diferentes *Lookup Services* podem ser interligados formando federações de serviços, aumentando o número de serviços disponíveis para os clientes.

Uma vez que o *Lookup Service* seja conhecido, a comunicação com os serviços e clientes passa a ser realizada através de um objeto Java que implementa a interface *ServiceRegistrar*. Este objeto atua como um *proxy* (intermediário) entre os clientes e serviços e o *Lookup Service*.

Através do método *register* presente nesta interface os serviços publicam no *Lookup Service* atributos que os descrevem e objetos *proxies* a partir dos quais os clientes poderão se comunicar diretamente com os serviços registrados. O processo de registro de serviços é baseado em um modelo de *leasing*, onde o *Lookup Service* informa o tempo que o serviço ficará registrado. Periodicamente os serviços deverão renovar seu tempo de permanência no *Service Lookup*, caso contrário, terão suas referências removidas do *Service Locator*.

Os clientes utilizam o método *lookup* da interface *ServiceRegistrar* para localizar

serviços baseados em seus atributos ou nas interfaces que os serviços implementam. Além disso, os clientes podem requisitar serem informados de mudanças nos atributos dos serviços bem como do surgimento e remoção de serviços num modelo baseado em eventos.

Jini poderia ser utilizado na construção de um serviço de diretório e de descoberta de recursos, dadas as funcionalidades presentes em sua arquitetura centrada no *Service Lookup*. Uma vez que as consultas submetidas ao *Service Lookup* são baseadas nas interfaces que os serviços possuem podendo ser realizadas consultas baseadas nos tipos, subtipos, supertipos e propriedades polimórficas, o modelo de contexto que mais se enquadra neste tipo de abordagem é o orientado a objetos. No entanto, como Jini é fortemente atrelado à plataforma Java, a comunicação a ser utilizada entre os serviços e o *Service Lookup* no processo de registro deverá ser realizada obrigatoriamente através de RMI (RMI, 2008), assim como a descoberta de serviços por parte dos clientes junto ao *Service Lookup*. Além disso, os *proxies* (mas não necessariamente os serviços em si) deverão ser construídos com foco na plataforma Java.

Cabe citar que a comunicação entre os *proxies* e os serviços que os mesmos representam junto ao *Service Lookup* bem como a comunicação entre os *proxies* e os clientes dos serviços não necessita ser realizada através de RMI.

2.1.3 NWS

NWS (*Network Weather Service*) (Wolski et al., 1999) é um sistema projetado para produzir previsões sobre o desempenho de redes e recursos computacionais de forma distribuída baseado em dados históricos provenientes do sensoriamento dos recursos sendo monitorados.

Quatro componentes principais fazem parte da arquitetura do NWS, a saber:

- processo de estado persistente: é responsável por armazenar os dados coletados disponibilizando uma interface a partir da qual os dados armazenados podem ser retornados. Os dados coletados são armazenados em formato texto e são associados ao recurso que originou a medida e opcionalmente a uma estampa de tempo. Assim que os dados são enviados para este componente, os mesmos são escritos para um armazenamento persistente que é baseado numa fila

circular. Desta forma, os dados coletados não são armazenados de forma indefinida ao longo do tempo.

- Processo servidor de nomes: este elemento é uma extensão do processo de estado persistente e é responsável por prover um sistema de nomeação a partir do qual os recursos sendo monitorados podem ser localizados. Todos os componentes da infra-estrutura registram-se neste elemento. Podendo originalmente existir apenas um servidor de nomes. Posteriormente foi introduzido o LDAP (Wahl, 1997) onde é possível ter vários servidores de nomes cujos dados podem ser replicados entre si.

- Processo de sensoriamento: responsável por coletar as medidas dos recursos a serem monitorados. A cada medida coletada é anexada uma estampa de tempo bem como o nome do recurso que originou a medida. Cada processo de sensoriamento pode ser responsável por medir diferentes grandezas.

- Processo de previsão: responsável por predizer os níveis de serviço que um ou mais recursos deverão apresentar no futuro baseado numa série temporal de dados coletados pelo processo de sensoriamento e armazenados pelo processo de estado persistente. O processo de previsão utiliza vários modelos de previsão com a mesma série temporal de dados. Os erros de previsão são calculados e armazenados para utilização posterior pelo processo de previsão. O modelo que apresenta o menor erro ao longo do tempo é que passa a ser o modelo a ser adotado em um determinado período de tempo. Ou seja, o modelo de previsão varia conforme a taxa erro aumenta ou diminui. Novos modelos de previsão podem ser incorporados aos já existentes através de módulos aderentes a uma interface padronizada pela infra-estrutura.

Os processos de sensoriamento do NWS poderiam ser utilizados para coletar dados dos recursos monitorados no ambiente de aplicações cientes do contexto. O *middleware* presente no serviço de contexto poderia ter uma camada de software responsável por abstrair os detalhes da interação com estes processos e as aplicações cientes do contexto. Além disso, o processo de previsão do NWS poderia ser utilizado na adaptação das aplicações cientes do contexto onde os melhores recursos para a execução das aplicações poderiam ser previstos com antecedência, auxiliando no processo de pré-alocação de recursos.

2.1.4 Ganglia

Ganglia (Massie et al., 2004; Ganglia, 2008) é um sistema de monitoramento distribuído voltado para sistemas baseados em grades e para *clusters*. Os estados dos elementos dos *clusters* são monitorados através de um processo de anúncio *multicast* no qual os dados são trafegados em formato XML/XDR. Os *clusters* monitorados são agrupados em uma estrutura de árvore de conexões ponto a ponto através da qual formam federações de *clusters*. Como a infra-estrutura é voltada para sistemas de alto desempenho, Ganglia procura utilizar estruturas de dados compactas e algoritmos que procuram causar pouca sobrecarga e alta concorrência sobre os recursos monitorados.

Cada nó que constitui o *cluster* monitora seu estado interno e o informa em um endereço *multicast* bem conhecido pelos elementos do *cluster* sempre que seu estado interno sofre alguma modificação significativa. Desta forma, cada um dos nodos presentes no *cluster* coleta os dados de monitoramento de todos os outros nós. Assim cada elemento do *cluster* acaba tendo uma visão aproximada dos demais elementos do *cluster*.

Através da estrutura em árvore que conecta os diferentes *clusters* os dados de monitoramento vão sendo agrupados e informados para os nós superiores da árvore através de um mecanismo de varredura periódica.

O elemento do sistema responsável por coletar as métricas de desempenho é o *gmond* (*Ganglia monitoring deamon*). Este elemento executa em cada nodo do *cluster* e informa os dados coletados em formato XML/XDR. Este *deamon* possui um *thread* responsável por coletar e por publicar os dados do nodo local e por manter o nodo no *cluster* através do envio de mensagens específicas para isso periodicamente. Há outro thread neste *deamon* responsável por coletar os dados dos outros nodos do *cluster* e armazená-los localmente. Nenhum estado é armazenado de forma persistente pelo *gmond*. Graças à utilização de mensagens *multicast* não há a necessidade de um elemento central no qual os nodos dos *clusters* tenham que se registrar para localizar outros nodos. Novos nodos são descobertos automaticamente bem como nodos com problemas de execução são automaticamente eliminados pelos demais nodos dos *clusters*. No entanto, as árvores de *clusters* são criadas de forma manual, devido às limitações impostas pelos roteadores.

O elemento *gmetad* (*Ganglia Meta Deamon*) é responsável por manter a federação de *clusters*. O *gmetad* varre os nós das árvores ligados entre si para se certificar de que os

mesmos encontram-se ativos, coletam os dados resumidos dos *clusters* ligados a ele e grava os dados coletados através da ferramenta RRD (*Round Robin Database*). Esta ferramenta armazena séries de dados temporais com diferentes granularidades (anos, meses, minutos, etc.) e os apresenta de forma tabular e gráfica.

Através de bibliotecas disponibilizadas pelo sistema ou através do aplicativo *gmetric* os dados de monitoramento podem ser acessadas pelas aplicações clientes.

De forma similar ao NWS o elemento *gmond* poderia ser utilizado para coletar dados dos recursos monitorados. A camada do *middleware* responsável pela interação com o Ganglia poderia ser implementada como mais um elemento do(s) cluster(s) e com isso, teria as métricas de cada elemento presente nos *cluster(s)*.

2.1.5 GMA

GMA (*Grid Monitoring Architecture*) (Tierney et al., 2002) é uma arquitetura de referência para monitoramento do desempenho de sistemas de computação em grade composta por três elementos: produtores, consumidores e serviço de diretório.

O serviço de diretório serve como um ponto inicial a partir do qual os consumidores e produtores podem localizar uns aos outros. Ou seja, um consumidor pode localizar produtores que lhe interessam ou produtores podem localizar consumidores interessados neles. Quando os consumidores e produtores se registram no serviço de diretório eles os fazem publicando informações sobre si que podem ser utilizadas como filtros nas consultas a serem realizadas posteriormente. Uma vez que um elemento descubra o outro a comunicação passa a ser efetuada diretamente entre os mesmos sem o intermédio do serviço de diretório.

A comunicação entre produtores e consumidores pode ser realizada através de três modos: através de um modelo centrado em eventos, através de um modelo de consulta/resposta ou através de um modelo de notificação. Em todos estes modelos tanto os produtores quanto os consumidores podem assumir o papel de servidor ou de cliente.

No modelo baseado em eventos o cliente contata o servidor indicando os eventos sobre os quais ele se interessa podendo informar também parâmetros que irão guiar o processo de transferência de dados entre eles, tais como o tamanho de *buffer*, onde os dados deverão ser entregues, como codificar/decodificar os dados, etc. Quando os eventos ocorrem os dados são enviados para os interessados. Qualquer um dos envolvidos na comunicação

pode finalizar a comunicação.

O modelo de consulta/resposta é inicializado apenas pelos consumidores. O consumidor contata o produtor de forma semelhante ao modelo baseado em eventos. O produtor envia os dados correspondentes aos eventos informados para o cliente uma única vez e termina a comunicação.

O modelo de notificação é efetuado apenas pelos produtores. Neste modelo o produtor transfere todos os eventos de desempenho para o cliente em uma única notificação.

A arquitetura de referência do GMA não especifica os protocolos de controle nem o formato dos dados trocados bem como o protocolo de comunicação em rede. Sendo uma arquitetura de referência, GMA pode ser construída de diferentes maneiras utilizando diferentes tecnologias de comunicação. Como exemplo, Jini vem sendo largamente utilizado na construção de produtores e consumidores tendo seu elemento *Service Lookup* utilizado como serviço de diretório. Ao mesmo tempo em que é uma arquitetura flexível possibilitando a criação de aplicações de monitoramento para diferentes aplicações de computação em grades, esta mesma flexibilidade torna-se um problema no que diz respeito à interoperabilidade entre diferentes produtos baseados nesta arquitetura. Dada a heterogeneidade das aplicações cientes do contexto, a falta de interoperabilidade pode se tornar um problema na adoção de soluções baseadas neste tipo de arquitetura.

2.2 Infra-estruturas para sistemas cientes do contexto

Os aspectos relacionados com a gerência de aplicações adaptativas, tipicamente cientes de contexto, como aquelas discutidas anteriormente, requerem uma infra-estrutura de suporte que integra (i) uma forma para especificar restrições de qualidade e políticas de adaptação; (ii) mecanismos para configurar, implantar e adaptar os componentes da aplicação e (iii) mecanismos para descobrir e monitorar componentes e recursos. Embora esta infra-estrutura não seja o foco deste trabalho, a discussão é benéfica. Infra-estruturas para gerenciar aplicações distribuídas usualmente oferecem suporte para lidar com aspectos não-funcionais dinâmicos, mas, em geral, a maioria não inclui o suporte para a descoberta de recursos ou trata este aspecto como um *hot spot* que deve ser programado manualmente. Assim, nesta seção discutimos algumas propostas recentes e como os serviços propostos neste trabalho poderiam ser integradas.

2.2.1 Context Managing Framework

Este projeto (Korpipää et al., 2003) é composto por quatro elementos principais: o gerente de contexto, os servidores de recursos, o serviço de reconhecimento de contexto e a aplicação propriamente dita. O gerente de contexto armazena as informações de contexto de forma centralizada. Os clientes e recursos interagem com ele podendo adicionar dados de contexto, inscrever-se para receber notificações num modelo baseado em eventos e podem requisitar diretamente informações de contexto.

Os servidores de recursos são responsáveis por coletar os dados brutos e postá-los para o gerente de recursos que pode efetuar ou não algum processamento sobre estes dados. Após efetuarem as medidas dos sensores, os servidores de recursos podem efetuar um pré-processamento com o objetivo de identificar ruído, por exemplo, que não deveria ser informado ao gerente de contexto.

O pré-processamento é baseado em um conjunto de medidas num determinado período de tempo onde se calcula o valor de uma propriedade para aquele intervalo de tempo. Este valor é processado em uma etapa chamada de extração de propriedades que calcula propriedades ainda mais próximas do que seria uma propriedade de contexto. Finalmente, os valores obtidos na fase anterior passam pela etapa de quantização e etiquetamento semântico onde os valores coletados são associados a grandezas do mundo real de acordo com uma ontologia pré-definida. Conjuntos *fuzzy* ou limites *crisp* são utilizados para quantizar os dados coletados e processados nas etapas anteriores. O contexto é modelado através de uma ontologia descrita em RDF (W3C3, 2008) com vocabulários pré-definidos como intensidade de som, intensidade, tipo e freqüência de luz, temperatura (frio, normal, quente). Este projeto também não permite que dados históricos de contexto sejam consultados.

O serviço de reconhecimento de contexto possui uma tabela interna que registra *plugins* de serviços de reconhecimento de contexto que permitem o compartilhamento de reconhecimento de contexto de alto nível. Estes *plugins* podem ser adicionados e removidos dinamicamente de acordo com as necessidades dos usuários.

2.2.2 SOCAM

SOCAM (*Service-oriented Context-Aware Middleware*) (Gu et al., 2004), é um

middleware para desenvolvimento de aplicações cientes do contexto que inclui um conjunto de serviços para executar a descoberta, aquisição e interpretação de dados de contexto.

A modelagem de contexto é baseada em ontologia descrita em OWL, sendo que o domínio de computação pervasiva é dividido em subdomínios para diminuir a complexidade da modelagem. Existe um elemento responsável por inferir sobre a base de conhecimento criada a partir dos dados de contexto de forma a resolver conflitos, mantendo a consistência da base de dados.

Seus serviços são construídos de forma aderente ao padrão OSGi (OSGi, 2009). OSGi é uma arquitetura aberta que define uma infra-estrutura padrão para a implantação, ativação, desativação, atualização, remoção de aplicações orientadas a serviço, independente de plataforma de hardware por ser baseada na plataforma Java, com vários níveis de segurança, capaz de hospedar múltiplos serviços de diferentes fornecedores e capaz de trabalhar com diferentes tecnologias de redes. SOCAM é construído a partir deste padrão com o intuito de potencialmente tomar vantagens das características de interoperabilidade disponibilizada por este padrão.

SOCAM é dividido em cinco componentes: (i) provedores de contexto, responsáveis por abstrair as diferentes fontes de dados contextuais convertendo-os para uma representação em OWL que pode ser compartilhada pelos demais componentes da infra-estrutura; (ii) um interpretador de contexto, provê a lógica necessária para processar as informações sobre o contexto a partir das ontologias em OWL, (iii) um banco de dados responsável por armazenar os dados sobre o contexto, (iv) um serviço de localização de serviços, onde os provedores e o interpretador de contexto se registram para que possam então ser localizados pelos usuários das aplicações e (v) as aplicações propriamente ditas.

2.2.3 CASS

CASS (*Context-awareness sub-structure*) (Fahy e Clarke, 2004) é uma infra-estrutura baseada em um servidor central que armazena os dados sobre o contexto. O elemento da infra-estrutura responsável por coletar estes dados é o *SensorListener*. Este elemento é informado quando ocorrem mudanças nos sensores do ambiente. Este informe é realizado pelo o que os autores chamam de nodos sensores (elementos responsáveis por obter o estado dos sensores, podendo existir mais de um nodo sensor).

O *Sensor Listener* armazena os dados recebidos em um banco de dados relacional e pode disponibilizar dados históricos sobre o contexto posteriormente. Além de dados sobre o contexto podem ser armazenados no banco de dados, domínios de conhecimento na forma de regras, comportamentos que sejam relevantes para a aplicação, ações a serem executadas ou informações anexadas a marcas, dados sobre entidades, aplicações e os contextos que sejam de seu interesse, perfis, fontes de dados de contexto, grupos e perfis de usuários.

O elemento da infra-estrutura responsável por retornar os dados de contexto *ContextRetriever* e o elemento responsável por notificar as mudanças no contexto aos nós remotos é chamado *ChangeListener*.

CASS possui um mecanismo de inferência que usa as regras armazenadas no banco de dados para resolver questões envolvidas com mudanças nos dados do contexto, como decidir se a mudança é significativa ou não para ser informada aos clientes da infra-estrutura.

Outra característica do projeto é que as regras podem ser atualizadas e modificadas pelos usuários da infra-estrutura sem a intervenção de programadores.

2.2.4 Q-Cad

O projeto Q-Cad (*Quality of Service and Context Aware Discovery Framework*) (Capra et al., 2005) é uma infra-estrutura para desenvolvimento de sistemas pervasivos que permite a descoberta e seleção de recursos que melhor satisfaçam os requisitos dos usuários destas aplicações. Para isso leva em conta o contexto em que tais aplicações se encontram juntamente com requisitos de qualidade de serviço. Uma vez selecionado o recurso, um processo de ligação é efetuado entre a aplicação e o recurso selecionado.

O processo de descoberta de recursos é baseado em perfis que a aplicação cria especificando as propriedades do contexto relevantes e o número de recursos retornado é resumido a um subconjunto de recursos plausíveis para o contexto em questão. Além disso, através de uma função de utilidade a aplicação especifica os níveis de qualidade de serviço desejados. A função de utilidade é aplicada ao subconjunto de recursos plausíveis retornado no processo de localização de recursos para aumentar a eficácia do processo de busca. As funções de utilidade podem ser utilizadas para minimizar ou maximizar determinadas propriedades dos recursos podendo ser trocadas dinamicamente assim como os perfis utilizados como critérios no processo de busca, sendo que ambos são expressos por meio de

XML.

A infra-estrutura diferencia os recursos remotos dos recursos locais. Os recursos remotos podem ser baixados e implantados no nó local, sendo que cada recurso deve estar registrado em um servidor de nomes central e identificado de forma única. O processo de ligação é feito através da identificação do recurso e um componente que deve estar presente na máquina onde a ligação é feita. A infra-estrutura trata de implantar o componente dinamicamente.

2.2.5 Gaia

O projeto Gaia (Roman et al., 2002; Gaia, 2009) é uma infra-estrutura baseada em espaços ativos. Um espaço ativo "é um espaço físico coordenado por uma infra-estrutura de software baseada em contexto que aperfeiçoa a habilidade de usuários móveis para interagir e configurar seus ambientes físicos e digitais".

Gaia estende os conceitos de um sistema operacional através da disponibilização de serviços básicos tais como eventos, presença de entidades, notificação de contexto e um sistema de nomes. Provedores de contexto tornam disponíveis informações de contexto sobre o contexto corrente através de um modelo de comunicação baseado em produtores, consumidores e canais. Cada canal possui um produtor e um ou mais consumidores. Através do serviço de contexto os clientes podem consultar os dados contextuais e efetuar o registro nos canais para que os mesmos possam ser informados de mudanças ocorridas no contexto. Há também um serviço de presença que é responsável por detectar entidades físicas (pessoas e dispositivos) e lógicas (serviços e aplicações) no espaço ativo denominado espaço virtual do usuário (*User Virtual Space*). Este espaço é composto pelo usuário e os dispositivos, dados, serviços e qualquer outra entidade associada ao usuário. Este espaço virtual é independente de dispositivo e está permanentemente associado ao usuário migrando com ele conforme o mesmo se desloca.

A infra-estrutura possui um sistema de arquivos de contexto que armazena automaticamente as informações sobre o contexto do usuário atrelado à posição atual do mesmo de forma hierárquica, similar a uma estrutura de diretórios que pode ser facilmente consultada. A modelagem de contexto é feita através de DAML+OIL (DAML, 2009) e é representado por um predicado quaternário na forma *Context(<ContextType>,<Subject>,<Relater>,<Object>)*, onde *ContextType* é o tipo do

contexto descrito pelo predicado, *Subject* representa um lugar, uma pessoa ou qualquer coisa relacionada ao contexto. *Object* é um valor associado ao *Subject* através do elemento *Relater* que é um operador de comparação, um verbo ou um predicado. As consultas são realizadas através do serviço de contexto utilizando-se este predicado. Através do serviço de contexto as aplicações se registram para serem informadas de mudanças no contexto.

2.2.6 Rainbow

Rainbow (Cheng et al., 2004; Cheng et al., 2002) é uma infra-estrutura baseada em arquitetura de software que descreve componentes, conectores e regras de composição que capturam os requisitos não funcionais dos sistemas. Os autores definiram um "estilo arquitetural" como um sistema de tipos mais um conjunto de regras e restrições. Este estilo arquitetural permite a especificação de elementos a serem monitorados bem como requisitos de qualidade de serviço de uma aplicação a serem mantidos através de estratégias de adaptação. O processo de monitoramento é gerenciado por *probes* associados às propriedades do modelo arquitetural da aplicação. Através do uso de invariantes que fazem parte do conjunto de restrições se especifica o nível de qualidade de serviço desejado, a infra-estrutura dispara as políticas de adaptação baseada nas regras quando as propriedades invariantes são desrespeitadas.

As adaptações são realizadas através do que os autores denominaram de operadores de adaptação. Estes elementos são conjuntos de operadores que podem ser definidos pelos projetistas das aplicações para determinar como a aplicação será adaptada. Um operador poderia adicionar um componente enquanto outro operador poderia mover um conjunto de componentes de um nodo para outro.

Os tipos deste "estilo arquitetural" são definidos em uma linguagem de definição de arquiteturas (ADL) genérica e as regras e restrições são definidas através de uma linguagem de lógica de predicados de primeira ordem similar à OCL da UML (UML, 2009). A comunicação é realizada através de RMI e XML.

2.2.7 JCAF

JCAF (*Java Context Awareness Framework*) (Bardram, 2005) é uma infra-estrutura cujos objetivos são prover uma infra-estrutura desenvolvida na linguagem Java para o

desenvolvimento de sistemas cientes do contexto que seja simples e robusto com um conjunto de interfaces simples e expressivo. Esta infra-estrutura é orientada a serviços, distribuída e baseada em eventos. A proposta do projeto é que a infra-estrutura seja utilizada de forma similar a outras APIs existentes no mercado como JDBC e ODBC. Estas APIs expõem um conjunto de interfaces padronizadas com a qual as aplicações interagem sendo possível a troca de forma bastante simples das classes que realizam estas interfaces, sem que modificações sejam feitas nas aplicações que as utilizam.

A infra-estrutura possui um serviço de contexto que pode ser distribuído ou não, sendo responsável por disponibilizar os dados sobre contexto (chamados de entidades) às aplicações que se inscrevem para serem notificadas de mudanças ocorridas no contexto. As entidades são programas escritos em Java que executam sobre o serviço de contexto e respondem quando mudanças em suas propriedades ocorrem em função de mudanças no contexto.

A modelagem do contexto é realizada através de pares chave/valor que podem ser agrupados em um relacionamento ternário composto por uma entidade, relacionamento e um item de contexto. Onde uma entidade pode ser uma pessoa ou uma aplicação, por exemplo, e o item de contexto é qualquer elemento pertencente ao ambiente de execução da aplicação que seja de interesse da mesma.

A infra-estrutura permite o desenvolvimento de novos serviços e mecanismos de sensoriamento (chamados de monitores de contexto) que podem ser adicionados e removidos sem que haja re-inicialização de quaisquer elementos presentes no ambiente. Além disso, disponibiliza na API mecanismos que permitem a modelagem de contexto genérica, permite a criação de pontos de extensão que definem a qualidade dos dados do contexto além de permitir a criação de atuadores (chamados de atividades na infra-estrutura). Toda a comunicação realizada entre os componentes da infra-estrutura é feita através de RMI.

2.2.8 Amigo

O projeto Amigo (Lacoste et al., 2007) foca no desenvolvimento de ambientes para casas inteligentes. O projeto possui o *Context Management Service* (CMS) que é uma infra-estrutura para gerenciamento de informações de contexto. O projeto lida com diferentes protocolos e padrões referentes a casas inteligentes. O CMS é responsável por coletar e agregar dados oriundos de diferentes dispositivos transformando-os em dados contextuais e disponibilizá-los para os clientes da infra-estrutura. As informações de contexto são descritas

através de RDF. Baseado nestes dados de contexto a infra-estrutura é capaz de conectar clientes às fontes de dados contextuais. Há um serviço de nomes centralizado onde os recursos registram suas características de forma aos clientes serem capazes de localizá-los para posterior utilização.

Os diferentes sensores dos recursos devem implementar a interface *ContextSource* que é a interface padrão que os recursos utilizam para se comunicar com o CMS e passam a ser chamados de *context sources* ou de *context producers*.

Finalmente, há o interpretador de contexto (*context interpreter*) que é um *context source* especial dedicado a agregar dados de contexto baseados em medidas físicas de forma a gerar novos dados de contexto.

2.2.9 MoCA

MoCA (*Mobile Collaboration Architecture*) (Sacramento et al., 2004) é uma arquitetura para o desenvolvimento de cientes do contexto voltada para dispositivos móveis. A infra-estrutura é composta por serviços responsáveis por monitorar e determinar o contexto de dispositivos móveis e por uma infra-estrutura através da qual é possível a construção de aplicações que consomem estas informações de contexto.

No dispositivo móvel existe um serviço denominado *Monitor* responsável por coletar as os dados de contexto do dispositivo e da rede e disponibilizá-los periodicamente para o CIS (*Context Information Service*). O CIS é o serviço responsável por coletar, armazenar e processar os dados de contexto dos dispositivos e disponibilizá-los para as aplicações que fazem uso do MoCA. As aplicações podem fazer consultas diretas ao CIS ou podem se registrar para receberem informações sobre contexto. Antes de enviar os dados de contexto o *Monitor* consulta o CS (*Configuration Service*). O CS é o elemento da infra-estrutura responsável por informar ao *Monitor* a qual CIS o mesmo se encontra associado. O elemento opcional denominado CoPS (*Context Privacy Service*) é responsável pelas políticas de segurança, controlando como, quando e quem possui acesso aos dados de contexto.

Outro serviço que faz parte da infra-estrutura é o LIS (*Location Inference Service*). Este elemento utiliza informações de contexto do CIS prover a localização aproximada dos dispositivos. Através deste serviço é possível a definição de regiões em um nível mais alto por parte das aplicações que utilizam o MoCA. Estas regiões podem ser hierarquizadas através do

uso do SRM (*Symbolic Region Manager*). Este serviço permite a criação e consulta destas regiões por parte das aplicações.

A infra-estrutura possui ainda o elemento *Discovery Service* (DS) através do qual as aplicações que fazem uso do MoCA se registram e poderão ser localizadas pelos clientes móveis da aplicação.

A modelagem de contexto é feita através de XML baseada numa abordagem centrada em tipos forte. No XML são descritas informações estruturais (atributos e seus tipos de dados), comportamentais (se um atributo possui um valor constante ou uma faixa de valores aceitável) e abstrações específicas do contexto.

2.2.10 Hydrogen

Hydrogen (Hofer et al. 2002), diferente dos projetos a serem abordados mais adiante, no *Hydrogen* não há um servidor central, pois esta infra-estrutura é totalmente centrada em ambientes móveis. Cada dispositivo contém todos os elementos da infra-estrutura que é formada por três camadas: camada da aplicação, camada de gerenciamento e camada de adaptação.

Uma característica deste projeto é que dados históricos do contexto não são disponíveis, devido ao projeto estar voltado para ambientes totalmente móveis sem um servidor central e com dispositivos de baixo poder de processamento e recursos de armazenamento escassos.

A camada de gerenciamento é composta por um servidor de contexto que é responsável por armazenar os dados de contexto e disponibilizá-los para o ambiente. O contexto é subdividido em contexto local e contexto remoto. O local é o contexto de execução do dispositivo e o contexto remoto é o contexto dos demais dispositivos presentes no ambiente em um dado momento.

Quando os dispositivos estão próximos uns dos outros, a camada de gerenciamento presente em cada um dos dispositivos compartilha seus contextos trocando-os entre si através de mensagens XML. A camada de aplicação é reage de acordo com as mudanças que ocorrem em ambos os tipos de contexto. A modelagem dos dados contextuais utilizada é baseada na abordagem orientada a objetos onde uma interface padronizada é utilizada para trocar dados entre os elementos da infra-estrutura. As classes têm que herdar da superclasse *ContextObject*

que provê todos os mecanismos padrões de troca de dados entre os elementos da infra-estrutura.

2.2.11 Resumo

A Tabela 1 apresenta um resumo das principais características dos projetos abordados ao longo da Seção 2.2. Nenhum deles possui a capacidade de federar seus serviços de forma nativa ficando a cargo dos projetistas de aplicações a construção de aplicações com esta característica. Além disso, todos eles são dependentes dos middlewares sobre os quais foram construídos, em sua maioria utilizando Java sem serem interoperáveis com diferentes serviços de contexto.

Tabela 1 - Comparação das propostas de infra-estrutura para contexto

	Provedor de Contexto	**Sensoriamento**	**Representação do Contexto**	**Notificação Assíncrona**	**Consulta Síncrona**	**Federação**
CMF	Sim	servidores de recursos	Ontologia (RDF)	sim	sim	não
SOCAM	Sim	provedores de contexto	Ontologia (OWL)	não	sim	não
CASS	Sim	nodos sensores	Banco de Dados Relacional	sim	sim	não
QCAD	Sim	recursos remotos/locais	Pares chave/valor	sim	sim	não
Gaia	Sim	provedores de contexto	predicados quaternários baseados em DAML + OIL)	sim	sim	não
Rainbow	Não	probes	ADL	Sim	não	não
JCAF	Sim	monitores de contexto	predicados ternários (entidade, relacionamento, item de contexto)	sim	não	não
Amigo	Sim	produtores de contexto	RDF	sim	sim	não
MoCA	Sim	monitores	baseado em modelo XML próprio	sim	sim	não
Hydrogen	n.a.	provedores de contexto	orientado a objetos	sim	não	não

CAPÍTULO 3 SERVIÇOS DE DESCOBERTA E CONTEXTO

Nossa proposta está incluída na linha de abordagem que considera que o projeto de uma aplicação ciente de contexto depende de serviços para localizar recursos de um determinado tipo, e que os recursos encontrados devem estar aderentes a certas restrições de contexto (características específicas ou estado de operação). Uma vez que um recurso é localizado e selecionado, a aplicação passa a ter uma referência para este recurso através da qual se torna possível interagir com a mesma.

Uma aplicação pode necessitar monitorar o estado de vários recursos de diferentes tipos para tomar alguma decisão a partir da coleção destas informações. Esta coleção de informações, tomada em conjunto, é o que denominamos de contexto. Dependendo da aplicação o contexto pode ser o considerado o conjunto de estados de todos os recursos monitorados com os quais a aplicação interage ou apenas um subconjunto. De posse do contexto atual a aplicação pode decidir, por exemplo, manter a configuração atual ou iniciar uma rotina de reconfiguração. Isto pode ocorrer com freqüência em alguns ambientes de execução. Uma seqüência de adaptação pode implicar na necessidade de se descobrir recursos com melhores níveis de qualidade em relação àqueles que estão sendo utilizados, ou até mesmo iniciar a descoberta de novos recursos com diferentes características.

Dados estas características, e com base no trabalho preliminar de (Cardoso, 2006a), propomos a utilização de um modelo para a descoberta de recursos e monitoração baseados em dados de contexto, centrados nos seguintes elementos: Agentes de Recursos (AR), Serviço de Registro e Diretórios (SRD), Serviço de Contexto (SC) e um Serviço de Descoberta (SD). Cada um destes elementos, assim como a modelagem dos dados contextuais, são descritos nas próximas subseções deste capítulo.

3.1 Modelo de contexto

Em nosso trabalho aderimos à definição apresentada por (Dey, 2000) onde se diz que qualquer informação relevante à execução das aplicações pode ser considerada como contexto. Sendo assim, se a temperatura do cômodo de uma residência, a freqüência de uma CPU, ou a localização de uma pessoa dentro de uma casa estiver relacionada com a aplicação, estas informações serão consideradas como parte do contexto da aplicação em questão. Com

isso em mente, adotamos um modelo que une características dos modelos de pares objeto/valor, dos modelos orientados a objetos e dos modelos esquemas de marcação que foram discutidos na Seção 1.1. A Figura 1 mostra como um recurso e seus atributos são modelados em nossa proposta.

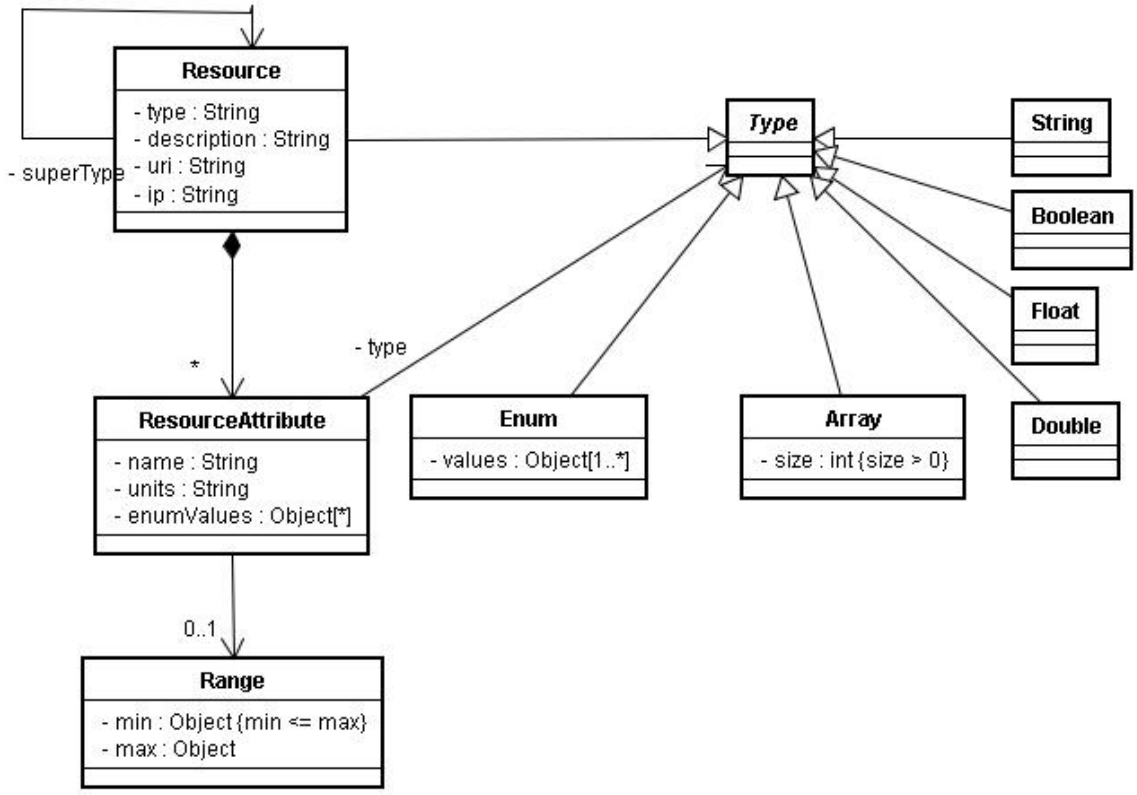


Figura 1 - Representação UML dos recursos

A classe abstrata *Type* representa os tipos possíveis das propriedades que um recurso pode ter. Em nosso modelo são definidos doze tipos primitivos (*boolean*, *byte*, *char*, *date*, *dateTime*, *double*, *enum*, *float*, *integer*, *long*, *short* e *string*). Por questão de simplicidade, na Figura 1 são apresentados apenas seis destes tipos Todos os tipos primitivos são classes da classe abstrata *Type*.

A classe *Resource* representa o recurso em si, e possui um relacionamento de composição com a classe *ResourceAttribute* que modela os atributos de um recurso. Um atributo possui um nome (propriedade *name*) e um tipo (relacionamento *type*). Estes dois atributos são obrigatórios. Além disso, um atributo pode ter opcionalmente associado a ele uma unidade de medida (de temperatura, pressão, etc.). Os valores de uma propriedade podem ser restringidos por meio de uma enumeração (representada pela propriedade *enumValues)* ou por meio de uma faixa de valores contíguos (aplicável apenas a atributos de tipos numéricos).

Esta faixa de valores é expressa através da classe *Range*, onde os atributos *min* e *max* indicam os valores inferior e superior da faixa respectivamente.

Como a classe *Resource* é uma subclasse da classe *Type*, uma propriedade pode ser de um tipo *Resource*, podendo-se expressar relacionamentos de composição típicos dos modelos baseados em objetos. Outra característica destes modelos é a herança que também é captada pela nossa abordagem. Na Figura 1 é mostrado que a classe *Resource* possui uma propriedade chamada *type*. Esta propriedade armazena o nome do tipo do recurso (que é uma subclasse da classe *Type*). Além disso, a classe *Resource* possui um auto-relacionamento chamado *supertype*. Este auto-relacionamento representa relações de herança. Ou seja, indica se um recurso possui ou não um supertipo.

O atributo *description* da classe *Resource* é opcional e deve ser utilizado para fornecer um texto legível por humanos. Desta forma, ferramentas administrativas podem utilizá-lo para dar aos usuários destas ferramentas informações sobre o recurso.

Cada recurso é identificado de forma única no ambiente por meio de um URI (*Universal Resource Identifier*) (W3C5, 2007). O atributo *uri* da classe *Resource* é utilizado para esta finalidade. Além do *uri* é armazenado também o endereço IP do recurso para permitir sua localização "física".

Podemos posicionar nosso modelo de contexto da seguinte forma:

- Cada propriedade primitiva de um recurso presente no contexto de execução de uma aplicação ciente do contexto pode ser encarada como um par chave/valor. No entanto, diferente dos modelos baseados em pares chave/valor nossa abordagem adiciona a característica de um sistema de tipos fortes que pode ser verificado pela infra-estrutura além de expressar restrições de valores e as unidades de medidas associadas aos valores armazenados;
- Nossa abordagem emprega os conceitos de herança e composição presentes nos modelos orientados a objetos, na medida em que possibilita expressar que um recurso possui um tipo (pertence a uma classe de recursos) e que seu tipo é um subtipo de outra classe de recursos. Pode-se representar assim um relacionamento de herança simples. Relacionamentos de heranças múltiplas não são contemplados;
- Além da herança, é possível também expressar relacionamentos de composição, ou seja, pode-se dizer que um determinado recurso é composto por suas propriedades e pelas propriedades de outros recursos previamente definidos;

- Finalmente, nossa abordagem emprega a linguagem de marcação XML para representar os recursos presentes no contexto assim como seus atributos e as relações de herança e composição.

O modelo de contexto adotado em nosso trabalho não inclui informações semânticas dos recursos representados, mas apenas a estrutura dos mesmos e seus relacionamentos. Ficando a cargo das aplicações o conhecimento do quê trata cada propriedade de cada recurso.

3.2 Representação das propriedades do contexto

Cada classe de recursos a serem descobertos e monitorados deverá ser descrita segundo um padrão de formato que permita às aplicações localizar os recursos que melhor atendam às suas necessidades de qualidade de serviço. Em nosso trabalho as classes de recursos são descritas em linguagem XML (W3C2, 2008) que é uma linguagem neutra com relação à implementação e que permite a criação de outras linguagens a partir dela. A Listagem 1 apresenta o esquema XML utilizado para descrever qualquer classe de recurso.

O elemento *tType* descreve os possíveis tipos primitivos dos valores das propriedades dos recursos. Existem doze tipos possíveis tais como *string* (linha 6), *boolean* (linha 7), *float* (linha 8), dentre outros.

O elemento *Resource* presente na linha 20 indica que se trata da descrição de um recurso. Seu tipo é especificado na linha 22 e seu supertipo pode ser especificado através do elemento *SuperType* (linha 23). Desta forma é possível expressar relacionamentos de herança existentes entre dois ou mais recursos. Não há limite quanto ao nível do relacionamento podendo ter hierarquias com um número qualquer de profundidade.

A classe (ou tipo) de recurso pode ser descrita através do elemento opcional *Description* presente na linha 24. Este elemento pode ser útil em ferramentas administrativas que apresentam os tipos de recursos que podem estar presentes no ambiente de execução. Além disso, mecanismos de busca baseados em palavras-chave nos mesmos moldes do Google (Google, 2008) podem fazer uso deste tipo de dado.

A partir da linha 25 os atributos do recurso são descritos. Cada atributo (elemento presente na linha 28) pode ter uma descrição (elemento *Description,* cuja utilização é análoga ao elemento de mesmo nome que descreve o recurso), possui obrigatoriamente um nome (elemento *Name* na linha 39), um tipo (elemento *Type* na linha 40), pode ter seus valores

limitados aos valores de uma enumeração (linha 41) e pode ter associado a si uma unidade de medida (elemento *Units* na linha 42). Cabe destacar que o tipo de um atributo pode ser qualquer um dos tipos previstos pelo elemento *tType* discutido anteriormente ou pode ser do tipo de um recurso previamente descrito. Podemos dizer que o tipo de um atributo é do tipo de outro recurso. Expressando-se assim o relacionamento de composição (ver Sseção 3.1.

```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 3      attributeFormDefault="unqualified" elementFormDefault="qualified">
 4 <xsd:simpleType name="tType">
 5   <xsd:restriction>
 6     <xsd:enumeration value="string" />
 7     <xsd:enumeration value="boolean" />
 8     <xsd:enumeration value="float" />
 9     <xsd:enumeration value="double" />
10     <xsd:enumeration value="byte" />
11     <xsd:enumeration value="short" />
12     <xsd:enumeration value="integer" />
13     <xsd:enumeration value="long" />
14     <xsd:enumeration value="date" />
15     <xsd:enumeration value="dateTime" />
16     <xsd:enumeration value="enum" />
17     <xsd:enumeration value="char" />
18   </xsd:restriction>
19 </xsd:simpleType>
20 <xsd:element name="Resource">
21   <xsd:sequence>
22     <xsd:element name="Type" type="xsd:string" />
23     <xsd:element name="SuperType" type="xsd:string" />
24     <xsd:element name="Description" type="xsd:string" minOccurs="0" />
25     <xsd:element name="Attributes" minOccurs="0">
26       <xsd:complexType>
27         <xsd:sequence>
28           <xsd:element name="Attribute" minOccurs="1" maxOccurs="unbounded">
29             <xsd:complexType>
30               <xsd:sequence>
31                 <xsd:element name="Range" minOccurs="0" maxOccurs="1">
32                   <xsd:complexType>
33                     <xsd:attribute name="min" use="required" />
34                     <xsd:attribute name="max" use="required" />
35                   </xsd:complexType>
36                 </xsd:element>
37               </xsd:sequence>
38               <xsd:attribute name="Description" type="xsd:string" />
39               <xsd:attribute name="Name" type="xsd:NCName" use="required" />
40               <xsd:attribute name="Type" type="tType" use="required"/>
41               <xsd:attribute name="EnumValues" type="xsd:string" />
42               <xsd:attribute name="Units" type="xsd:string" />
43             </xsd:complexType>
44           </xsd:element>
45         </xsd:sequence>
46       </xsd:complexType>
47     </xsd:element>
48   </xsd:sequence>
49 </xsd:element>
```

Listagem 1 - XSD para descrição de recursos

A Listagem 2 apresenta um exemplo de descrição de uma classe de recursos do tipo *SimpleProcessing* (linha 3). Esta classe de recursos encapsula os atributos de recursos de processamento sem capacidade de armazenamento permanente.

```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <Resource>
 3   <Type>SimpleProcessing</Type>
 4   <Description>Simple processing resource without storage</Description>
 5   <Attributes>
 6     <Attribute Name="CPUClock" Type="float" Units="MHz" />
 7     <Attribute Name="TotalMemory" Type="float" Units="MB"/>
 8     <Attribute Name="FreeMemory" Type="float" Units="MB"/>
 9     <Attribute Name="CPUIdle" Type="float" Units="%"
10                Description="% of cpu time available">
11       <Range min="0" max="100"/>
12     </Attribute>
13   </Attributes>
14 </Resource>
```

Listagem 2 - Descrição da classe de recursos *SimpleProcessing*

O elemento *Description* (linha 4) contém um texto legível por humanos indicando que se trata de um recurso de processamento. Há quatro atributos: *CPUClock* (linha 6), do tipo *float* expresso em MHz; *TotalMemory* (linha 7) e *FreeMemory* (linha 8) do tipo *float* e expressos em MB; *CPUIdle* (linhas 9 e 10), do tipo *float* e expresso em porcentagem.

O atributo *CPUIdle* possui ainda uma descrição legível por humanos (atributo *Description* presente na linha 10*)* além de ter seus valores limitados ao intervalo fechado entre zero e 100 pelo elemento *Range* (linha 11).

A Listagem 3 apresenta a descrição de outro recurso denominado *StorableProcessing*. O tipo *SimpleProcessing* é o supertipo deste recurso (linha 4). Ou seja, o tipo *StorableProcessing* herda todas as propriedades de seu supertipo: *CPUClock*, *TotalMemory*, *FreeMemory* e *CPUIdle*.

```
1 <?xml version="1.0" encoding="UTF-8"?>
 2 <Resource>
 3   <Type>StorableProcessing</Type>
 4   <SuperType>SimpleProcessing</SuperType>
 5   <Description>Processing resource with storage</Description>
 6   <Attributes>
 7     <Attribute Name="TotalDisk" Type="float" Units="MB" />
 8     <Attribute Name="DiskFree" Type="float" Units="MB"/>
 9     <Attribute Name="OSName" Type="string"
10                EnumValues="Linux|Windows|Solaris|Mac|OS2|Symbian">
11   </Attributes>
11 </Resource>
```

Listagem 3 - Descrição da classe de recursos *StorableProcessing*

Além das propriedades herdadas de *SimpleProcessing*, *StorableProcessing* define mais três propriedades: *TotalDisk* (linha 7) e *DiskFree* (linha 8) ambas do tipo *float* com unidade de medida em MB e a propriedade *OSName* cujos valores estão restritos às seqüências de caracteres Linux, Windows, Solaris, Max, OS2 e Symbian (linhas 9 e 10).

Por razões de consistências é importante que as descrições dos tipos de recursos possam ser registradas em um repositório central a partir do qual as mesmas possam ser recuperadas para posterior utilização, funcionando como um *template* e como um contrato que todos os recursos de um determinado tipo previamente registrado deverão seguir. Para tanto, faz-se necessária a existência de um Serviço de Registro e Diretório (SRD) que servirá como ponto inicial a partir do qual os recursos e componentes da infra-estrutura de suporte poderão ser localizados. Este serviço será discutido na Seção 3.3.2.

3.3 Arquitetura do f*ramework*

Nosso trabalho propõe uma arquitetura centrada em quatro componentes responsáveis por disponibilizar a infra-estrutura necessária para o monitoramento de aplicações cientes do contexto. Os serviços comunicam-se utilizando um protocolo baseado em XML independente de plataforma que pode ser implementado em qualquer linguagem de programação. Os componentes principais da infra-estrutura são: os Agentes de Recursos (AR), o Serviço de Registro e Diretório (SRD), o Serviço de Contexto (SC) e o Serviço de Descoberta (SD).

A Figura 2 ilustra os componentes do *Framework* apresentando os relacionamentos existentes entre eles.

Os Agentes de Recursos (ARs) representam os recursos disponíveis para as aplicações cientes do contexto. São os elementos responsáveis por coletar as diversas medidas dos sensores presentes no ambiente de execução das aplicações. Todo AR tem que efetuar seu registro no Serviço de Registro e Diretório (SRD) descrevendo suas características e dados relativos à sua localização. Os clientes da infra-estrutura podem ter acesso aos dados de contexto disponibilizados pelos ARs através do Serviço de Descoberta (SD) ou através do Serviço de Contexto (SC).

Quando os clientes não conhecem a localização de um AR eles podem efetuar consultas ao SD baseadas nos tipos e atributos dos recursos. O SD consulta o SRD e obtém uma relação de recursos que respeitam os critérios passados. Em seguida o SD determina

quais propriedades dos recursos identificados sofrem variações de acordo com o contexto e efetua uma consulta ao SC para obter os valores atuais destas propriedades. De posse destes valores, o SD é capaz de determinar quais recursos estão de acordo com os critérios definidos pelos clientes e retorna uma relação destes recursos.

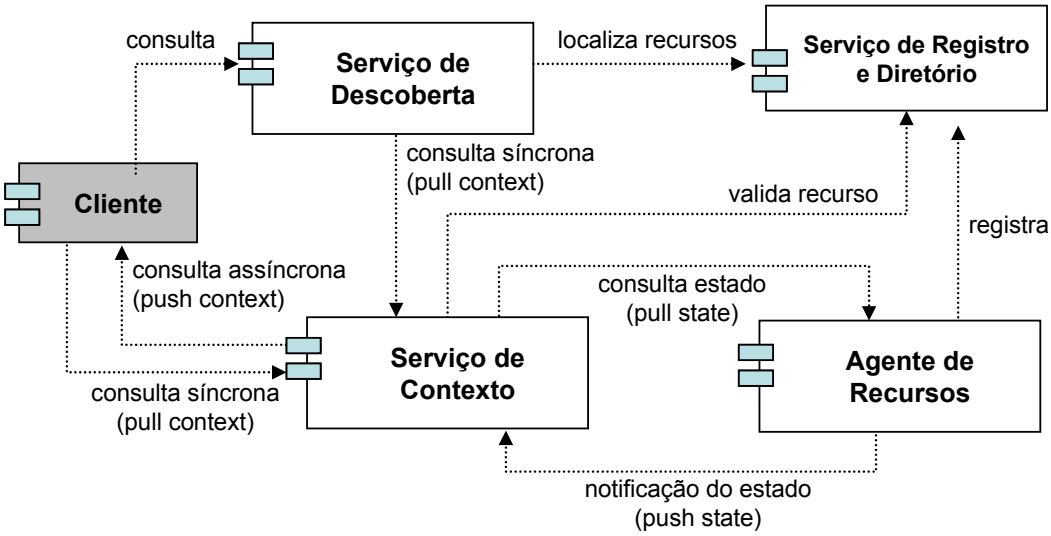


Figura 2 - Componentes da infra-estrutura

Caso os clientes conheçam as localizações dos recursos que desejam utilizar, podem fazer consultas diretamente ao SC para obterem os valores das propriedades que sofrem variações ao longo do tempo.

Estas consultas podem síncronas, ou seja, os clientes fazem as consultas e esperam que o SC retorne com os valores desejados (*pull context*), ou podem ser assíncronas (*push context*). Nesta última forma, é utilizado um modelo baseado em eventos, onde o cliente efetua seu registro junto ao SC indicando o tipo ou o AR no qual está interessado e o SC informa quando houver mudanças nas propriedades destes ARs. O SC pode efetuar uma consulta síncrona aos ARs (*pull state*) ou pode ser informado pelos ARs de forma assíncrona via modelo de notificação baseado em eventos (*push state*), onde os ARs registrados no SC informam o SC sempre que alguma mudança ocorre em seus estados internos.

É importante mencionar que as interações possíveis entre um cliente e os serviços propostos bem como as interações entre estes serviços e os ARs são definidas por protocolos e mensagens que são trocados e descritos usando uma representação em XML independente de *middleware*. Esta abordagem desacopla os elementos que compõem a arquitetura de mecanismos de comunicação específicos, sendo possíveis implementações que podem utilizar Serviços Web, RMI ou mesmo soquetes TCP/IP. Em nossa implementação de referência foram utilizados Serviços Web (Capítulo 4).

Nas seções seguintes serão abordados com maiores detalhes os serviços e o modelo que compõem o *framework*.

3.3.1 Agentes de Recursos (AR)

Os Agentes de Recursos são os elementos da infra-estrutura responsáveis por coletar os dados de contexto dos diversos sensores presentes no ambiente no qual as aplicações sensíveis ao contexto encontram-se presentes. Eles atuam como intermediários entre os sensores e as aplicações cientes do contexto tornando as informações de contexto disponíveis, além de responder às consultas de monitoramento realizadas sobre os recursos, escondendo detalhes de baixo nível utilizados no sensoriamento e aquisição de dados brutos. Para que isso seja possível, é necessária a existência uma interface uniforme para que a aplicação não tenha que estar ciente dos detalhes específicos de cada tipo de mecanismo de sensoriamento. Isto está de acordo com a abordagem centrada em middleware que foi apresentada na Seção 1.2.

Cada AR possui associado a ele uma descrição em XML que deverá ser registrada em um Serviço de Registro e Diretório (SRD) discutido logo a seguir. Esta descrição está associada à descrição do tipo de recursos discutida na Seção 3.2. Ou seja, as descrições da Seção 3.2 descrevem os tipos de recursos que poderão estar presentes no contexto de execução das aplicações enquanto que as descrições de recursos representam descrições de diferentes instâncias destes tipos. Cada tipo de recurso tem o suporte de um ou mais ARs correspondentes que, como mencionado, é a entidade da infra-estrutura responsável por prover os dados de contexto do recurso que ele representa.

A Listagem 4 apresenta o XSD utilizado para registrar ARs no Serviço de Registro e Diretório.

Na linha 4 está presente o elemento *ResourceRegister* indicando que se trata de uma mensagem de registro de AR. Na linha 6 o elemento *Type* informa o tipo do recurso. Este tipo deve estar previamente registrado no RDS. Através desta informação, o SRD é capaz de validar as propriedades do AR com as propriedades do tipo, garantindo a consistência dos dados informados (tipo correto de dado, limites inferior e superior válidos, etc.).

O elemento opcional *Description* (linha 7) é utilizado para fornecer um texto livre legível por humanos cujo objetivo é descrever o AR. A linha 8 contém o elemento *Attributes*

que contém a relação de atributos do AR. As linhas 13 a 16 contêm o nome (elemento *Name*) e o valor (elemento *Value*) de cada atributo. Todo atributo presente na descrição do tipo do recurso tem que ser declarado na descrição do AR, com os valores consistentes com os tipos e restrições dos atributos previamente declarados.

```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 3   attributeFormDefault="unqualified" elementFormDefault="qualified">
 4   <xsd:element name="ResourceRegister"/>
 5     <xsd:sequence>
 6       <xsd:element name="Type" type="xsd:string"/>
 7       <xsd:element name="Description" type="xsd:string" minOccurs="0"/>
 8       <xsd:element name="Attributes" minOccurs="0">
 9         <xsd:complexType>
10           <xsd:sequence>
11             <xsd:element name="Attribute" maxOccurs="unbounded">
12               <xsd:complexType>
13                 <xsd:attribute name="Name"
14                                type="xsd:NCName" use="required"/>
15                 <xsd:attribute name="Value" type="xsd:anySimpleType"
16                                use="required"/>
17               </xsd:complexType>
18             </xsd:element>
19           </xsd:sequence>
20         </complexType>
21       </xsd:element>
22       <xsd:element name="URI" type="xsd:anyURI" />
23       <xsd:element name="IP" type="xsd:string" />
24       <xsd:element name="CollectInterval" type="xsd:long" />
25       <xsd:element name="Tecnologies">
26         <xsd:complexType>
27           <xsd:sequence>
28             <xsd:element name="Tecnology" maxOccurs="unbounded">
29               <xsd:complexType>
30                 <xsd:attribute name="Type" type="xsd:string"
31                                use="required" />
32                 <xsd:attribute name="URL" type="xsd:anyURI"
33                                use="required" />
34               </xsd:complexType>
35             </xsd:element>
36           </xsd:sequence>
37         </xsd:complexType>
38       </xsd:element>
39     </xsd:sequence>
40   </xsd:element>
41 </xsd:schema>
```

Listagem 4 - XSD para registro de Agentes de Recursos no SRD

A linha 22 contém o elemento *URI* que é o identificador único do AR em todo o ambiente de execução da aplicação. É de responsabilidade da pessoa que implanta o AR especificar a URI. Caso já exista um AR registrado com este URI no SRD o mesmo não efetuará o registro do AR. A linha 23 indica a localização do AR (elemento *IP*).

O intervalo entre cada leitura dos sensores é informada através do elemento

CollectInterval (linha 24). Este dado é importante, pois dependendo da natureza da aplicação, a mesma pode estar interessada em ARs cujos valores desta propriedade estejam dentro de intervalos pré-definidos.

O elemento *Tecnologies* (linha 25) contém uma relação de informações sobre as diferentes tecnologias de comunicação com as quais o AR é capaz de lidar. O atributo *Type* (linha 30) contém um nome indicativo do tipo de tecnologia com a qual o AR é capaz de se comunicar (serviço web ou RMI, por exemplo), enquanto que o atributo *URL* informa a localização do serviço com a indicação do protocolo com o qual o AR é capaz de lidar.

A Listagem 5 apresenta a mensagem XML enviada por um AR do mesmo tipo que o apresentado na Listagem 3.

```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <ResourceRegister>
 3   <Type>StorableProcessing</Type>
 4   <Description>Processing AR with storage at leme machine</Description>
 5   <Attributes>
 6     <Attribute Name="CPUClock" Value="Dynamic" />
 7     <Attribute Name="TotalMemory" Value="2048" />
 8     <Attribute Name="FreeMemory" Value="Dynamic" />
 9     <Attribute Name="CPUIdle" Value="Dynamic" />
10     <Attribute Name="TotalDisk" Value="60000" />
11     <Attribute Name="DiskFree" Value="Dynamic" />
12     <Attribute Name="OSName" Value="Linux" />
13   </Attributes>
14   <URI>leme.pel.uerj.br/ProcessingAgent</URI>
15   <IP>152.92.155.196</IP>
16   <CollectInterval>5</CollectInterval>
17   <Tecnologies>
18     <Tecnology Type="webservice"
19                URL="http://leme.pel.uerj.br:1978/ProcessingAgent?wsdl"/>
20     <Tecnology Type="rmi"
21                URL="rmi://leme.pel.uerj.br/ProcessingAgent"/>
22   </Tecnologies>
23 </Resource>
```

Listagem 5 - Mensagem para registro de um Agente de Recurso

Nesta mensagem está contido na linha 3 o tipo do Agente de Recurso (*StorableProcessing*) e seus atributos (linhas 6 a 12). As propriedades *CPUClock*, *TotalMemory*, *FreeMemory* e *CPUIdle* não estão presentes na Listagem 3, mas pertencem ao tipo *SimpleProcessing* (Listagem 2). Como o tipo *SimpleProcessing* é o supertipo de *StorableProcessing* este último herda automaticamente suas propriedades.

Os valores das propriedades *CPUClock* (linha 6), *FreeMemory* (linha 8), *CPUIdle* (linha 9) e *DiskFree* (linha 11) são declarados como dinâmicos (*Dynamic*). Esta informação indica que os valores destas propriedades podem sofrer variações ao longo do tempo, ou seja,

seus valores variam de acordo com o contexto de execução no qual a aplicação se encontra e serão disponibilizadas através do Serviço de Contexto (SC) a ser discutido mais tarde nesta seção.

O AR sendo registrado é identificado de forma única através de seu URI (linha 14). Esta identificação nada tem a ver com a tecnologia com a qual o AR foi construído ou com a(s) tecnologia(s) de comunicação que o AR utiliza para se comunicar. Ele tem que ser único em todo o domínio no qual o RDS encontra-se inserido.

Nas linhas 18 e 19 é informado que o AR é capaz de se comunicar através de um serviço web e que a definição do serviço encontra-se disponível em *http://leme.pel.uerj.br:1978/ProcessingAgent?wsdl*. Além de se comunicar através de serviço web, o AR é capaz de se comunicar através de RMI (RMI, 2008) (linha 20) e que o *proxy* para o objeto Java que representa o Agente de Recursos pode ser localizado através da URL *rmi://leme.pel.uerj.br/ProcessingAgent*.

3.3.2 Serviço de Registro e Diretório (SRD)

Para que os recursos possam ser localizados é necessário que haja um componente que possua uma interface bem definida através da qual os recursos possam registrar sua localização e suas características. Uma vez efetuado o registro dos recursos os clientes podem realizar consultas e localizar os recursos na infra-estrutura. O Serviço de Registro e Diretório (SRD) é o elemento da nossa proposta que permite a realização destas tarefas.

As propriedades dos recursos e suas localizações são armazenadas em um repositório de meta-nível que fica disponível para consultas através da interface padronizada do SRD. Estas consultas são efetuadas sobre as propriedades estáticas dos recursos. Esta característica facilita o trabalho do Serviço de Contexto e do Serviço de Descoberta que precisam lidar apenas com as propriedades dinâmicas dos recursos (que sofrem variações de acordo com o ambiente operacional em que os recursos se encontram).

Para que um recurso possa ser registrado no SRD é necessário que seu tipo esteja previamente registrado no SRD. A descrição do tipo de um recurso é importante por questões de segurança e de escopo de atuação. Por exemplo: um administrador de sistema pode querer permitir que apenas recursos de um determinado tipo sejam registrados em um determinado SRD.

Pode-se considerar o SRD como o ponto inicial a partir do qual todos os outros serviços da infra-estrutura podem ser localizados, uma vez que o Serviço de Contexto e o Serviço de Descoberta também se registram junto ao SRD e podem ser considerados como recursos "especiais" da infra-estrutura. Na verdade, nada impede que estes serviços possuam Agentes de Recursos que informem, por exemplo, quantas consultas foram feitas em um determinado período de tempo ou quaisquer outros dados que o projetista destes serviços acharem necessários. Outro motivo para que o SC e do SD sejam registrados junto ao SRD é que pode haver mais de um destes serviços disponíveis simultaneamente. Desta forma uma aplicação pervasiva poderia fazer uso de alguma heurística para selecionar um ou mais destes serviços baseado em suas características.

Cada SRD define um domínio no qual existem vários ARs registrados e pode haver dois ou mais SCs e SDs. Cada SRD pode ser ligado a outros SRDs formando uma federação aumentando ainda mais o escopo das consultas que podem ser realizadas. No entanto, o fato de um SRD estar federado a um ou mais SRDs é transparente para os clientes da infra-estrutura. O administrador da mesma é responsável por definir se um SRD estará ou não federado a outro SRD.

Uma vez que o SRD é o ponto inicial de toda a infra-estrutura é necessário que haja alguma forma padrão do mesmo ser localizado pelos demais elementos. Uma forma possível seria convencionar o envio de uma mensagem de descoberta em um canal de comunicação *broadcast* através do qual os clientes poderiam enviar uma mensagem e o SRD responderia enviando sua localização. Outra forma seria o SRD enviar periodicamente uma mensagem em um canal *multicast* informando sua localização. Em nossa proposta, a forma de localizar o SRD é deixada a cargo do projetista dos serviços. Na implementação padrão é possível fazer uso de *broadcast* ou especificar o endereço IP e a porta do SRD diretamente.

Finalmente, cabe destacar que uma vez que toda a troca de mensagens é efetuada em formato XML, é possível implementar o SRD de diferentes formas. Uma forma possível é utilizar um banco de dados relacional para armazenar as propriedades dos recursos. Outra abordagem é fazer uso de um servidor LDAP (Wahl, 1997) para persistir as propriedades dos recursos. A camada de software que atende as pesquisas feitas ao SRD serviria como um *proxy* para o banco de dados relacional ou o servidor LDAP.

Na Listagem 4, Seção 3.3.1, foi apresentado o esquema XML utilizado no registro de ARs junto ao SRD. A Listagem 6 apresenta o esquema XML da operação de remoção

explícita de um AR.

```
1  <xsd:element name="ResourceRemove">
2    <xsd:complexType>
3      <xsd:attribute name="URI" type="xsd:string" use="required"/>
4    </xsd:complexType>
5  </xsd:element>
```

Listagem 6 - XSD para remoção de um AR

A linha 1 indica a operação a ser realizada (remoção de um recurso). A linha 3 indica que o elemento *ResourceRemove* possui uma propriedade chamada URI. Esta propriedade deverá conter o identificador único do recurso que se deseja remover. A Listagem 7 é um exemplo da remoção do recurso registrado previamente como apresentado na Listagem 5.

```
<ResourceRemove URI="leme.pel.uerj.br/ProcessingAgent"/>
```

Listagem 7 - Remoção de um Agente de Recurso

A decisão de remover um AR de um SDR pode ser tomada por um administrador do sistema, pelo próprio AR durante seu processo de finalização ou por outros elementos da infra-estrutura. Por exemplo, o projetista do Serviço de Contexto pode decidir remover um AR que apresenta problemas de comunicação por um determinado intervalo de tempo.

Além de o AR poder ser removido de forma explícita, um AR pode ser removido de forma implícita caso o AR não efetue seu registro em intervalos de tempo pré-determinados. Na implementação padrão o AR deverá se registrar a cada 10 minutos. O objetivo desta abordagem é garantir que um AR será removido caso apresente alguma falha inesperada, como término repentino ou algum problema na infra-estrutura da rede.

A mensagem de consulta aos tipos registrados é feita de acordo com o XSD apresentado na Listagem 8. A linha 1 representa a operação a ser executada (*DirectoryQuery*). O elemento *all* do XML esquema indica que qualquer um dos elementos que a sucedem pode aparecer na representação XML. Caso seja necessário consultar os atributos de um AR cujo URI seja conhecido, pode ser utilizado o elemento URI presente na linha 4. O elemento *Type* (linha 5) pode ser utilizado quando se deseja localizar todos os ARs de um determinado tipo (definido pelo atributo *Value* na linha 7). Se o atributo *Strict* (linha 9) for definido com o valor *false*, a consulta levará em conta também todos os subtipos do tipo especificado. Caso seja necessário localizar todos os ARs, independente do tipo, o conteúdo do atributo *Value* deverá ser igual ao caractere "*".

```
 1 <xsd:element name="DirectoryQuery">
 2   <xsd:complexType>
 3     <xsd:all>
 4       <xsd:element name="URI" type="xsd:anyURI" minOccurs="0" />
 5       <xsd:element name="Type" minOccurs="0">
 6         <xsd:complexType>
 7           <xsd:attribute name="Value" type="xsd:string"
 8                          use="required" />
 9           <xsd:attribute name="Strict" type="xsd:boolean"
10                          default="true" />
11         </xsd:complexType>
12       </xsd:element>
13       <xsd:element name="SuperType" type="xsd:string" minOccurs="0" />
14       <xsd:element name="IP" type="xsd:string" minOccurs="0" />
15       <xsd:element name="IncludeFederation" type="xsd:boolean"
16                    minOccurs="0" default=”false”/>
17     </xsd:all>
18   </xsd:complexType>
19  </xsd:element>
20 </xsd:element>
```

Listagem 8 - XSD das consultas realizadas ao SRD

Pode-se ainda efetuar consultas baseadas no supertipo (linha 13) ou no IP (linha 14). Todas as consultas podem especificar se serão propagadas ou não para os SRDs federados ou não através do elemento *IncludeFederation* presente na linha 15.

A Listagem 9 apresenta um exemplo de consulta sendo feita baseada no tipo de recurso *SimpleProcessing* (linha 2) e que será propagado aos demais SRD da federação se o SRD sendo consultado estiver federado a um ou mais SRDs.

```
1  <DirectoryQuery>
2    <Type Value=”SimpleProcessing” Strict=”false”/>
3    <IncludeFederation>true</IncludeFederation>
4  <DirectoryQuery>
```

Listagem 9 - Exemplo de consulta ao SRD

3.3.3 Serviço de Contexto (SC)

O CS é responsável por prover informação de contexto por intermédio de uma API de alto nível e por esconder os detalhes de baixo nível relacionados com a comunicação necessária com os ARs presentes no ambiente de execução das aplicações cientes do contexto. A aplicação cliente tem somente que se preocupar com os dados que necessita e não como irá obtê-los.

O SC expõe duas APIs padronizadas que podem ser utilizadas pelos clientes da infra-estrutura, a saber: uma baseada em mecanismo *pull* e outro em mecanismo *push.* Na primeira API, o cliente de forma pró-ativa consulta o CS e bloqueia até que a resposta seja retornada

(consulta síncrona). A Figura 3 ilustra este tipo de consulta.

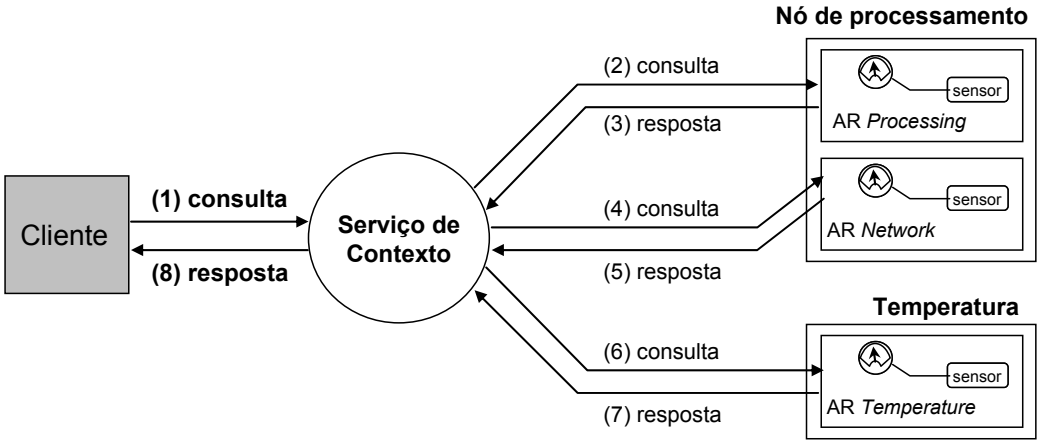


Figura 3 - Arquitetura do Serviço de Contexto

A consulta pode conter um ou mais recursos-alvo. Assim que o CS recebe uma consulta, as propriedades de cada recurso são verificadas e então o CS se comunica com cada AR envolvido (representados pelos ARs de processamento, rede e temperatura) para obter as informações de contexto individuais. Após coletar e consolidar todas as informações necessárias, o CS retorna o resultado para o cliente.

No segundo caso, o cliente comporta-se de forma reativa. A consulta retorna resultados parciais assim que os dados tornam-se disponíveis. Neste caso, o cliente registra o interesse em receber notificações sobre mudanças ocorridas no estado de um AR ou dos ARs de um determinado tipo. Se alguma mudança ocorrer nestes ARs, o cliente é notificado pelo CS. Ambos os mecanismos serão descritos a seguir.

3.3.3.1 Consultas ao contexto síncronas (Método *Pull*)

Consultas síncronas ao CS podem ser usadas quando o cliente necessita informações atualizadas (sem levar em consideração se alguma mudança ocorreu ou não desde a última medida) e sincronizadas de um mais ARs. A Listagem 10 apresenta o XSD que descreve as consultas síncronas que podem ser realizadas ao Serviço de Contexto.

```
1 <xsd:element name="ContextQuery">
2   <xsd:complexType>
3     <xsd:sequence>
4       <xsd:element name="Target" maxOccurs="unbounded">
5         <xsd:complexType>
6           <xsd:sequence>
7             <xsd:element name="Attributes" minOccurs="0"
8                          maxOccurs="unbounded">
```

```
 9               <xsd:complexType>
10                 <xsd:sequence>
11                   <xsd:element name="Attribute" maxOccurs="unbounded">
12                     <xsd:complexType>
13                       <xsd:attribute name="Name" use="required"
14                                      type="xsd:NCName" />
15                       <xsd:attribute name="op" use="optional"
16                                      type=”string”/>
17                       <xsd:attribute name="Value"
18                                      type="xsd:string" use="optional" />
19                     </xsd:complexType>
20                   </xsd:element>
21                   <xsd:element name="CollectTime" minOccurs="0">
22                     <xsd:complexType>
23                       <xsd:attribute name="Min" use="required" />
24                     </xsd:complexType>
25                   </xsd:element>
26                   <xsd:sequence minOccurs="0">
27                     <xsd:element name="CollectInterval" minOccurs="0"
28                                  maxOccurs="1">
29                       <xsd:complexType>
30                         <xsd:attribute name="Min" use="required"
31                                        type="xsd:integer" />
32                         <xsd:attribute name="Units" use="required"
33                                        type="xsd:string" />
34                       </xsd:complexType>
35                     </xsd:element>
36                     <xsd:element name="Results" minOccurs="0"
37                                  maxOccurs="1">
38                       <xsd:complexType>
39                         <xsd:attribute name="Max" use="required"
40                                        type="xsd:integer" />
41                       </xsd:complexType>
42                     </xsd:element>
43                   </xsd:sequence>
44                 </xsd:sequence>
45                 <xsd:attribute name="From" use="required"
46                                type="xsd:string" />
47               </xsd:complexType>
48             </xsd:element>
49           </xsd:sequence>
50           <xsd:attribute name="URI" use="required" type="xsd:anyURI" />
51         </xsd:complexType>
52       </xsd:element>
53     </xsd:sequence>
54   </xsd:complexType>
55 </xsd:element>
```

Listagem 10 - XSD das consultas síncronas ao Serviço de Contexto

O elemento *ContextQuery* (linha 1) indica que a operação a ser realizada é a de consulta síncrona. O elemento *Target* (linha 5) pode ocorrer inúmeras vezes na representação XML e indica através de seu atributo obrigatório *URI*, o AR objeto da consulta. O elemento *Attributes* (linhas 7 e 8) pode ocorrer zero ou mais vezes e encapsula os atributos de um AR que serão retornados pela consulta e que poderão ser utilizados como filtros na consulta. Caso o elemento *Attributes* não seja especificado todos os atributos do AR identificado pelo *URI*

serão retornados. Caso o elemento *Attributes* for especificado, ele deverá conter obrigatoriamente pelo menos um elemento *Attribute* (linha 11) que corresponde a uma propriedade do AR especificado e cujo nome corresponde ao valor do atributo *Name* (linhas 13 e 14). O elemento *op* (linhas 15 e 16) corresponde a um operador lógico que poderá ser utilizado na consulta. Caso seja utilizado, o atributo *Value* (linhas 17 e 18) deverá ser utilizado para especificar o valor que a propriedade deverá ter. Os operadores lógicos que podem ser utilizados são: == (igualdade), != (diferente), > (maior), >= (maior ou igual), < (menor) e <= (menor ou igual).

O elemento *Attributes* possui a propriedade opcional *From* (linhas 45 e 46). Se esta propriedade for utilizada, seu valor deverá indicar um tipo de AR previamente registrado no SRD. O Serviço de Contexto irá pesquisar no SRD os ARs que estejam localizados no mesmo nodo do AR especificado pelo *URI* e irá retornar as propriedades de contexto daqueles ARs que obedecerem às cláusulas especificadas pelos elementos *Attribute*.

O atributo *Min* do elemento opcional *CollectTime* (linhas 1 a 25) especifica o limite inferior de tempo referente ao momento da coleta dos dados junto ao(s) sensor(es). Ou seja, especifica que as medidas a serem consideradas deverão ser posteriores a esta data. É importante destacar que a arquitetura não impõe uma política referente à sincronização dos relógios dos elementos constituintes da infra-estrutura (ARs, SRD, SC, SD e aplicações) ficando a cargo dos projetistas destes elementos decidirem qual política será adotada.

O elemento *CollectInterval* especifica o intervalo mínimo entre cada leitura de dados dos sensores. Dependendo do tipo da aplicação este parâmetro juntamente com o parâmetro *CollectTime* pode ser de vital importância. Sendo possível estipular um critério para analisar a validade dos dados retornados.

Finalmente, o elemento *Results* (linhas 36 a 41) especifica o número máximo (atributo *Max*) de leituras deverão ser retornadas. Caso este elemento não for especificado, será retornada apenas uma leitura.

Na Listagem 11 é apresentada uma consulta ao Serviço de Contexto que ilustra os elementos discutidos acima. Este exemplo ilustra uma consulta realizada pela infra-estrutura baseada em contratos denominada CR-RIO (a ser apresentada na Seção 5.2) que utiliza os elementos propostos neste trabalho como base para o processo de monitoramento dos elementos que compõem as aplicações que esta infra-estrutura gerencia. A aplicação a ser apresentada na Seção 5.2 acrescenta características de tolerância a faltas adaptativas para

aplicações Web construídas com o intermédio do servidor HTTP da fundação Apache (Apache.org, 2007a) e o container JSP/Servlet Tomcat (Apache.org 2007b).

```
 1 <ContextQuery>
 2   <Target URI="services.Tomcat">
 3     <Attributes>
 4       <Attribute Name="ApplicationName" Value="SalesOrder"/>
 5     </Attributes>
 6     <Attributes From="Processing">
 7       <Attribute Name="CPUIdle" op=">" Value="40"/>
 8       <Attribute Name="FreeMemory" op=">=" Value="512"/>
 9       <CollectTime Min="2009-05-23T10:13:20"/>
10       <Results Max="5"/>
11       <CollectInterval Min="300"/>
12     </Attributes>
13   </Target>
14   <Target URI="services.Apache">
15     <Attributes From="Communication">
16       <Attribute Name="responseTime" op="<" Value="200"/>
17     </Attributes>
18   </Target>
19 </ContextQuery>
```

Listagem 11 - Exemplo de consulta síncrona ao Serviço de Contexto

Este documento XML contém duas consultas que serão efetuadas em dois ARs. O alvo principal da primeira consulta (linhas 2 a 13) é o AR identificado pelo *URI* "*services.Tomcat*" (linha 2). Como o elemento *Attributes* da linha 3 não possui o atributo *From* todos os elementos presentes neste atributo dizem respeito ao AR "*services.Tomcat*". Neste caso, somente medidas efetuadas quando a aplicação "*SalesOrder*" estava implantada serão considerados na consulta.

Na linha 6 é especificada uma restrição de contexto a ser aplicada relacionada ao AR *Processing* (atributo *From* da linha 6). Apenas medidas relativas ao servidor *Tomcat* cujo AR do tipo *Processing* (que esteja implantado em seu nodo de processamento) com *CPUIdle* maior que 40% e o atributo *FreeMemory* maior ou igual a 512 MB serão retornadas como resultado da consulta.

Deverão ser retornadas no máximo cinco medidas referentes ao AR *services.Tomcat* (elemento *Results* da linha 8) e o intervalo entre cada medida deverá ser de no mínimo 5 minutos (elemento *CollectInterval* da linha 9). Além disso, as medidas deverão ser posteriores às 10:13:20h do dia 23 de maio de 2009 (elemento *CollectTime* da linha 7). Dependendo da natureza das aplicações, pode ser necessário um determinado número de leituras junto aos ARs em um intervalo de tempo previamente conhecido para que os dados de contexto coletadas possam ser utilizadas pela aplicação.

A segunda consulta (linhas 14 a 18) tem como alvo principal o AR identificado pelo *URI* "*services.Apache*" (linha 14). Existe apenas a restrição de contexto relacionada ao AR *Communication* que deverá ser aplicada às instâncias Apache. Este AR (não mostrado no texto) contém medidas relacionadas ao tempo de resposta fim-a-fim das aplicações. Na consulta do exemplo, deverão ser retornados dados do SC que tiveram tempo de resposta menor que 200ms.

A Listagem 12 apresenta o documento XSD que descreve as respostas retornadas às consultas síncronas realizadas junto ao Serviço de Contexto.

```
 1 <xsd:element name="ContextResponse">
 2   <xsd:complexType>
 3     <xsd:sequence>
 4       <xsd:element name="ResourceInfo" minOccurs="0"
 5                    maxOccurs="unbounded">
 6         <xsd:complexType>
 7           <xsd:sequence>
 8             <xsd:element name="Attributes" minOccurs="0">
 9               <xsd:complexType>
10                 <xsd:sequence>
11                   <xsd:element name="Attribute" maxOccurs="unbounded">
12                     <xsd:complexType>
13                       <xsd:attribute name="Name" use="required"
14                                      type="xsd:string" />
15                       <xsd:attribute name="Type" use="required"
16                                      type="xsd:string" />
17                       <xsd:attribute name="Value" use="required"
18                                      type="xsd:anySimpleType" />
19                       <xsd:attribute name="Units" type="xsd:string" />
20                     </xsd:complexType>
21                   </xsd:element>
22                 </xsd:sequence>
23                 <xsd:attribute name="From" use="required"
24                                type="xsd:string" />
25                 <xsd:attribute name="Interval" type="xsd:integer" />
26                 <xsd:attribute name="Updated" type="xsd:dateTime"
27                                use="required" />
28               </xsd:complexType>
29             </xsd:element>
30           </xsd:sequence>
31         <xsd:attribute name="URI" use="required" type="xsd:anyURI" />
32       </xsd:complexType>
33     </xsd:element>
34     </xsd:sequence>
35   </xsd:complexType>
36 </xsd:element>
```

Listagem 12 - XSD de resposta à consulta síncrona junto ao Serviço de Contexto

O elemento *ContextResponse* presente na linha 1 indica que se trata de uma resposta a uma consulta síncrona. O elemento *ResourceInfo* (linhas 4 e 5) contém as informações de contexto referentes a cada AR que foi o alvo principal da consulta e que é identificado pelo atributo *URI* (linha 31). O elemento *Attributes* (linha 8) contém os atributos dos ARs que

participaram da consulta, sendo que o tipo de cada AR é especificado pelo atributo *From* (linhas 23 e 24). Caso este atributo não for especificado, é assumido que os atributos pertencem ao AR principal da consulta identificado pelo *URI*. Caso o atributo *From* esteja especificado, ele irá conter o tipo do AR que os atributos relacionados pertencem.

Cada atributo é especificado pelo elemento *Attribute* (linha 11) sendo que seu nome, tipo, valor e unidade de medida são especificados pelos atributos *Name* (linhas 13 e 14), *Type* (linhas 15 e 16), *Value* (linhas 17 e 18) e *Units* (linha 19) respectivamente. Os atributos *Interval* (linha 25) e *Updated* (linha 26) do elemento *Attributes* representam o intervalo (em segundos) utilizado pelo AR para coletar as medidas junto aos sensores e a data da última leitura dos sensores respectivamente.

A Listagem 13 apresenta uma possível resposta à consulta da Listagem 11.

```
 1  <ContextResponse>
 2    <ResourceInfo URI="services.Tomcat">
 3      <Attributes  From="Processing" Interval="600"
 4                   Updated="2009-05-23T11:12:20">
 5        <Attribute Name="CPUIdle" Type="float" Value="90" Units="%"/>
 6        <Attribute Name="FreeMemory" Type="float" Value="1348"
 7                   Units="MB"/>
 8      </Attributes>
 9      <Attributes  From="Processing" Interval="600"
10                   Updated="2009-05-23T10:35:14">
11        <Attribute Name="CPUIdle" Type="float" Value="70" Units="%"/>
12        <Attribute Name="FreeMemory" Type="float" Value="896"
13                   Units="MB"/>
14      </Attributes>
15   </ResourceInfo>
16   <ResourceInfo  URI="services.Apache">
17     <Attributes  From="Communication" Interval="300"
18                  Updated="2009-05-23T11:24">
19       <Attribute Name="responseTime" Type="integer" Value="180"
20                  Units="ms"/>
21     </Attributes>
22   </ResourceInfo>
23 <ContextResponse>
```

Listagem 13 - Exemplo de resposta de consulta síncrona ao serviço de contexto

Neste exemplo, são retornadas informações sobre o contexto de execução (elemento *ResourceInfo*) dos ARs identificados pelos URIs "*services.Tomcat*" (linha 2) e "*services.Apache*" (linha 10). Nas linhas 3 a 15 encontram-se duas medidas do AR de processamento (*Processing*) referentes ao percentual de tempo ocioso da CPU (linhas 5 e 11) e a quantidade de memória livre (linhas 6-7 e 12-13).

Nas linhas 16 a 22 está o resultado da consulta referente ao AR *services.Apache*. No exemplo, o tempo de resposta (*responseTime*) do AR *Communication* é igual a 180ms.

Como foi especificado um número máximo de leituras igual a cinco e inferiores às 10:13:20h do dia 23 de maio de 2009, o Serviço de Contexto retornou as medidas que correspondem a estas restrições: no caso duas leituras apenas.

Se o cliente necessitar de informações de contexto assim que uma mudança ocorre, uma interação assíncrona com o CS se torna necessária. Para tanto devem ser utilizadas as trocas de mensagens descritas na seção seguinte.

3.3.3.2 Consultas assíncronas ao contexto (Método *Push*)

Para o projetista a diferença entre uma consulta assíncrona e uma síncrona é clara: a estrutura do programa é diferente. Para receber respostas assíncronas do Serviço de Contexto um objeto que implemente o padrão *Observer* (Gamma et al., 1995) tem que ser registrado junto ao CS. A Listagem 14 apresenta o XSD das mensagens utilizadas para este fim.

```
 1 <xsd:element name=”ResourceObserver”>
 2   <xsd:complexType>
 3     <xsd:sequence>
 4       <xsd:element name="URI" type="xsd:string" minOccurs="1"
 5                    maxOccurs="1" />
 6       <xsd:element name="ResourceAgent" type="xsd:string" />
 7       <xsd:element name="Description" type="xsd:string" />
 8       <xsd:element name="ResourceType" type="xsd:string" />
 9     </xsd:sequence>
10   </xsd:complexType>
11 </xsd:element>
```

Listagem 14 - XSD para registro de observadores nas consultas assíncronas

O elemento *ResourceObserver* (linha 1) indica que se trata de uma mensagem com o objetivo de efetuar o registro de um observador. O elemento URI (linhas 4 e 5) informa a URI do elemento que está se registrando e servirá como referência a ser utilizada pelo CS quando o mesmo for informar as mudanças ocorridas. O CS armazena uma lista contendo os observadores registrados e informa os observadores sempre que ocorre uma mudança nas propriedades dos ARs que correspondam aos critérios especificados na mensagem de registro. O informe é realizado através do URI informado pelo observador. Uma vez que o CS conheça a URI do observador, torna-se capaz de identificar o mesmo junto ao Serviço de Diretório, obtendo sua descrição. A descrição do observador (ele mesmo é um AR) contém as tecnologias com as o observador se comunica (serviços web, RMI, etc.) e utiliza a URL associada para se comunicar com o observador informando assim o estado do(s) AR(s) observado(s).

O elemento *ResourceAgent* (linha 6) indica a URI do AR em cujas mudanças o observador está interessado em ser notificado. O elemento opcional *Description* (linha 7) é utilizado para fornecer uma descrição legível para humanos sobre o observador. Finalmente o elemento *ResourceType* (linha 8) indica o tipo de ARs em que o observador está interessado em ser notificado sobre mudanças ocorridas. Os elementos *ResourceType* e *ResourceAgent* são mutuamente exclusivos. Um observador tem que registrar o interesse em um tipo de AR (*ResourceType*) ou em um AR específico (*ResouceType*). Múltiplos registros são permitidos para cada observador. É importante ressaltar que a interação é sempre entre o observador e o CS. O observador nunca interage diretamente com o AR.

No exemplo da Listagem 15, o cliente identificado por seu URI (linha 2) está registrando interesse em ser notificado quando mudanças ocorrerem nos recursos do tipo *Processing* (linha 4).

```
1 <ResourceObserver>
2   <URI>http://grajau:2345/SelectorObserver</URI>
3   <Description>CR-RIO Selector</Description>
4   <ResourceType>Processing</ResourceType>
5 </ResourceObserver>
```

Listagem 15 - Registrando um observer

Por padrão todo AR é programado para notificar um CS assim que uma mudança em suas propriedades ocorre. A informação do novo estado é "empurrada" (*pushed*) para o CS registrado junto ao AR. Após receber uma notificação de um AR, o CS notifica os *Observers* registrados pelos clientes. A Listagem 16 contém o XSD que representa as mensagens de notificação que um AR envia para o CS informando seu estado interno.

O elemento *ResourceState* (linha 1) indica que se trata de uma mensagem contendo o estado interno de um AR. O elemento *Type* (linha 4) indica o tipo do AR, o elemento *CollectTime* (linha 5) informa o horário em que as medidas foram realizadas junto aos sensores que o AR encapsula e o elemento *Interval* (linha 6) informa o intervalo entre as medidas.

O elemento *Attributes* (linha 7) encapsula os valores das propriedades que compõem o AR. O atributo *Name* informa o nome da propriedade, o atributo *Type* (linha 14) informa o tipo da propriedade, o atributo *Value* (linha 16) informa o valor medido e o elemento *Units* (linha 18) informa a unidade de medida correspondente. Por fim o elemento *URI* informa o identificador do AR.

```
 1 <xsd:element name=”ResourceState”>
 2   <xsd:complexType>
 3     <xsd:sequence>
 4       <xsd:element name="Type" type="xsd:string" />
 5       <xsd:element name="CollectTime" type="xsd:dateTime" />
 6       <xsd:element name="Interval" type="xsd:int" />
 7       <xsd:element name="Attributes">
 8         <xsd:complexType>
 9           <xsd:sequence>
10             <xsd:element name="Attribute" maxOccurs="unbounded">
11               <xsd:complexType>
12                 <xsd:attribute name="Name" use="required"
13                                type="xsd:NCName" />
14                 <xsd:attribute name="Type" use="required"
15                                type="xsd:NCName" />
16                 <xsd:attribute name="Value" use="required"
17                                type="xsd:anySimpleType" />
18                 <xsd:attribute name="Units" type="xsd:string" />
19               </xsd:complexType>
20             </xsd:element>
21           </xsd:sequence>
22         </xsd:complexType>
23       </xsd:element>
24       <xsd:element name="URI" type="xsd:anyURI" />
25     </xsd:sequence>
26   </xsd:complexType>
11 </xsd:element>
```

Listagem 16 - XSD da notificação de mudanças no contexto

A Listagem 17 apresenta um exemplo de uma mensagem trocada entre um AR do tipo *Processing* (linha 2) e um CS.

```
 1  <ResourceState>
 2    <Type>Processing</Type>
 3    <CollectTime>2008-09-25 T 23:24:15</CollectTime>
 4    <Interval>5</Interval>
 5    <Attributes>
 6      <Attribute Name=”CPUClock” Value=”1800” />
 7      <Attribute Name=”TotalMemory” Value=”1024” />
 8    …
 9   </Attributes>
10   <URI>leme.pel.uerj.br/Processing</URI>
11 </ResourceState>
```

Listagem 17 - Exemplo de informe assíncrono do estado de um AR

Neste exemplo, o AR identificado de forma única através do *URI* “*leme.pel.uerj.br*” informa ao CS que os valores de seus atributos (linhas 5 a 9) foram atualizados em 25/09/2008 às 23:24:15h (elemento *CollectTime* da linha 3) e que o intervalo entre cada medida é de 5 segundos (elemento *Interval* linha 4).

3.3.4 Serviço de Descoberta (SD)

O Serviço de Descoberta provê uma interface que pode ser utilizada para identificar Agentes de Recursos registrados (de quaisquer tipos possíveis) em um Serviço de Registro e Diretório que sejam de um dado tipo e que cumpram um conjunto de restrições de contexto. A lista de ARs retornada pelo SD pode ser posteriormente filtrada ou classificada pela aplicação de forma a identificar o melhor recurso que satisfaça às restrições especificadas.

Como discutido anteriormente, o SD consulta o SRD e o SC, se necessário, pra efetuar as consultas. Para descobrir um novo recurso um cliente submete uma consulta ao SD e recebe como resposta uma lista de referências (URIs) para os recursos. Uma consulta é composta obrigatoriamente pelo tipo do recurso e restrições de contexto que devem ser satisfeitas. Por exemplo, em um cenário hipotético, uma aplicação pode necessitar de um Servidor *Web* que tenha um mínimo de 50 *threads* disponíveis para atender requisições concorrentes. Ao receber esta consulta o SD obtém do RDS uma lista de referências para todos os ARs registrados do tipo "Servidor Web" juntamente com todas as informações de contexto relacionadas com eles que são estáticas. Ou seja, que não mudam ao longo do tempo. Um filtro inicial é aplicado a esta lista com o objetivo de excluir aqueles recursos que não satisfaçam restrições estáticas que tenham sido especificadas na consulta. Para as referências remanescentes na lista, se houver alguma propriedade dinâmica no conjunto de restrições da consulta, o SD obtém do SC a informação complementar para cada recurso na lista. Em seguida o SD aplica um filtro composto por estas restrições relacionadas com propriedades dinâmicas e descarta os recursos que não as satisfazem. A lista de ARs remanescentes é finalmente retornada para o cliente.

Como exemplo de uma propriedade estática, podemos citar a capacidade de um Servidor Web executar aplicações PHP. Como exemplo de uma propriedade dinâmica, podemos citar o tempo de resposta do servidor que deveria ser inferior a 50ms.

Outro exemplo de aplicação poderia ser um HSS, que recebe em tempo real informações médicas de pacientes. Tais servidores poderiam estar localizados na residência do paciente, em um contexto remoto assistido, ou em um hospital. Aplicações como estas poderiam utilizar o SD para realizar consultas com o objetivo de localizar o cômodo em que o paciente se encontra e em seguida localizar os sensores (ARs) que se encontram no cômodo (por exemplo, um AR de temperatura e outro de nível de luminosidade). Para posteriormente atuar sobre os dispositivos adequando o ambiente às preferências do paciente ou a um plano

de cuidados médicos.

A seguir são apresentadas as mensagens enviadas ao SD para realizar consultas e o formato das mensagens de retorno.

3.3.4.1 Formato das consultas ao Serviço de Descoberta

A Listagem 18 apresenta o esquema XML das mensagens utilizadas nas consultas realizadas junto ao Serviço de Descoberta.

```
 1 <xsd:element name="ResourceQuery">
 2   <xsd:complexType>
 3     <xsd:sequence>
 4       <xsd:element name="MaxResults" type="xsd:integer"
 5                    minOccurs="0" />
 6       <xsd:element name="Constraints" minOccurs="0">
 7         <xsd:complexType>
 8           <xsd:sequence>
 9             <xsd:element name="Attribute" maxOccurs="unbounded">
10               <xsd:complexType>
11                 <xsd:attribute name="Name" use="required"
12                                type="xsd:string" />
13                 <xsd:attribute name="op" use="required" />
14                 <xsd:attribute name="Value" use="required" />
15               </xsd:complexType>
16             </xsd:element>
17           </xsd:sequence>
18           <xsd:attribute name="From" type="xsd:string" />
19         </xsd:complexType>
20       </xsd:element>
21     </xsd:sequence>
22     <xsd:attribute name="Type" type="xsd:string" use="required" />
23     <xsd:attribute name="Strict" type="xsd:boolean" default="true" />
24   </xsd:complexType>
25 </xsd:element>
```

Listagem 18 - XSD de consultas realizadas ao Serviço de Descoberta

O elemento *ResourceQuery* (linha 1) indica que a mensagem se trata de uma consulta ao Serviço de Descoberta. O número de referências a Agentes de Recursos que atendem às restrições especificadas na consulta podem opcionalmente ser limitado através do uso do elemento *MaxResults* (linha 4). O atributo *Type* (linha 22) especifica o tipo de recurso que se deseja consultar. Caso seja especificado o caractere "*" todos os tipos de recursos serão considerados na consulta. Caso o atributo *Strict* tenha o valor igual à *false* os subtipos do tipo especificado serão considerados na consulta. Este atributo é passado na consulta ao SRD conforme explicado na Seção 3.3.2 (Listagem 8).

O elemento *Constraints* (linha 6) contém as definições das restrições de contexto a

serem especificadas na consulta. O elemento *Attribute* (linha 9) pode ocorrer num número ilimitado de vezes e é utilizado para especificar os valores que os atributos deverão obedecer. O atributo XML *Name* (linha 11) é utilizado para especificar o nome do atributo do AR, o atributo XML *op* (linha 13) especifica o operador de comparação a ser utilizado (==, !=, <, <=, > e >=), e o atributo XML *Value* especifica o valor que a propriedade do AR deve ter.

O atributo *From* (linha 18) do elemento *Constraints* indica a qual tipo de AR os atributos pertencem. Caso o atributo *From* não seja utilizado, os atributos enumerados serão considerados como pertencentes à classe de ARs especificada no atributo *Type* da consulta. Caso o atributo *From* seja utilizado, assume-se que os atributos enumerados pertencem ao tipo especificado pelo atributo *From*. Com isso é possível especificar, por exemplo, que se deseja localizar ARs de Servidores *Web* que são aptos a executar aplicações PHP cujos ARs de processamento presentes na mesma localização do Servidor *Web* indicam utilização de CPU abaixo de 60%. A Listagem 19 apresenta um exemplo onde estas duas formas de utilização do atributo *From* são utilizadas.

```
1 <ResourceQuery Type="WebServer">
2   <MaxResults>5</MaxResults>
3   <Constraints>
4     <Attribute Name="application" op="==" Value="PHP"/>
5   </Constraints>
6   <Constraints From="SimpleProcessing">
7     <Attribute Name="CPUIdle" op=">=" Value="60" />
8   </Constraints>
9 </ResourceQuery>
```

Listagem 19 - Exemplo de consulta ao serviço de descoberta

A linha 1 indica que se deseja localizar ARs que sejam do tipo *WebServer* e que o número de referências deverá ser limitado a 5 (elemento *MaxResults* na linha 2). Caso este elemento seja omitido, todas as referências localizadas serão retornadas para o cliente.

O elemento *Constraints* da linha 3 indica que os atributos contidos nele se referem ao tipo *WebServer*, pois nenhum tipo foi especificado através do atributo *From*. Ou seja, o atributo "*application*" pertence ao tipo *WebServer* e, neste exemplo, deverá possuir o valor "PHP".

Em seguida na linha 6, o elemento *Constraints* possui o atributo *From* cujo valor é *"SimpleProcessing*". Assim, está sendo declarado que os atributos contidos no elemento *Constraints* pertencem ao tipo *SimpleProcessing*. E que os recursos pesquisados deverão ter a propriedade *CPUIdle* com valores iguais ou superiores a 60.

Com isso, o SD irá utilizar o SRD para localizar os ARs do tipo *WebServer*. Para cada

referência retornada, o SD irá verificar se a propriedade *application* é igual ao valor PHP (aqui estamos considerando que esta é uma propriedade estática). Para aqueles ARs que atendem a esta restrição, o SD irá utilizar o SRD para obter a referência do AR do tipo *SimpleProcessing* que se localiza no mesmo nodo que o AR do tipo *WebServer* sendo verificado no momento. Uma vez que o SD obtém a referência para este AR ele constata que a propriedade *CPUIdle* é dinâmica e efetua uma consulta síncrona ao SC conforme descrito na Seção 3.3.2.1 e elimina aqueles ARs do tipo *WebServer* que não possuam a propriedade *CPUIdle* do AR *SimpleProcessing* abaixo de 60. Finalmente uma lista (que pode ser vazia) contendo os ARs restantes é retornada ao cliente.

3.3.4.2 Formato da resposta ao Serviço de Descoberta

O XSD utilizado para descrever a resposta a uma consulta realizada ao Serviço de Descoberta é apresentado na Listagem 20.

```
 1 <xsd:element name="DiscoveryResponse">
 2   <xsd:complexType>
 3     <xsd:sequence>
 4       <xsd:element name="ResourceInfo" maxOccurs=”unbounded”>
 5         <xsd:complexType>
 6           <xsd:sequence>
 7             <xsd:element name="Attributes" maxOccurs="unbounded">
 8               <xsd:complexType>
 9                 <xsd:sequence>
10                   <xsd:element name="Attribute" maxOccurs="unbounded">
11                     <xsd:complexType>
12                       <xsd:attribute name="Name" use="required"
13                                      type="xsd:string" />
14                        <xsd:attribute name="Value" use="required"
15                                       type="xsd:string" />
16                     </xsd:complexType>
17                   </xsd:element>
18                 </xsd:sequence>
19               </xsd:complexType>
20               <xsd:attribute name="From" type="xsd:string"
21                              use="required"/>
22             </xsd:element>
23           </xsd:sequence>
24           <xsd:attribute name="URI" use="required" type="xsd:anyURI" />
25         </xsd:complexType>
26       </xsd:element>
27     </xsd:sequence>
28   </xsd:complexType>
29 </xsd:element>
```

Listagem 20 - XSD de resposta de consulta ao Serviço de Descoberta

O elemento *DiscoveryResponse* na linha 1 indica que a mensagem é uma resposta a uma consulta realizada ao Serviço de Descoberta. O elemento *ResourceInfo* (linha 4) possui

um atributo chamado URI (linha 24) que contém a referência para o recurso que atendeu às restrições especificadas na consulta. Cada recurso que tenha atendido às restrições passadas na consulta terá um elemento *ResourceInfo* contendo seu URI.

O elemento *Attributes* (linha 7) contém todas as propriedades do recurso que atendeu às restrições especificadas na consulta. O tipo do recurso é identificado pelo atributo *From* (linhas 20 e 21) do elemento *Attributes*. As propriedades *Name* e *Value* (linhas 12 a 15) representam o nome da propriedade do recurso e o valor atual respectivamente.

Um exemplo de resposta à consulta especificada na Listagem 19 é apresentado na Listagem 21.

```
 1 <DiscoveryResponse>
 2   <ResourceInfo URI="apache.grajau.pel.uerj.br"/>
 3     <Attributes From="WebService">
 4       <Attribute Name="application" Value="PHP"/>
 5       <Attribute Name="version" Value="2.0.1" />
 6     </Attributes>
 7     <Attributes From="SimpleProcessing">
 8       <Attribute Name="CPUClock" Value="3000"/>
 9       <Attribute Name="TotalMemory" Value="2048"/>
10       <Attribute Name="FreeMemory" Value="644"/>
11       <Attribute Name="CPUIdle" Value="80" />
12     </Attributes>
13   </ResourceInfo>
14 </xsd:element>
```

Listagem 21 - Exemplo de resposta a uma consulta realizada ao Serviço de Descoberta

Na linha 2 encontra-se o URI do Servidor Web encontrado que atende às restrições passadas na consulta apresentada na Listagem 19. Nas linhas 3 a 6 estão os nomes e valores das propriedades deste Servidor Web e nas linhas 7 a 12 são listadas as propriedades do AR de processamento que teve sua propriedade *CPUIdle* especificada como uma das cláusulas da consulta (os recursos deveriam ter valores maiores ou iguais a 60).

Cabe citar que o Serviço de Descoberta pode ser especializado ainda mais para melhorar o processo de filtragem de forma, por exemplo, a selecionar o recurso que tenha os melhores níveis de qualidade. Para isso, a aplicação poderia empregar uma política que poderia dar preferência aos Servidores Web com níveis mais altos de disponibilidade de CPU (*CPUIdle*), respeitando-se as restrições especificadas para as demais propriedades. Técnicas mais sofisticadas poderiam ser utilizadas, por exemplo, a utilização de funções de utilidade poderia ter como entrada um conjunto de propriedades de contexto onde cada uma poderia ter um peso/prioridade que poderiam ser utilizadas na política de seleção dos recursos.

CAPÍTULO 4 IMPLEMENTAÇÃO DE REFERÊNCIA

Uma implementação de referência dos serviços propostos, chamada de *Contextual Discovery of Resources Framework* (CDRF), foi desenvolvida na forma de um *framework* de componentes e classes usando a versão 6 da linguagem Java. Os princípios desta implementação são: (i) dependência mínima entre os serviços, e (ii) mínima dependência entre protocolos de comunicação.

As classes abstratas do framework, que encapsulam o comportamento padrão dos serviços, foram especializadas como Serviços Web. Neste sentido foi utilizado a XML Web Services API, JAX-WS 2.0 (JAX-WS, 2008). Esta API possibilita a comunicação entre os serviços propostos e entre estes serviços e os Agentes de Recursos através de RPC usando mensagens SOAP 1.0.

Os serviços da infra-estrutura e os Agentes de Recursos foram desenvolvidos como *deamons* independentes, cada qual executando como um Serviço Web. Utilizamos para isso o servidor HTTP embutido no conjunto de ferramentas disponibilizado pela plataforma Java Versão 6.0, que foi configurado para encaminhar requisições às classes dos serviços propostos, sem a necessidade de instalar e configurar um servidor de aplicação JEE completo (JEE, 2008).

A programação dos Serviços Web e dos clientes é simplificada pela API JAX-WS com o uso de anotações. O trecho de código apresentado na Listagem 22 exemplifica como a interface do AR é anotada para especificar que o elemento deverá ser implantado pela infra-estrutura da plataforma Java como um Serviço Web e que a comunicação a ser utilizada deverá ser no estilo RPC (RPC, 2008).

```
1  @WebService
2  @SOAPBinding(style = SOAPBinding.Style.RPC)
3  public interface ResourceAgent {…}
```

Listagem 22 - Exemplo de utilização da API JAX-WS na infra-estrutura

A seguir detalhamos a implementação dos elementos que constituem a infra-estrutura desenvolvida.

4.1 Implementação de Agentes de Recursos

O principal ponto de extensão do *framework* é o Agente de Recurso. Novos tipos de recursos podem ser facilmente integrados à infra-estrutura. O diagrama de classes apresentado na Figura 4 apresenta os principais elementos que constituem um AR e que interagem com um AR. A interface *ResourceAgent* expõe as operações comuns para todo tipo de recurso e que são publicamente acessíveis, constituindo assim o contrato mínimo que todo AR tem que obedecer. O contrato estabelecido por esta interface dita que todo AR tem que ter um método responsável por inicializar e parar o processo de monitoramento (*startMonitoring* e *stopMonitoring* respectivamente). Um método específico é utilizado para obter o estado interno do AR (*getResourceState*). Este método retorna a representação em XML do estado interno do AR conforme apresentado na Listagem 16 da Seção 3.3.3.2.

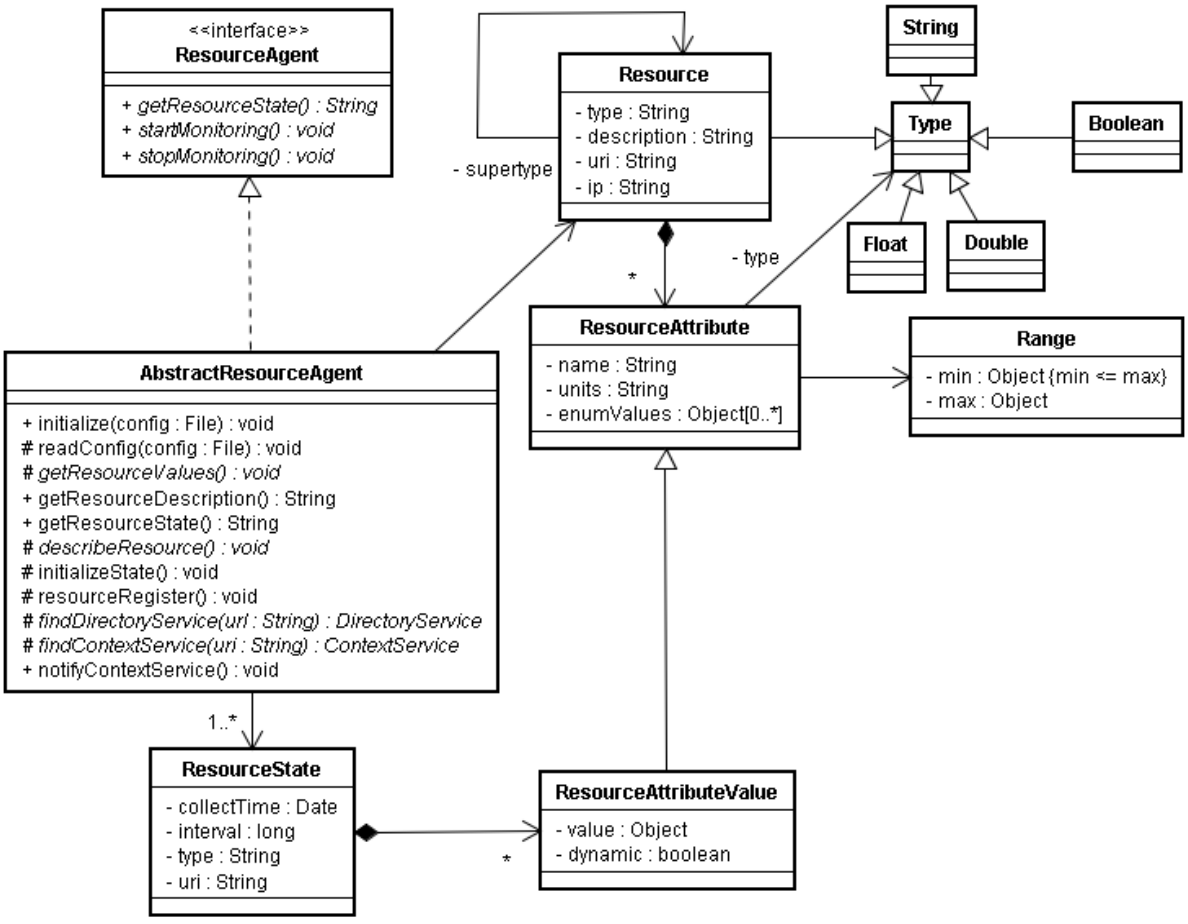


Figura 4 - Classes básicas de um Agente de Recurso

O *framework* disponibiliza a classe *AbstractResourceAgent* que implementa a interface *ResourceAgent*. A partir desta classe, novos Agentes de Recursos podem ser criados de forma bastante simplificada. Os três métodos presentes na interface *ResourceAgent* são

implementados pela classe *AbstractResourceAgent* que expõe um conjunto de métodos abstratos que deverão ser implementados por suas classes. Esta classe faz uso do padrão de projeto denominado *Template Method* (Gamma et al., 1995).

A classe *AbstractResourceAgent* mantém a representação interna do recurso sendo monitorado (classe *Resource* apresentada na Sseção 3.1). Esta classe será utilizada para fornecer ao Serviço de Diretórios a descrição do AR no momento de sua inicialização. Além da representação do AR, a classe *AbstractResourceAgent* mantém internamente uma ou mais instâncias da classe *ResourceState*. Esta classe contém os valores dos estados internos de um AR identificado pelo atributo URI. O atributo *collectTime* armazena a data e o horário em que os valores foram lidos, o atributo *interval* armazena o intervalo em milissegundos desde que a última medida foi efetuada. A classe *ResourceAttributeValue* armazena os valores individuais (atributo *value*) de cada atributo do recurso e possui o indicativo se a propriedade é dinâmica ou não (atributo *dynamic*).

O primeiro método da classe *AbstractResource* a ser abordado é o método *initialize*. O diagrama de seqüência da Figura 5 apresenta este método com maiores detalhes.

Este método é invocado no momento em que o AR é inicializado e recebe como parâmetro um objeto do tipo *java.io.File* contendo o arquivo de configuração a ser utilizado pelo AR. Na implementação de referência este arquivo é armazenado em formato XML (detalhes podem ser consultados em (Rodrigues, 2009d). A classe responsável por inicializar o AR é representada no diagrama pela classe *Main* e deverá ser criada pelo projetista do AR. O método *initialize* invoca o método *readConfig* que é responsável por efetuar o *parsing* do arquivo XML. Neste arquivo há a localização (URL) do Serviço de Diretório. Esta URL será utilizada pelo método *findDirectoryService* no processo de localização do SRD. Como resultado deste processo de localização, um *proxy* através do qual qualquer comunicação futura com o SRD será efetuada é retornado.

Uma vez localizado o SRD, é invocado o método *findContextService* passando-se como parâmetro o URI do Serviço de Contexto. Uma vez que o Serviço de Contexto tem que estar registrado no SRD, este URI será utilizado para localizar a referência para o Serviço de Contexto no SRD.

O próximo passo é descrever o recurso representado pelo AR. Esta tarefa é realizada pelo método abstrato *describeResource*. Todos os ARs construídos a partir do *framework* deverão implementar este método, criando um objeto do tipo *Resource* que conterá a

descrição do recurso.

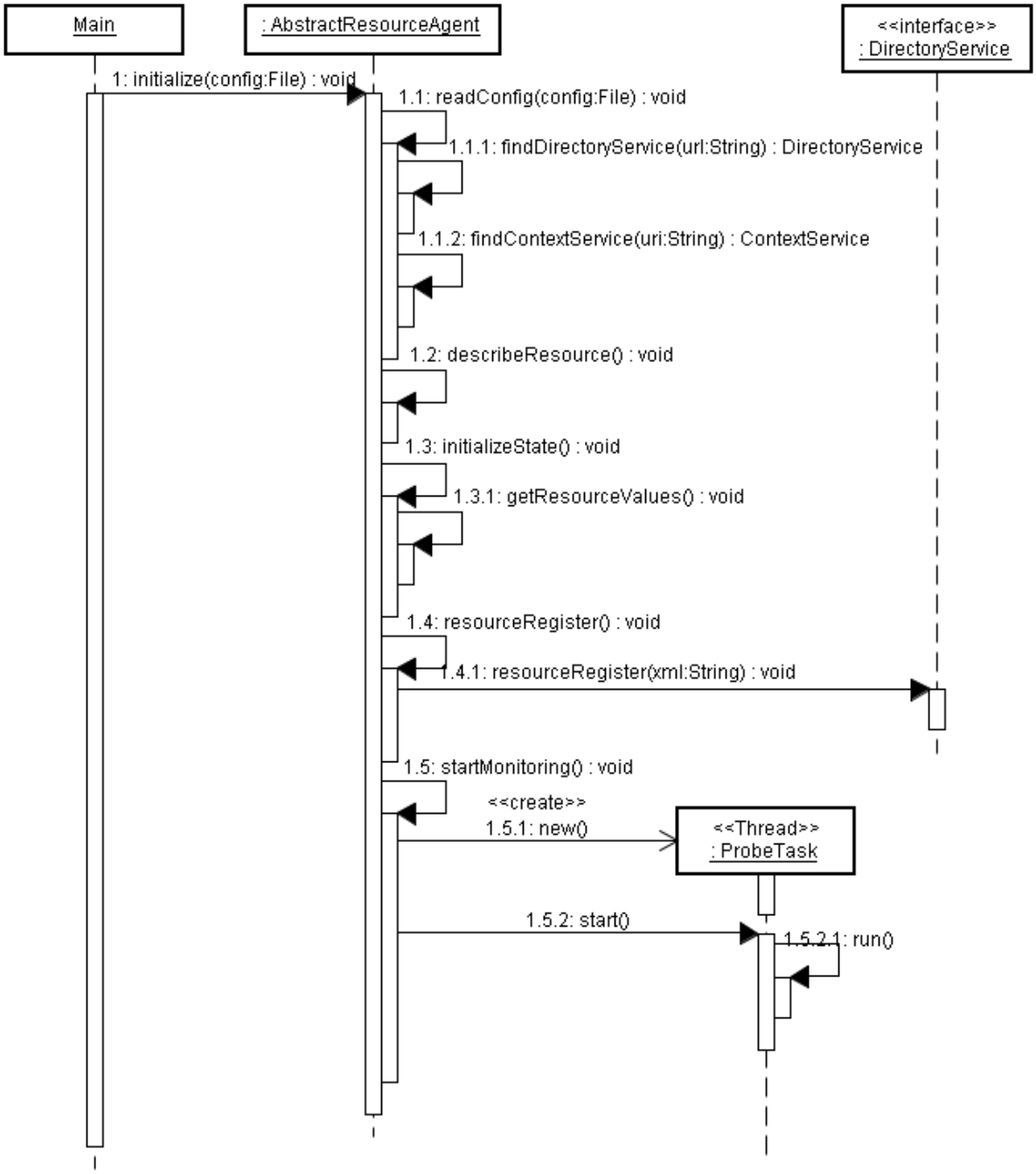


Figura 5 - Diagrama de seqüência da operação de inicialização de um AR

Uma vez que algumas propriedades não-dinâmicas dos ARs precisam ser obtidas quando um AR é inicializado, o método *initializeState* é invocado. Como exemplo deste tipo de propriedade, pode-se citar a quantidade total de memória de um computador. O projetista pode utilizar este método para obter esta propriedade. Para tanto, o método abstrato *getResourceValues* é invocado pelo método *initializeState*. Este método constrói uma instância do objeto *ResourceState*. O projetista do AR que irá definir uma implementação

concreta deste método fará chamadas às APIs específicas dos sensores, lidando com as especificidades de cada um deles. O projetista irá ainda armazenar no objeto do tipo *ResourceState* os valores lidos. Uma vez que o estado do AR é conhecido, o mesmo é registrado junto ao SRD através do método *resourceRegister* do SRD. Este método recebe como parâmetro a descrição do AR em formato XML. Uma vez que o registro do AR tenha sido realizado com sucesso o método *startMonitoring* é invocado. Este método cria um objeto do tipo *ProbeTask* que é na verdade a *Thread* responsável por atualizar periodicamente o estado do AR invocando o método *getResourceValues*.

Os projetistas de ARs deverão, portanto implementar os seguintes métodos: *findDirectoryService*, *findContextService, describeResources* e *getResourceValues*. Todo o resto das interações entre os elementos de mais alto nível da infra-estrutura são abstraídos do projetista.

Os métodos *describeResources* e *getResourceValues* deverão ser definidos pelos projetistas, pois são específicos de cada classe de AR (ARs de temperatura, pressão sangüínea, processamento, etc.) e os métodos *findDirectoryService* e *findContextService* também deverão ser definidos, pois são eles que decidirão como esta comunicação será realizada. Se o projetista decidir que o AR irá se comunicar através de RMI ou através de Serviços Web, por exemplo, ele deverá lidar com os detalhes relacionados com estes tipos de comunicação. Obviamente, o AR irá se comunicar apenas com SRDs e CSs que utilizam a(s) mesma(s) tecnologia(s) de comunicação. No entanto, todos os detalhes referentes às interações (os protocolos de mais alto nível) entre os elementos da infra-estrutura serão abstraídos do projetista. Inclusive, nada impede que um AR seja capaz de utilizar RMI, Serviços Web ou quaisquer outras tecnologias de comunicação simultaneamente. Na implementação de referência construímos classes utilitárias que lidam com os detalhes envolvidos na comunicação com os Serviços Web e estamos desenvolvendo classes para comunicação via RMI.

Uma vez que o Agente de Recurso tenha obtido uma referência para o Serviço de Contexto, a *thread ProbeTask* periodicamente efetua a leitura do(s) sensore(s) associado(s) ao AR. O intervalo de atualização pode ser definido pelo projetista do AR, sendo que o intervalo padrão é de cinco segundos. Como esta informação pode ser importante para algumas classes de aplicações, o intervalo entre cada leitura é exposto na descrição do AR no momento de seu registro junto ao SRD e também é exposto pelo CS.

A Figura 6 ilustra o processo de atualização. A classe *ProbeTask* invoca o método

getResourceValues da classe *AbstractResourceAgent* e em seguida invoca o método *notifyContextService* desta classe. Caso tenha ocorrido alguma modificação nos valores lidos pelo AR a representação em XML do estado do AR será obtida através do método *getResourceState* e o método *notifyChangeState* do Serviço de Contexto será invocado. Desta forma o Serviço de Contexto é informado de forma pró-ativa pelo AR sobre as mudanças ocorridas no mesmo. Esta técnica é também chamada de método *push*, pois o AR "empurra" seu estado para o Serviço de Contexto. Como o método *getResourceState* é exposto pela interface *ResourceAgent* o Serviço de Contexto (ou qualquer outro cliente interessado) pode utilizá-la para recuperar o estado de um AR a qualquer instante sob demanda. Este método é denominado método *pull*, pois o cliente do AR "pega" a informação do mesmo.

Novamente fica evidenciada uma vantagem em se utilizar o *framework*, pois todas estas interações entre os componentes da infra-estrutura são efetuadas sem que os projetistas dos ARs tenham que elaborar uma única linha de código para tanto.

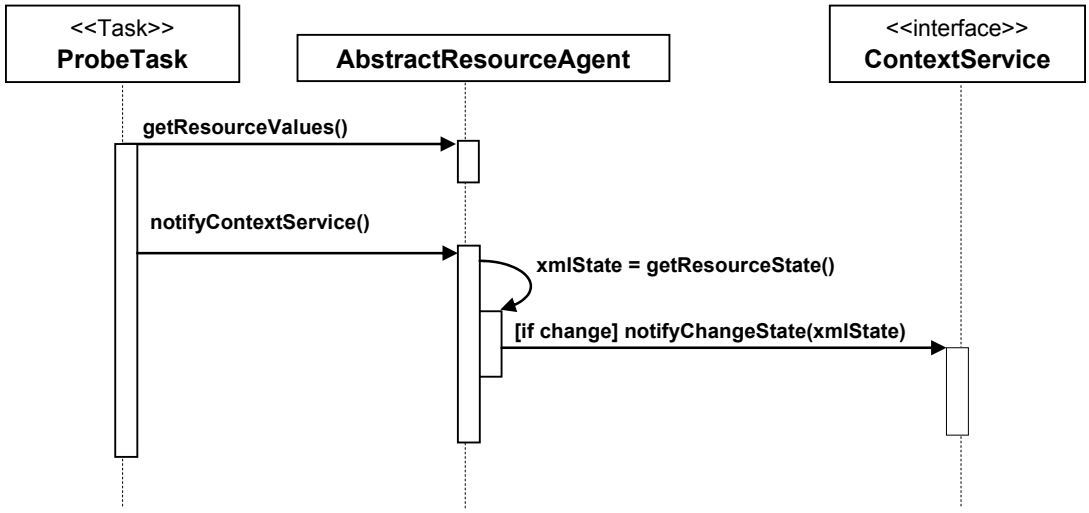


Figura 6 - Notificação de mudanças no estado de um AR

A seguir é apresentado um exemplo de especialização das classes do *framework* para a criação de um AR de processamento.

4.1.1 Exemplo de AR

A Figura 7 apresenta o modelo de classes para a criação de um AR para monitorar um nó de processamento Linux.

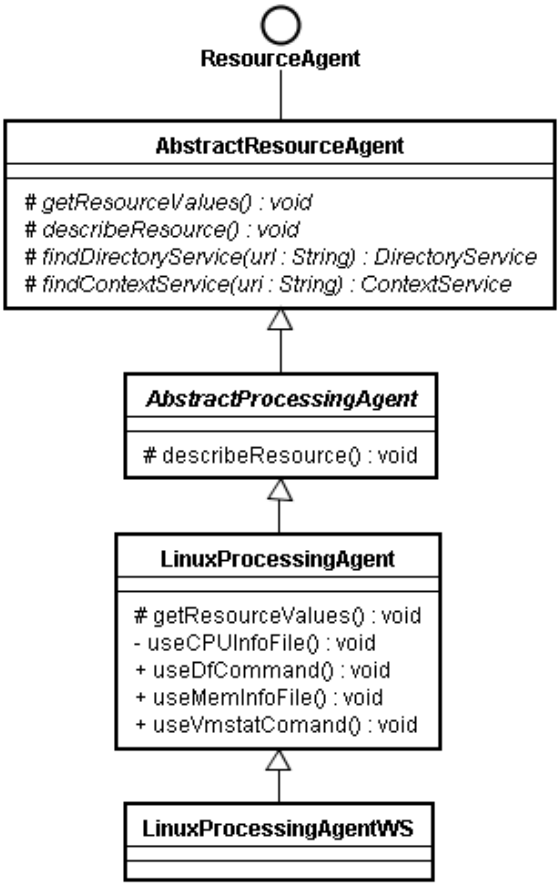


Figura 7 - Exemplo de especialização de AR para processamento

A abordagem para a arquitetura desta classe de recurso foi especializar a classe *AbstractResourceAgent* em outra classe (*AbstractProcessingAgent*) de forma que se possa usar a mesma no projeto de ARs para sistemas específicos. As funcionalidades consideradas genéricas para nodos de processamento ficam encapsuladas nesta classe (neste caso a descrição dos atributos dos recursos realizada pelo método *describeResource*) e as funcionalidades específicas de cada plataforma ficarão encapsuladas nas subclasses da classe *AbstractProcessingAgent*. No exemplo, a classe *LinuxProcessingAgent* efetivamente contém os métodos para monitorar o nó. Neste caso, o método *getResourceValues* invoca os métodos *use[*]* que utilizam as informações disponíveis nos arquivos do diretório */proc* do Linux e o comando *df* disponível nas distribuições GNU deste *kernel*.

Adicionalmente, nós ainda especializamos esta classe para customizar como o agente vai receber consultas e entregar as respostas (ou seja, qual a tecnologia de comunicação a ser utilizada). Em nossa implementação utilizamos Serviços Web (*LinuxProcessingAgentWS*). Esta classe fica responsável pelos detalhes envolvidos na criação dos *proxies* necessários à comunicação.

O método *describeResource* do AR *Processing* é parcialmente apresentado no Listagem 23. Neste método os atributos específicos para o agente são especificados. O projetista pode codificar a informação ou um *parser* pode ser provido de forma a se ler esta informação de um arquivo. Baseado neste método a descrição XML correspondente é gerada e registrada no SRD quando apropriado. Observe que no exemplo os atributos *CPU_CLOCK* (linha 4) e *MEMORY_FREE* (linha 10) são dinâmicos, portanto não podem ser "cacheados".

```
 1 protected void describeResource() {
 2 Resource resource = getResource();
 3 ResourceAttributeValue attrib=new ResourceAttributeValue();
 4 attrib.setName(CPU_CLOCK);
 5 attrib.setType("float");
 6 attrib.setUnits("MHz");
 7 attrib.setDynamic(true);
 8 resource.addAttribute(attribute);
 9 attrib = new ResourceAttributeValue();
10 attrib.setName(MEMORY_FREE);
11 attrib.setType("float");
12 attrib.setUnits("MB");
13 attrib.setDynamic(true);
14 resource.addAttribute(attribute);
15 [...]
16 }
```

Listagem 23 - Declarando atributos de um AR

Os pontos de extensão definidos pelo *framework* relacionados ao AR em si (sua descrição e a coleta dos dados junto aos sensores) são, portanto, definidos pelos métodos *describeResource* e *getResouceValues* respectivamente. Os pontos de extensão relacionados com a comunicação com a comunicação com os demais componentes da infra-estrutura são os métodos *findDirectoryService* e *findContextService*, respectivamente para o Serviço de Diretório e para o Serviço de Contexto. Na implementação de referência (baseada em Serviços Web) foi utilizada a API JAX/WS que facilita o desenvolvimento de Serviços Web e de consumidores destes serviços. Os ARs são clientes do Serviço de Registro e Diretório, pois necessitam efetuar o registro de suas descrições e localizações no mesmo. Com relação ao Serviço de Contexto, os ARs podem ser tanto clientes quanto servidores. No primeiro caso, os ARs enviam seus estados para o SC (método *push*) atuando como clientes do mesmo. O segundo caso fica caracterizado quando o SC consulta de forma síncrona os estados dos ARs (método *pull*).

A Listagem 24 apresenta o trecho de código utilizado na implementação de referência para definir o AR de processamento especializado para o sistema operacional Linux apresentado na Figura 7.

```
 1 @WebService(serviceName = "ProcessingAgent",
 2             targetNamespace = "http://monitoring.cdrf.edu.br",
 3             endpointInterface = "br.edu.cdrf.monitoring.ResourceAgent")
 4 public class LinuxProcessingAgentWS extends LinuxProcessingAgent {
 5   @Override
 6   protected DirectoryService findDirectoryService(String url){
 7     return WebServicesUtils.findDirectoryService(url);
 8   }
 9
10   @Override
11   protected ContextService findContextService(DirectoryService service,
12                                               String uri) {
13     return WebServicesUtils.findContextService(service, uri);
14   }
15 }
```

Listagem 24 - Exemplo de implementação de um AR como Serviço Web

A anotação *@WebService* presente na linha 1 acessível em tempo de execução indica que os objetos desta classe são Serviços Web. O atributo *serviceName* indica o nome que o serviço terá, o atributo *targetNamespace* define o espaço de nomes XML ao qual o serviço pertence e o atributo *endpointInterface* indica a interface que o serviço web deverá realizar, definindo assim as operações que o serviço disponibiliza para seus clientes. Neste caso, todos os métodos presentes nesta interface. Na linha 4 é especificado que a classe em questão especializa a classe *LinuxProcessingAgent* como ilustrado pela Figura 7.

Conforme discutido anteriormente, os métodos *findDirectoryService* e *findContextService* são utilizados para localizar os objetos que atuaram como intermediários (*proxies*) destes serviços junto ao AR, abstraindo do mesmo os detalhes de comunicação utilizados. Nossa implementação de referência disponibiliza uma classe utilitária denominada *WebServicesUtils* cuja função é disponibilizar métodos específicos para a comunicação com serviços web. Esta classe será discutida mais abaixo.

Para que um AR seja capaz de se comunicar como um Serviço Web é necessário apenas que seja criada uma classe seguindo o estilo apresentado na Listagem 24 e que uma classe inicializadora do serviço seja criada conforme a Listagem 25.

```
1  public class LinuxProcessingAgentStarter {
2    public static void main(String[] args) throws Exception {
3      LinuxProcessingAgentWS agent = new LinuxProcessingAgentWS();
4      File file = new File(System.getenv("CDRF_HOME"),
5                           "/conf/linuxProcessingAgent.xml");
6      WebServicesUtils.initialize(agent, file);
7      System.out.println("Linux Processing Agent is started.");
8    }
9  }
```

Listagem 25 - Exemplo de classe inicializadora de um AR

Na linha 3 um objeto da classe anotada como um serviço web é instanciado. Em

seguida o arquivo de configuração a ser utilizado é definido (linhas 4 e 5) e o AR é inicializado utilizando-se o método *initialize* da classe utilitária *WebServicesUtils* disponibilizada pela implementação referência (Listagem 26) passando-se para ele como parâmetros o arquivo de configuração e o AR.

```
 1 public class WebServicesUtils{
 2   public static void initialize(File file, ResourceAgent agent){
 3     agent.initialize(file);
 4     Endpoint endpoint = Endpoint.publish(agent.getResource().
 5                             getTecnologies().get("webservice"), agent);
 6   }
 7
 8   public static DirectoryService findDirectoryService(String url){
 9     URL directoryUrl = new URL(url);
10     QName serviceName = new QName("http://directory.cdrf.edu.br",
11                                   "DirectoryService");
12     Service service = Service.create(directoryUrl, serviceName);
13     DirectoryService directoryService =
14                                service.getPort(DirectoryService.class);
15   }
16
17   public ContextService findContextService(DirectoryService service,
18                                            String uri) {
19     Resource resource = findContextServiceByUri(service, uri);
20     URL contextUrl = new URL(resource.
21                                    getTecnologies().get("webservice"));
22     QName serviceName = new QName("http://context.cdrf.edu.br",
23                                   "ContextService");
24     Service service = Service.create(contextUrl, serviceName);
25     ContextService contextService =
26                                  service.getPort(ContextService.class);
27     return contextService;
28   }
29   [...]
30 }
```

Listagem 26 - Exemplo de implementação de um AR como Serviço Web

O método *initialize* da classe utilitária *WebServicesUtils* invoca o método *initialize* do AR (linha 3) onde o arquivo de configuração em formato XML é lido e processado. O processo de inicialização do AR se dá conforme ilustrado pelo diagrama de seqüência da Figura 5. Nesta figura se pode notar que são efetuadas chamadas aos métodos *findDirectoryService* e *findContextService*. Como estes métodos foram definidos na classe *LinuxProcessingAgentWS*, os mesmos definirão os *proxies* para o SRD e para o SC como sendo compatíveis com serviços web.

Na linha 12 da Listagem 26 a definição da classe do *proxy* para o SRD é criado através do método *create* da classe *Service* que pertence à API JAX/WS. Este método tem dois parâmetros: o primeiro é a URL onde a descrição do serviço se encontra (o documento WSDL do mesmo) e o segundo parâmetro é o nome do serviço. De posse destas duas informações (a

localização do WSDL e o nome do serviço) o método *create* utiliza instrumentação de *bytecode* (Lindholm, 1999) para criar dinamicamente o *proxy* para o serviço web. Finalmente, o método *getPort* (linhas 13 e 14) efetua a conversão do objeto do serviço para a interface *DirectoryService*.

A localização do *proxy* para o SC (método *findContextService* nas linhas 17 a 28) se dá de forma um pouco diferente. Como o SC é registrado no SRD, é necessário primeiro determinar a localização do mesmo através de seu URI. Isso é feito através do método *findContextServiceByUri* (detalhes de como são efetuadas as consultas ao SRD serão discutidos na Seção 4.2). Uma vez que o AR do SC seja localizado, se pode determinar a localização física do mesmo (linhas 20 e 21). Uma vez que a URL do SC seja conhecida a obtenção do *proxy* para o mesmo se dá de forma similar à do SRD discutida acima.

A seguir será descrita a implementação do Serviço de Registro e Diretório.

4.2 Implementação do Serviço de Registro e Diretório

As operações expostas pelo Serviço de Registro e Diretório são definidas pela interface *DirectoryService* apresentada na Listagem 27.

```
 1  public interface DirectoryService {
 2   String typeRegister(String xml);
 3   String removeType(String xml);
 4   String resourceRegister(String xml);
 5   String removeResource(String xml);
 6   String listTypes(String xml);
 7   String findResourceByURI(String xml);
 8   String findResourcesByType(String xml);
 9   String findResourcesByLocalization(String xml);
10 }
```

Listagem 27 - Interface do Serviço de Registro e Diretório

Como todos os demais componentes da infra-estrutura, todos os métodos recebem como parâmetro documentos XML contendo as mensagens a serem trocadas com o SRD.

O método *typeRegister* (linha 2) é responsável pelo registro de um tipo de recurso. O XML passado como parâmetro é o mesmo apresentado na Seção 4.2. O tipo será utilizado para validar a descrição que um AR poderá informar ao SRD.

O método *resourceRegister* (linha 4) é responsável pelo registro de um AR. Quando um AR é inicializado o mesmo pode enviar sua descrição ao SRD através deste método. Na implementação de referência é possível também armazenar a descrição dos ARs de uma

instalação física em um diretório padronizado pela infra-estrutura e, quando o SRD é inicializado, as definições são lidas e armazenadas internamente.

As contrapartes dos métodos *typeRegister* e *resourceRegister* são os métodos *removeType* (linha 3) e *removeRegister* (linha 5). A representação em XML da mensagem trocada utilizada nestes métodos é a mesma a ser utilizada nos demais métodos desta interface e será apresentada mais abaixo.

O método *listTypes* (linha 6) é responsável por fornecer as descrições dos tipos de recursos armazenadas em um SRD. Será retornada uma mensagem XML contendo as descrições armazenadas que correspondam aos critérios informados no XML utilizado como parâmetro.

Os demais métodos são métodos utilizados para efetuar consultas sobre os ARs registrados em um SRD. Pode-se localizar um AR através de seu URI (método *findResourceByURI*), ou localizar ARs de um determinado tipo/supertipo (método *findResourcesByType*), ou ainda localizar os ARs que estejam em uma determinada localização (*findResourcesByLocation*).

O XSD dos documentos XMLs utilizados nas consultas e nos processos de remoção de tipos e de ARs é o apresentado na Listagem 28.

```
 1 <xsd:element name="DirectoryQuery">
 2   <xsd:complexType>
 3     <xsd:all>
 4       <xsd:element name="URI" type="xsd:anyURI" minOccurs="0" />
 5         <xsd:element name="Type" minOccurs="0">
 6           <xsd:complexType>
 7             <xsd:attribute name="Value" type="xsd:string"
 8                            use="required" />
 9             <xsd:attribute name="Strict" type="xsd:boolean"
10                            default="true" />
11           </xsd:complexType>
12         </xsd:element>
13         <xsd:element name="SuperType" type="xsd:string" minOccurs="0" />
14         <xsd:element name="IP" type="xsd:string" minOccurs="0" />
15         <xsd:element name="IncludeFederation" type="xsd:boolean"
16                      minOccurs="0" />
17       </xsd:all>
18     </xsd:complexType>
19   </xsd:element>
20 </xsd:element>
```

Listagem 28 - XSD das consultas ao SRD

O elemento *DirectoryQuery* (linha 1) indica que se trata de uma consulta ao SRD, sendo que o elemento URI (linha 4) indica que se deseja localizar um recurso através de seu URI (utilizado pelo método *findResourceByURI* e pelo método *removeResource*).

O elemento *Type* informa contém os atributos *Value* que conterá o nome do tipo que se deseja consultar (o valor "*" indica que se deseja retornar todos os tipos) e o atributo *Strict* que informa se a consulta ao SRD sendo executada deverá ser limitada apenas ao tipo (valor igual à *true*) ou também aos seus subtipos (valor igual à *false*). O elemento *SuperType* é utilizado para definir um supertipo específico. Estes elementos são utilizados pelos métodos *listTypes*, *removeType* e *findResourcesByType*.

O elemento *IP* está relacionado com a localização física do AR. Ou seja, através dele se especifica o endereço IP da máquina (ou o nome da mesma) onde o AR se encontra instalado. Este elemento é utilizado pelo método *findResourceByLocation.*

A infra-estrutura disponibiliza a classe *AbstractDirectoryService* que implementa todos os métodos presentes na interface *DirectoryService* e expõe alguns métodos abstratos a serem implementados pelos projetistas de Serviços de Registro e Diretório. Ou seja, através do *framework* é possível a construção de SRDs que efetuam o registro das descrições dos tipos e recursos em bancos de dados relacionais, LDAP ou outro mecanismo de armazenamento permanente, e que se comuniquem com os demais elementos da infra-estrutura através de diferentes protocolos de comunicação em rede, bastando para isso especializar a classe *AbstractDirectoryService* através da implementação dos pontos de extensão definidos por esta classe.

Os métodos declarados na interface *DirectoryService* são implementados pela classe *AbstractDirectoryService* e fazem uso de classes utilitárias que efetuam o *parser* dos documentos XML passados como parâmetro e delegam para métodos da classe que possuem o mesmo nome dos métodos presentes na interface *DirectoryService*, mas que recebem como parâmetro objetos Java que representam os dados passados através dos documentos XML. Assim, mantém-se a neutralidade do XML com relação à linguagem na qual o serviço é construído. Esta abordagem é utilizada em todos os elementos da infra-estrutura (ARs, SRD, SC e SD). Caso não se deseje utilizar a implementação de referência destes serviços (implementando-se um mais destes elementos em outra linguagem) os componentes ainda poderão inter-operar porque cada elemento deverá obedecer aos formatos das mensagens XML definidas. Desde que os serviços se comuniquem através da mesma tecnologia de rede (Serviços Web, por exemplo).

A Figura 8 apresenta um diagrama de classes contendo as classes disponibilizadas na implementação de referência relacionadas com a implementação de um SRD.

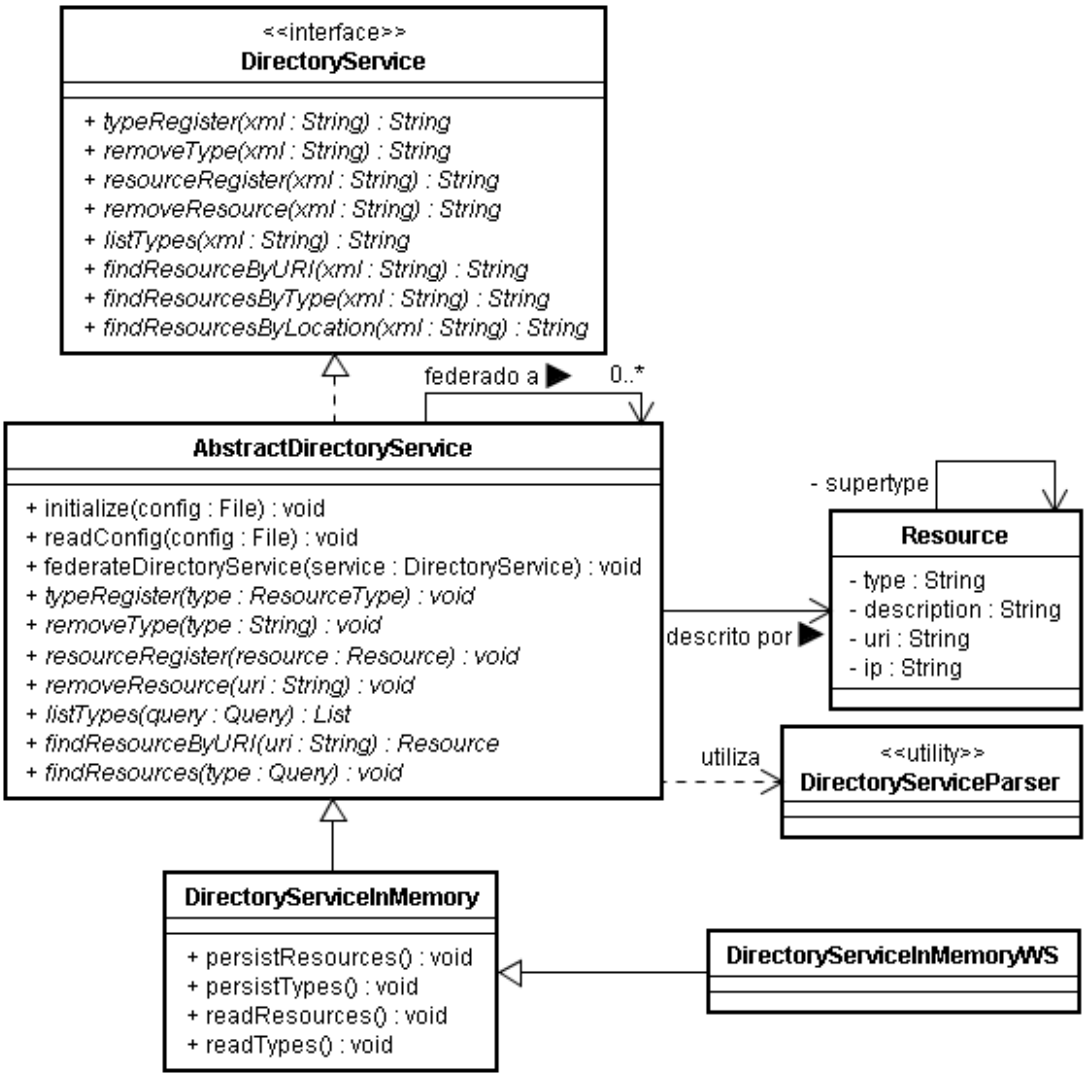


Figura 8 - Diagrama de classes da implementação de referência do SRD

No diagrama identificamos primeiro a interface *DirectoryService* e seus métodos. A classe *AbstractDirectoryService*, que implementa esta interface, contém a implementação de cada método da mesma. Cada um destes métodos tem um parâmetro do tipo *string* cujo valor é a representação da mensagem XML correspondente a cada operação que os métodos realizam. Os métodos *typeRegister*, *removeType*, *resourceRegister*, *removeRegister*, *listTypes* e *findResourceByURI* foram construídos da seguinte forma: o *parser* da mensagem em XML passada como parâmetro é realizado e os dados presentes na mensagem são convertidos em objetos Java. Em seguida, (por questão de clareza do código) é invocado um método abstrato cujo nome é igual ao método original, mas tendo como parâmetro o objeto Java (diagrama de classes da Figura 8). Os métodos *findResourcesByType* e *findResourcesByLocation* também fazem o *parser* da mensagem XML e convertem a convertem em objetos Java, mas não

delegam para um método com o mesmo nome, mas para o método mais genérico *findResource* que recebe como parâmetro um objeto do tipo *Query* que representa os parâmetros que podem ser utilizados nas consultas junto ao SRD.

Portanto, estes sete métodos abstratos é que realmente serão os responsáveis por realizarem as operações junto ao SRD. Sendo assim constituem pontos de extensão para a implementação de SRDs que fazem uso de diferentes tecnologias para armazenar as representações dos recursos e seus tipos. Na implementação de referência, construímos um SRD que armazena seus dados em tabelas *hash* conforme será abordado mais abaixo, mas poderia ser construído um SRD que registraria e consultaria seus dados em um servidor LDAP, bastando para isso implementar os sete métodos abstratos da classe *AbstractDirectoryService*.

O SRD possui também o método *initialize* que é responsável por inicializar o SRD de forma similar à inicialização de ARs. Este método invoca o método *readConfig*, que é responsável por ler o arquivo XML contendo a configuração do SRD. Este arquivo contém a localização de outros SRDs com os quais o SRD que está inicializando deverá se associar em uma federação. Isso é feito pelo método *federateDirectoryService* que é invocado pelo método *initialize* logo após a leitura do arquivo de configuração.

A implementação de referência do SRD armazena as descrições dos tipos e recursos em tabelas *hash* mantidas em memória. Periodicamente, o SRD armazena estas tabelas em disco apenas para que os registros não tenham que ser realizados pelos ARs a cada inicialização do SRD. Todas as consultas ao repositório de meta-nível mantido pelo SRD da implementação de referência são realizadas em memória. Os métodos prefixos "*persist*" e "*read*" são utilizados para armazenar e retornar respectivamente do disco as tabelas *hash*.

Caso alguém deseje implementar um SRD que utiliza um banco de dados relacional para armazenar as definições de tipos e de recursos e então efetuar consultas, deverá especializar a classe *AbstractDirectoryService* de forma similar à apresentada no diagrama da Figura 7.

A classe *DirectoryServiceInMemoryWS* é uma especialização que expõe como Serviços Web o SRD com tabelas *hash* da implementação de referência. Este processo é realizado através da API JAX/WS da mesma forma que a descrita na Seção 4.1.

A seguir é apresentada a implementação de referência do Serviço de Contexto.

4.3 Implementação do Serviço de Contexto

O Serviço de Contexto está estruturado conforme o diagrama de classes apresentado na Figura 9. O método *query* é utilizado nas consultas síncronas ao SC (método *pull*) conforme descrito na Seção 3.3.3.1. O parâmetro declarado no método é a representação da mensagem XML de consulta ao SC (Listagem 10) e o valor de retorno é a representação XML da resposta (Listagem 12).

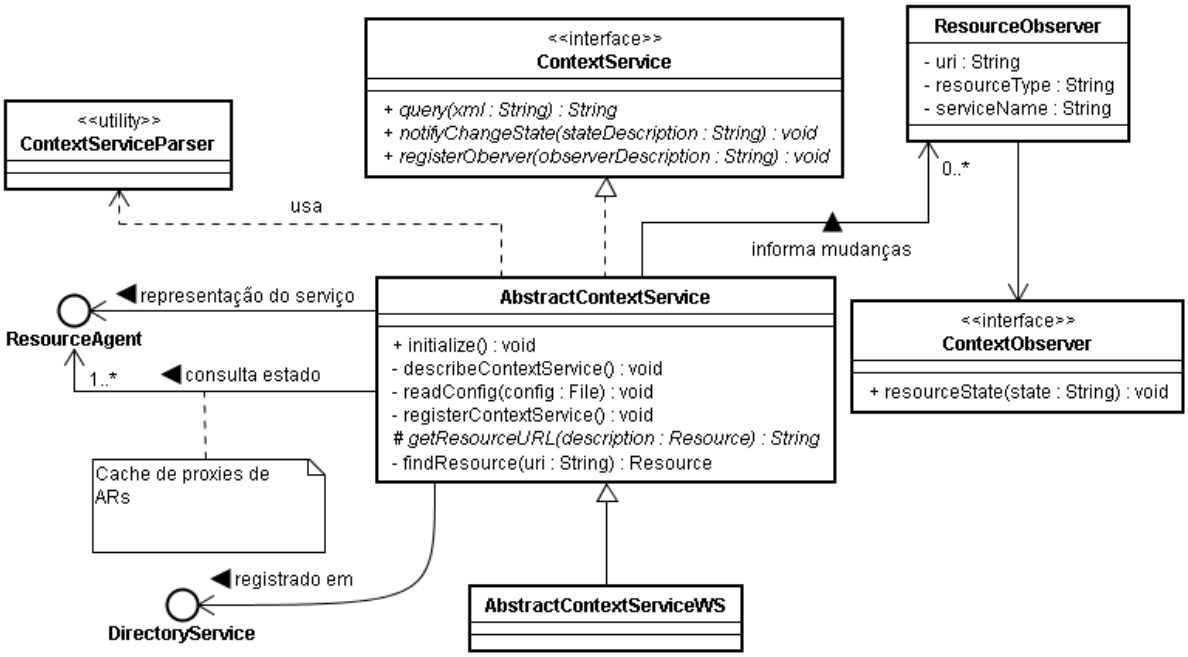


Figura 9 - Diagrama de classes da implementação de referência do SC

O método *notifyChangeState* é invocado pelos ARs quando os mesmos detectam mudanças nos valores monitorados (Figura 6). É através deste método que o SC é informado de forma assíncrona sobre mudanças nos valores monitorados por um AR. O parâmetro declarado neste método é a representação XML descrito na Listagem 16 e que contém a descrição do estado interno do AR.

O método *registerObserver* é o método a ser utilizado pelos clientes que desejam ser informados de forma assíncrona sobre mudanças no contexto. O parâmetro declarado no método é o documento XML descrito na Listagem 14 e que contém o tipo do recurso sobre o qual o cliente deseja ser informado sobre mudanças no contexto. Isso deve ser utilizado se o cliente não conhece o URI de um determinado AR ou se ele deseja ser informado sobre o estado de um conjunto de ARs que correspondam ao tipo especificado. O parâmetro pode ainda conter o URI de um AR específico. Além de um destes dois parâmetros (os mesmos são

mutuamente exclusivos) a representação da mensagem em XML contém também o URL do cliente que servirá como função de *callback* para que o SC possa informá-lo quando mudanças ocorrerem nos ARs que são do seu interesse.

A implementação de referência disponibiliza a classe *AbstractContextService* que fornece um ponto de extensão a partir do qual outros SCs podem ser criados. O funcionamento do método *initialize* é similar ao dos métodos de mesmo nome presentes no SRD e nos ARs. Este método invoca o método privado *readConfig* que lê a configuração do mesmo a partir de um arquivo XML. Em seguida o SC localiza o(s) Serviço(s) de Registro e Diretório que estão discriminados no arquivo de configuração e efetua seu registro (método privado *registerContextService*) em cada um deles informando sua representação (vide a classe *ResourceAgent* no diagrama de classes da Figura 9) da mesma forma que os ARs fazem. Assim, o SC fica disponível para uso por parte de clientes que utilizam diferentes SRDs, aumentando seu escopo de utilização.

Quando um cliente se registra para receber notificações sobre mudanças no contexto através do método *registerObserver* ele informa seu URI. O SC utiliza o método *findResource* para determinar a descrição do cliente (ele mesmo deve ter sua descrição registrada no SRD). Uma vez que o SC obtém a descrição do cliente (uma instância do objeto *Resource*), o SC é capaz de determinar a localização do mesmo para se comunicar de volta (*callback*) com o cliente quando mudanças no contexto ocorrerem. Os dados sobre cada observador que efetua seu registro são encapsulados em um objeto da classe *ResourceObserver* que é posteriormente armazenada num *array* interno do SC para referência futura. Quando uma mudança no contexto ocorre o SC consulta este array e informa o observador (o cliente). Para que o cliente possa ser informado pelo SC, o mesmo terá que implementar a interface *ContextObserver*. Esta última tem apenas um método (*resourceState*) que tem como parâmetro a representação em XML do estado do AR conforme descrito na Listagem 16.

Finalmente, quando uma consulta síncrona é realizada através do método *query*, os ARs envolvidos na consulta têm que ser consultados de forma de síncrona pelo SC para que ele obtenha seus estados. Para isso, o SC consulta o SRD conforme descrito na Seção 3.3.3.1. Como a construção de *proxies* é um processo que demanda tempo e recursos computacionais, o SC armazena estes *proxies* em um *cache* para que o desempenho de consultas posteriores seja maximizado.

A seção seguinte apresenta a implementação do Serviço de Descoberta.

4.4 Implementação do Serviço de Descoberta

As classes envolvidas na implementação do Serviço de Descoberta estão contidas no diagrama de classes da Figura 10.

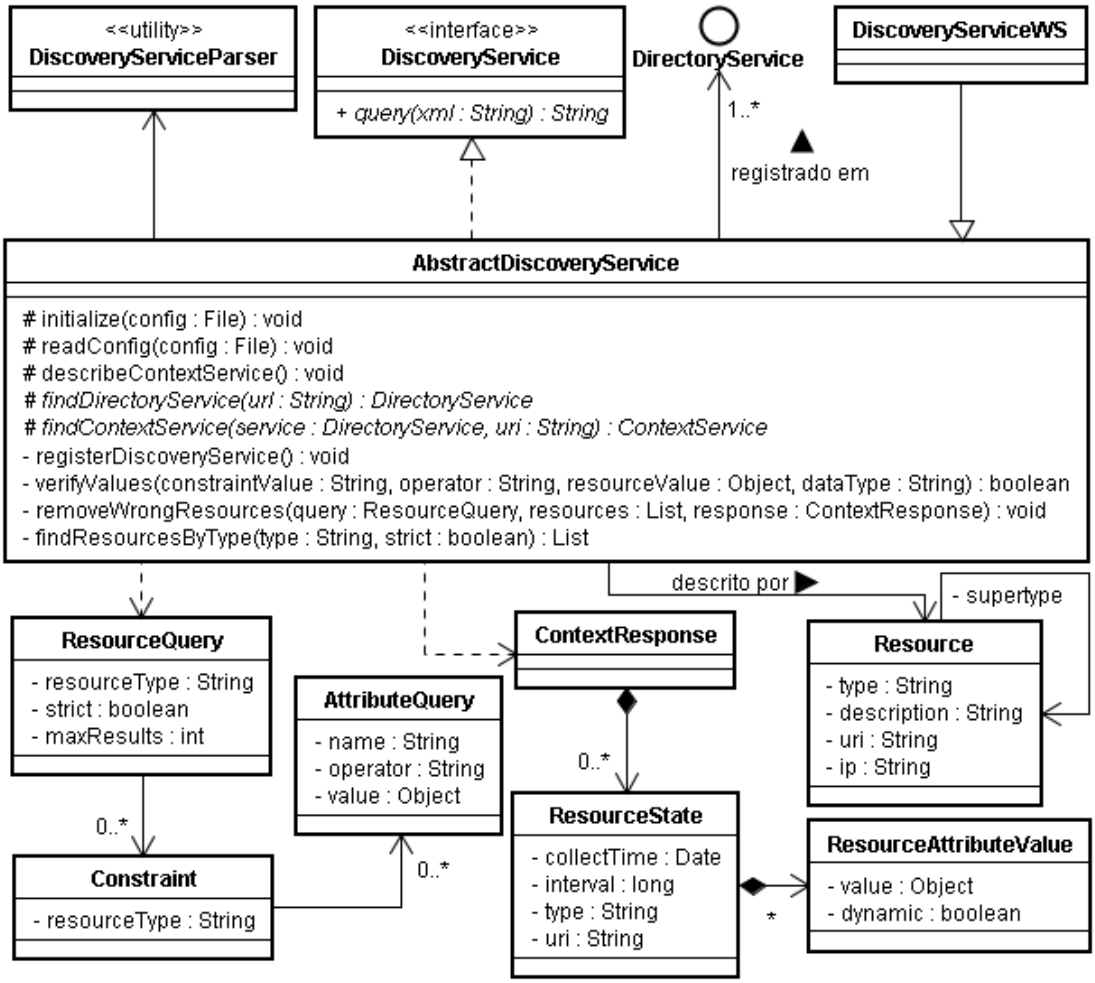


Figura 10 - Classes envolvidas na implementação do SD

A interface *DiscoveryService* contém o método *query*, a partir do qual as consultas são realizadas ao Serviço de Descoberta. Este método possui um parâmetro cujo valor deverá ser a representação em XML da consulta a ser realizada de acordo com o esquema apresentado na Listagem 18 e que contenha os critérios da consulta a ser realizada. O método deverá retornar a representação da resposta em XML respeitando o esquema apresentado na Listagem 20.

A classe *AbstractDiscoveryService* disponibilizada pela implementação de referência possui o método *initialize* que recebe como parâmetro um objeto do tipo *File* que representa o arquivo de configuração em formato XML (detalhes podem ser consultados em (Rodrigues, 2009c)). Este arquivo, a exemplo dos demais serviços do *framework*, contém os parâmetros

de inicialização do serviço. O processo de inicialização do Serviço de Descoberta é similar ao processo de inicialização dos demais serviços. Ou seja, o arquivo de configuração é lido (*readConfig*), o serviço cria um objeto *Resource* que conterá sua descrição (*describeContextService*), irá localizar o serviço de diretório (*findDirectoryService*), irá realizar seu registro junto ao Serviço de Registro e Diretório (*registerDiscoveryService*) com base em sua URL e localizará o serviço de contexto (*findContextService*) com base em seu URI. A URL do SRD e o URI do SC são obtidos do arquivo de configuração do SD. O arquivo de configuração pode conter uma ou mais URLs de SRD, permitindo ao SC se registrar em mais de um SRD e ficar disponível para uma gama maior de clientes potenciais. De forma similar, as consultas que o SD efetuará para obter dados sobre o contexto dos recursos, poderão ser realizadas em mais de um SC, pois no arquivo de configuração poderá existir uma ou mais referências a SCs (URIs).

A classe *DiscoveryServiceParser* é uma classe utilitária que realiza o *parser* das representações XML envolvidas nas consultas realizadas ao SD e nas respostas às consultas que o SD efetua ao SC.

O diagrama de classes apresenta ainda as classes *ResourceQuery* e *ContextResponse* que representam os critérios que constituem as consultas realizadas ao SD (esquema XML apresentado na Listagem 18) e as respostas às consultas realizadas ao SC (esquema XML apresentado na Listagem 12) respectivamente.

A classe *ResourceQuery* possui o atributo *resourceType* que representa o tipo do recurso sendo consultado, o atributo *strict* que indica se a consulta deverá levar ou não em consideração também os subtipos e possui ainda o atributo *maxResults* que representa o número máximo de recursos que deverão ser retornados como resposta à consulta.

Objetos da classe *ResourceQuery* podem conter zero ou mais instâncias da classe *Constraint*. Esta classe representa as restrições de contexto as quais os recursos deverão obedecer. Cada objeto deste tipo possui um atributo *resourceType* que armazena o tipo do recurso e um ou mais objetos do tipo *AttributeQuery* que representa os atributos do recurso e as restrições que os mesmos deverão obedecer. Esta classe possui o atributo *name* que é o nome da propriedade do recurso, o atributo *operator* que é o operador lógico a ser aplicado sobre a propriedade (maior, menor, diferente, etc.) e o atributo *value* que representa o valor a ser comparado à propriedade do recurso.

A classe *ContextResponse* é composta por uma coleção de objetos do tipo

ResourceState. Esta classe representa o estado do recurso cujas propriedades obedecem às restrições passadas na consulta realizada ao SC. Esta classe possui os atributos *collectTime* (a hora em que os valores foram coletados pelo AR), *interval* (o intervalo entre cada medição), *type* (o tipo da propriedade) e *uri* (o URI do recurso). Os objetos da classe *ResourceState* possuem ainda uma coleção de objetos do tipo *ResourceAttributeValue* que representa os valores que as propriedades do recurso possuíam no momento em que a medição das mesmas foi realizada. Os atributos presentes nesta classe são: o valor (propriedade *value*) e a indicação se os valores desta propriedade são ou não dinâmicos (propriedade *dynamic*).

O diagrama de seqüência da Figura 11 apresenta a seqüência de métodos invocados na execução do método *query* do SD.

No diagrama, o método *query* (passo 1) do SD é invocado pelo cliente do serviço. O SC invoca o método *perseQuey* (passo 1.1) da classe utilitária *DiscoveryServiceParser* que retorna uma instância da classe *ResourceQuery* que contém a representação da consulta. De posse da representação da consulta em um objeto, o SD executa uma consulta (passo 1.2) ao SRD baseada no tipo do recurso (método *findResourcesByType*). O SRD retorna uma representação em XML dos recursos que são do tipo especificado e o SD delega à classe *DiscoveryServiceParser* o processamento do documento XML obtido.

A lista de recursos construída por esta etapa é percorrida seqüencialmente e o SD identifica quais propriedades do recurso sendo analisado são dinâmicas e quais não (sofrem variações ao longo do tempo ou não). O SD identifica as propriedades que são estáticas e que estão presentes na consulta na forma de restrições. Para cada propriedade que se encaixa nesta condição, o SD verifica se as restrições são obedecidas ou não. Em caso negativo, o SD elimina o recurso da lista retornada pelo SRD.

Após o teste das propriedades estáticas, o SD passa para a validação das propriedades dinâmicas. Para isso, monta a representação em XML da consulta a ser realizada ao SC através do uso da classe *DiscoveryServiceParser* (passo 1.4) e executa uma consulta ao SC (passo 1.5). Após o *parsing* da resposta (passo 1.6), o SD invoca o método *removeWrongResources*. Este método é responsável por remover aqueles recursos cujas propriedades não obedecem aos critérios especificados na consulta ao SD. Finalmente, a resposta em formato XML é montada com a ajuda da classe *DiscoveryServiceParser* (não representado no diagrama) e retorna a resposta ao cliente.

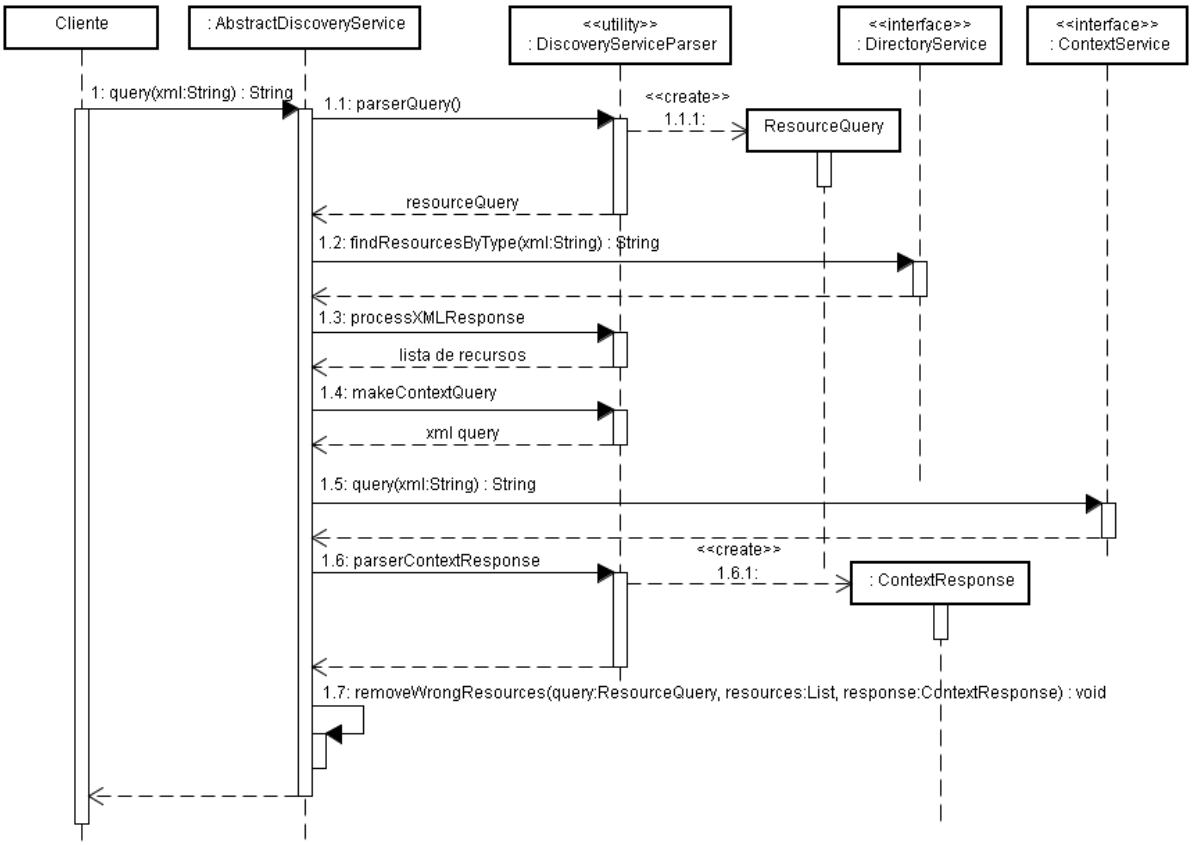


Figura 11 - Diagrama de seqüência do método query do SD

A classe *AbstractDiscoveryService*, presente na implementação padrão, disponibiliza todas as funcionalidades necessárias para a execução de um Serviço de Descoberta. Os únicos pontos de extensão que ficam a cargo dos desenvolvedores destes serviços, caso seja necessário, são os métodos *findDirectoryService* e *findContextService*. Estes dois métodos são responsáveis por configurar os *proxies* para estes dois serviços. Na implementação padrão é disponibilizada uma implementação deste serviço na forma de um Serviço Web (classe *DiscoveryServiceWS*) através da API JAX/WS que foi implementada de forma similar à classe *LinuxProcessingAgenteWS* discutida na Seção 3.1.1.

4.5 Utilização dos serviços

O uso e a integração da implementação de referência em qualquer aplicação são simples e orientados a Serviços Web. Os clientes podem utilizar a API JAX/WS para construir *proxies* dinâmicos que irão efetivar a comunicação com os serviços da infra-estrutura (Serviço de Registro e Diretório, Serviço de Contexto e Serviço de Descoberta). Como a implementação de referência foi construída utilizando-se Serviços Web, nada impede

que um cliente ou um novo AR sejam construídos utilizando-se outras plataformas de desenvolvimento tais como a baseada na linguagem de programação C# da Microsoft. Esta interoperabilidade é possível graças aos padrões mantidos pelo W3C e que fazem parte do cerne dos Serviços Web.

Os *proxies* utilizados para efetuar a interação com os serviços da infra-estrutura são criados a partir das descrições dos serviços (WSDL) e que são disponibilizadas de forma transparente através das URLs dos serviços.

No caso do SRD que é o ponto inicial de interação com os elementos da infra-estrutura, sua URL tem que ser conhecida pelos clientes sendo desenvolvidos. É através da URL do SRD que o WSDL deste serviço é acessado. As URLs dos demais serviços e dos ARs da infra-estrutura estão presentes em suas descrições e são utilizadas no processo de registro dos mesmos junto ao SRD. Uma vez que os serviços são registrados no SRD suas descrições se tornam disponíveis para os clientes através da interface de consulta do SRD. Ou seja, os clientes podem localizar os Serviços de Contexto e de Descoberta registrados no SRD obtendo as descrições dos mesmos. Nestas descrições em XML encontra-se o elemento *Tecnologies* que contém a relação de tecnologias (elemento *Tecnology*) com os quais o serviço é capaz de lidar. Este elemento possui o atributo *Type* utilizado para armazenar o tipo de tecnologia (“webservice", “rmi”, “soquets”, “rpc”, etc.) e a URL para acessar o mesmo (atributo URL). No caso de Serviços Web, a URL contém a localização do WSDL do mesmo. Assim o cliente é capaz de criar os *proxies* para interagir com os serviços. No caso de se utilizar RMI, o atributo URL contém o nome que o serviço possui no serviço de registro desta tecnologia. Uma observação importante a fazer é que uma implementação de um serviço pode utilizar uma ou mais tecnologias para se comunicar (nada impede de se ter um serviço que se comunique utilizando RMI ou Serviços Web). A decisão de qual, ou quais, tecnologias a serem utilizadas cabe ao projetista da implementação do serviço.

O trecho de código da Listagem 29 apresenta um exemplo de uma consulta ao SRD efetuada por intermédio da linguagem Java utilizando a API JAX/WS. Primeiro, é especificado o nome do serviço totalmente qualificado (linhas 1 e 2). Este nome é definido pelo espaço de nomes ao qual o serviço pertence (http://directory.cdrf.edu.br) e o nome do serviço (neste caso *DirectoryService*).

Em seguida é especificada a URL do serviço onde o WSDL se encontra (linha 3). No exemplo, o serviço se encontra na máquina *leme* e porta 1974. Depois a representação local do serviço é construída (linha 4) através de instrumentação de *byte code* e o *proxy* a ser usado

é obtido (linha 5). Neste momento os métodos do SRD podem ser invocados transparentemente. O último passo é a consulta efetivamente. No exemplo, é efetuada uma consulta aos tipos registrados no SRD. O parâmetro *XML* contém a consulta conforme o esquema XML apresentado na Listagem 8. A resposta retornada em formato XML é então armazenada na variável *responseXML*.

```
1  private static final QName SR = new Name("http://directory.cdrf.edu.br",
2                                           "DirectoryService");
3  URL url = new URL("http://leme:1974/directory/DirectoryService?wsdl");
4  Service srv =  Service.create(url, SR);
5  DirectoryService service = srv.getPort(DirectoryService.class);
6  String responseXML = service.listTypes(xml);
```

Listagem 29 - Exemplo de consulta realizada ao SRD

Outro aspecto de integração é o uso da informação *Type* e *Units* presentes na descrição dos recursos. Quando as classes abstratas dos ARs são especializadas para a implementação de um AR específico, estas informações podem ser usadas para permitir a associação ou casamento de tipos e unidades diferentes daquelas especificadas quando o recurso foi registrado no SRD. A idéia é que o SRD e o SC podem realizar as conversões necessárias em tempo de execução. Um possível aprimoramento poderia estender o modelo de descrição de recursos de forma a empregar ontologias para guiar conversões e casamento de tipos. Isso possibilitará o uso de recursos com diferentes descrições (digamos, fornecidas por diferentes provedores), mas semanticamente equivalentes.

4.6 Integração com os serviços da infra-estrutura

Como prova de conceito foi desenvolvida uma interface administrativa através da qual os serviços podem ser consultados. A interface foi dividida em três abas, uma para cada um dos serviços (SRD, SC e SD) que são inicializados juntamente com o sistema operacional das máquinas do laboratório nas quais os mesmos se encontram. Nesta interface os serviços são explorados e exercitados através de descrições de meta-nível para exibir as informações referentes aos RAs e guiar as consultas ao CS e DS.

A Figura 12 apresenta o resultado de uma consulta realizada ao SRD através da interface administrativa. Neste exemplo, a descrição do SRD está localizada na URL http://leme.pel.uerj.br:1974/directory/DirectoryService?wsdl. A tela apresenta os tipos de recursos registrados no SRD. Na figura podem ser vistos os tipos *ContextService*, *DiscoveryService* e *RegistryDirectoryService* que são os tipos dos ARs correspondentes aos

serviços da infra-estrutura (Serviço de Contexto, Serviço de Descoberta e Serviço de Registro e Diretório).

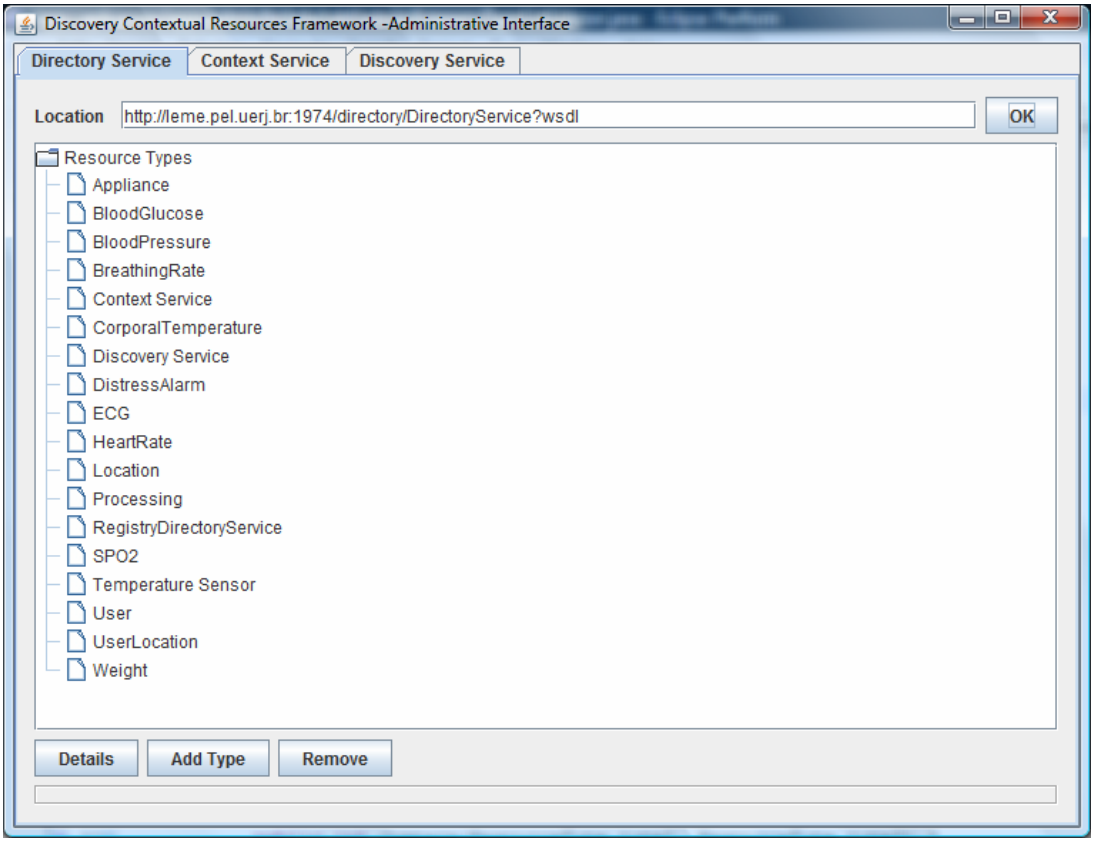


Figura 12 - Interface administrativa, consulta ao SRD

Quando se seleciona um nodo da árvore através de um duplo *click* são apresentados os ARs do tipo registrados no SRD. Conforme ilustrado na Figura 13. Na figura são apresentados dois ARs do tipo *Processing* que se encontram registrados no Serviço de Registro e Diretório: um com o URI *leme.pel.uerj.br/ProcessingAgent* e outro com o URI *ipanema.pel.uerj.br/ProcessingAgent*. Através da seleção de um desses nodos e subseqüente pressionamento do botão *Details* é apresentada na tela da Figura 14. Nesta tela são apresentados os atributos do AR selecionado. No exemplo desta figura, são apresentados os atributos do AR do tipo *Processing* identificado pelo URI leme.pel.uerj.br/ProcessingAgent cuja localização física é leme.pel.uerj.br. Um detalhe a ser notado é que este AR é capaz de se comunicar utilizando Serviços Web e RMI conforme demonstra a tabela que se encontra na parte inferior da figura.

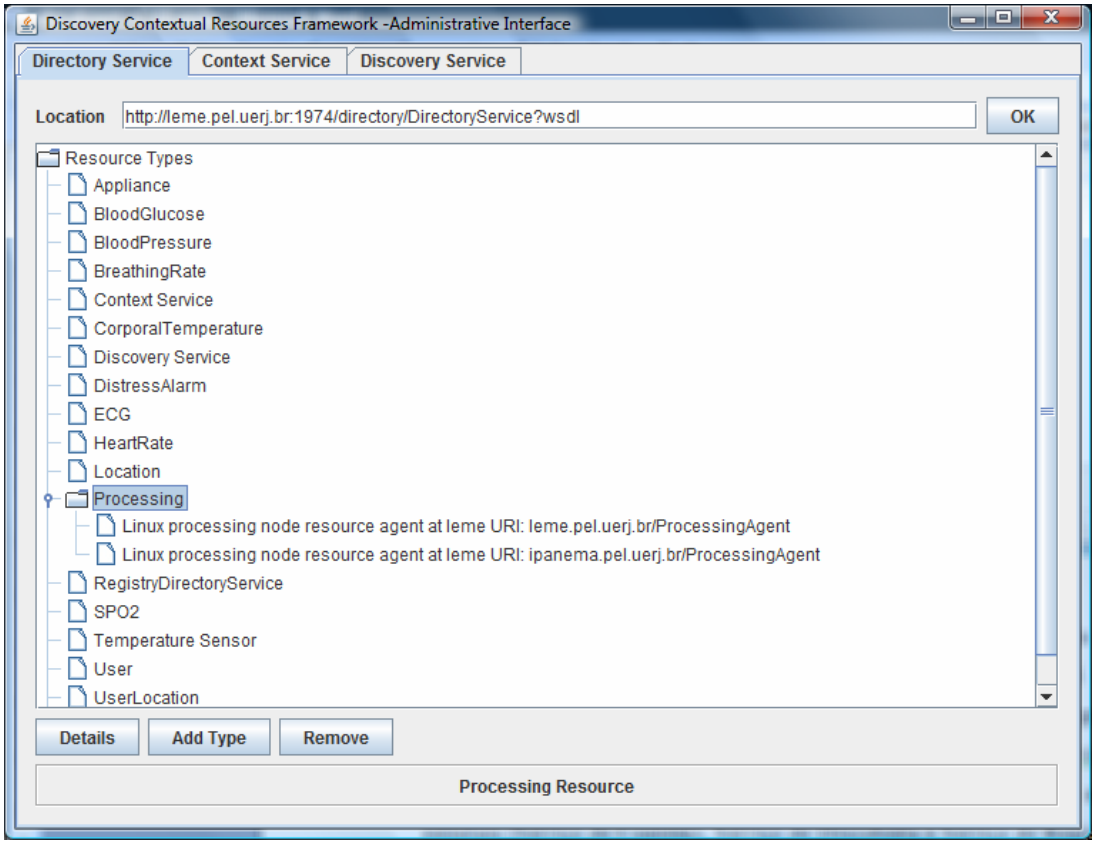


Figura 13 - Lista de ARs do tipo processing através da interface administrativa

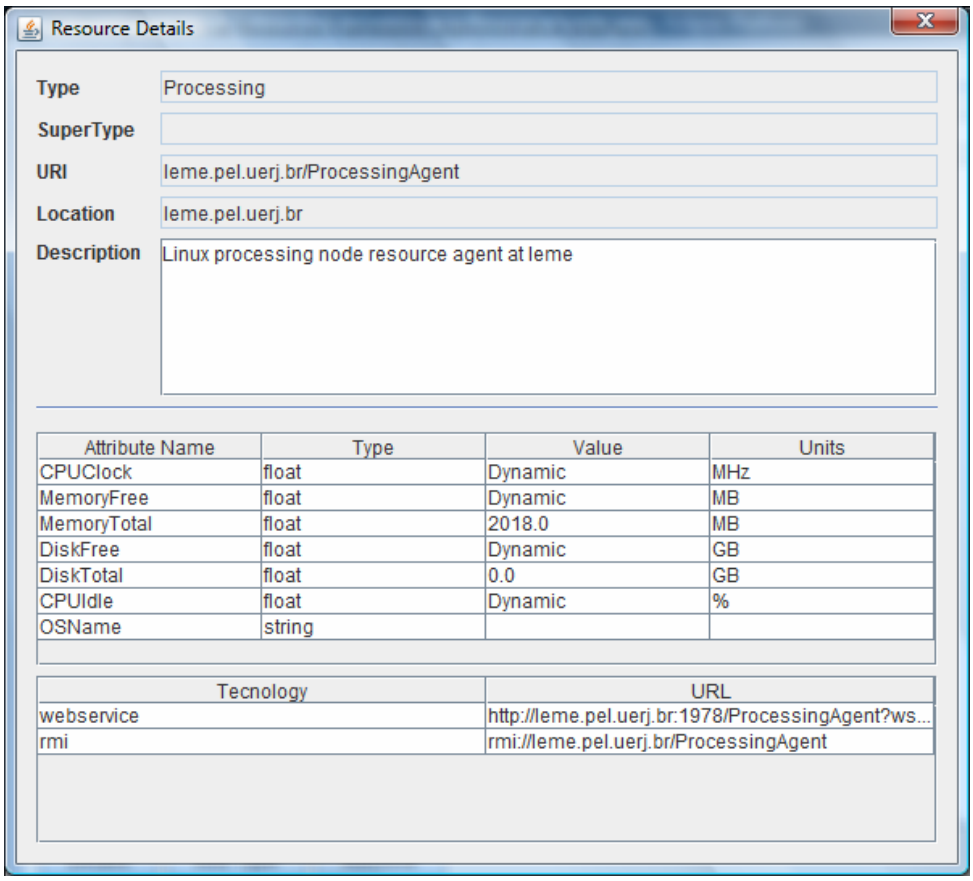


Figura 14 - Descrição de um Agente de Recurso através da Interface Administrativa

Através da interface administrativa é possível executar consultas assíncronas ao Serviço de Contexto baseadas no tipo do recurso. A Figura 15 apresenta os critérios utilizados na consulta. Na tela são informados a localização do Serviço de Contexto (que poderia ser obtida de uma consulta ao SRD) e o tipo do recurso que se deseja obter informações. Em seguida deve-se pressionar o botão *"Query"* para se efetuar a consulta.

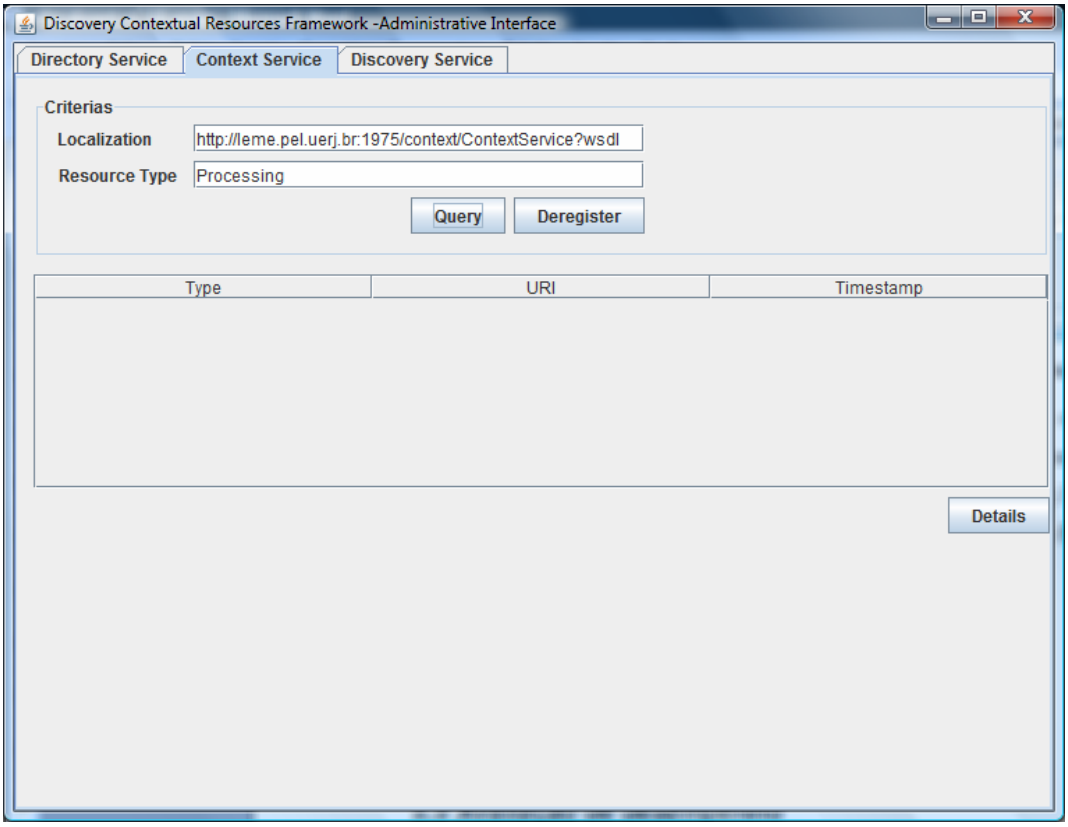


Figura 15 - Consultas assíncronas através da Interface Administrativa

A Figura 16 apresenta o resultado de uma consulta assíncrona realizada ao Serviço de Contexto localizado na URL *http://leme.pel.uerj.br:1975/context/ContextService?wsdl*. A cada mudança ocorrida nos recursos do tipo *Processing* a interface administrativa é informada e um novo registro é inserido na tabela apresentada na figura.

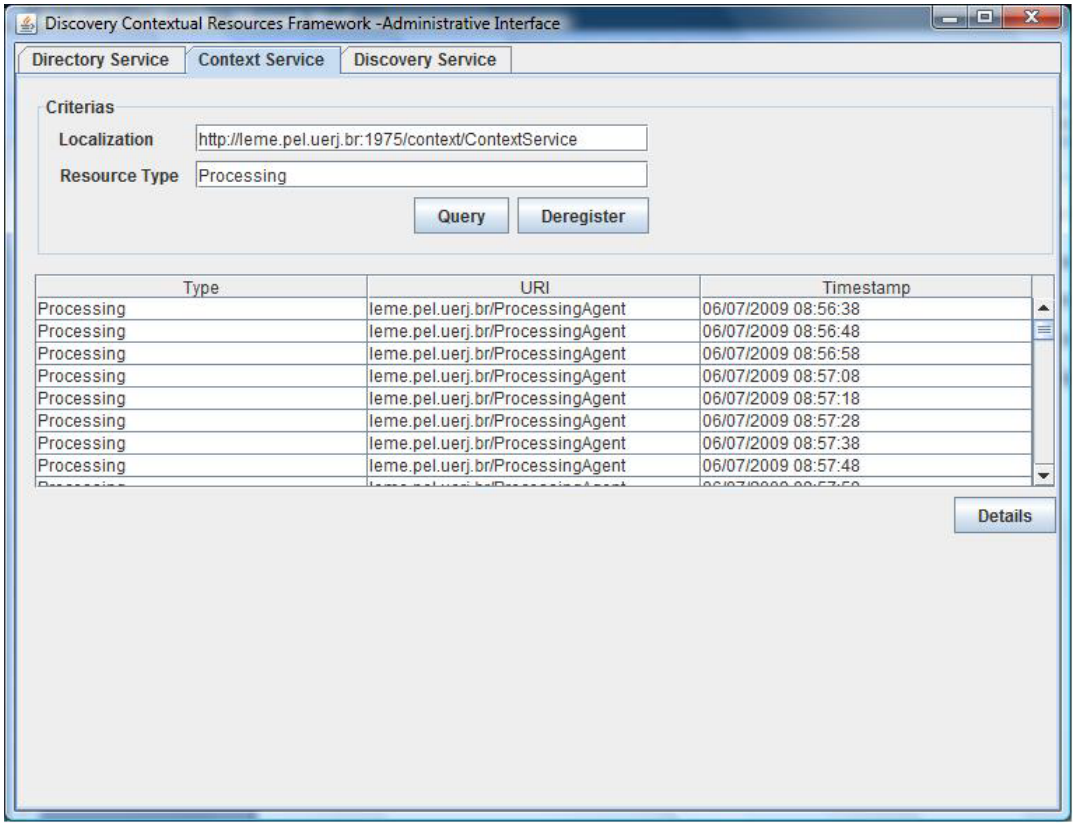


Figura 16 - Resultado de consulta ao SC assíncrona através da Interface Administrativa

A tabela da Figura 16 contém cada medida enviada pelo Serviço de Contexto à interface administrativa. Caso se deseje interromper o envio de mensagens, deve-se pressionar o botão *"Deregister"* que irá informar ao SC que o mesmo deve remover a interface administrativa de sua lista de observadores.

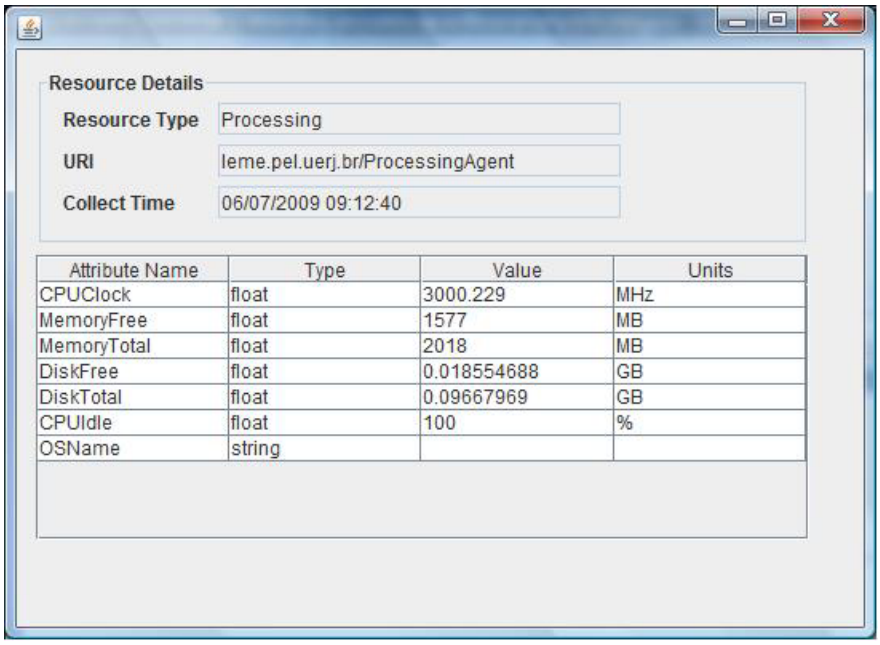


Figura 17 - Detalhes sobre o estado de um AR através da Interface Administrativa

O botão *"Details"* é utilizado para exibir os detalhes sobre uma media específica que foi coletada. Para tanto antes de se pressionar o botão uma medida deve ser seleciona na lista. A interface irá exibir uma tela similar à apresentada na Figura 17. Onde todos os valores dos atributos do AR no momento em que os mesmos foram coletados são exibidos. Na tela se vê o momento em que as medidas foram feitas (às 09:12:40h do dia 07/07/2009), o tipo e o URI do recurso assim como as propriedades do mesmo.

A Interface Administrativa foi desenvolvida, conforme comentado, como exercício de prova de conceito. Como tal seu funcionamento é limitado. Entretanto, o mesmo também serve de exemplo para desenvolvedores e pode ter suas funcionalidades ampliadas facilmente.

A seguir serão apresentados os resultados obtidos na avaliação de desempenho que foi realizada junto aos elementos da infra-estrutura.

4.7 Avaliação de desempenho

Para avaliar o impacto e a escalabilidade dos serviços propostos foram realizadas medidas de desempenho com a implementação de referência. Para as medidas foi utilizado o utilitário JMetter (JMetter, 2007), uma ferramenta capaz de efetuar testes de carga em aplicações construídas com diferentes tecnologias: páginas HTML estáticas e dinâmicas, acessa bancos de dados através de JDBC, servidores LDAP, JMS, email e, no caso da implementação de referência, mensagens SOAP.

No teste de desempenho dos serviços componentes da infra-estrutura simulamos o acesso aos mesmos inicialmente por 10 usuários simultâneos, e incrementamos a quantidade de usuários em múltiplos de 10 até um total de 100 usuários simultâneos com um período de aquecimento (*ramp-up*) de 1 segundo. Ou seja, no período de um segundo todos os usuários do teste enviaram suas requisições para o servidor. Cada teste foi repetido 32 vezes e o tempo médio de resposta e o intervalo de confiança foram calculados para cada conjunto de usuários.

Para o teste do SRD, foram desenvolvidos 1.000 Agentes de Recurso de teste com 10 propriedades do tipo *string* cada um. O teste consistiu em retornar a lista de todos os ARs registrados no SRD. Ou seja, os 1.000 ARs desenvolvidos para os testes.

Os resultados obtidos são apresentados na Figura 18. Com 10 usuários simultâneos, o tempo médio foi de 110ms enquanto que com 50 usuários o tempo médio foi de 435ms e com 100 usuários 999ms.

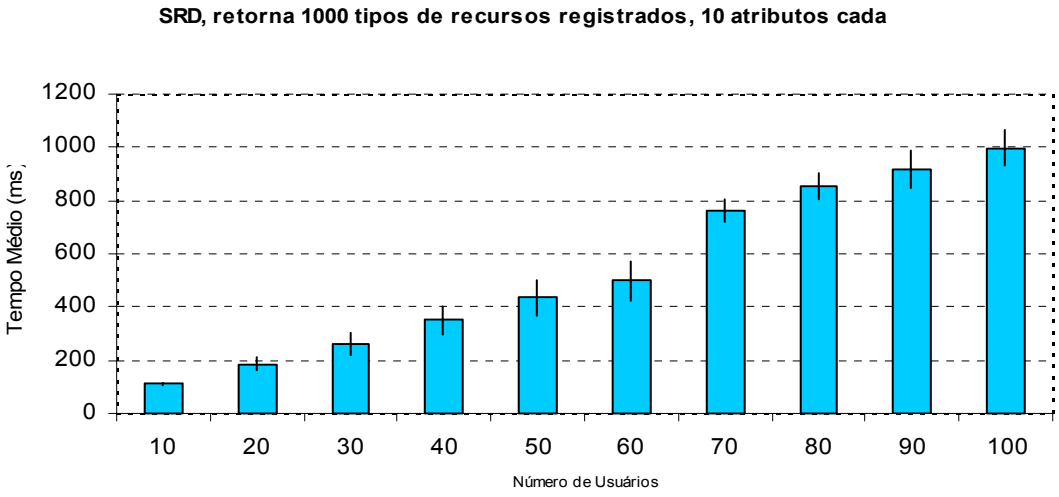


Figura 18 - Desempenho do SRD em função do número de usuários

Na Figura 19 são apresentados os resultados obtidos na avaliação do Serviço de Contexto. Neste teste o SRD continuava com os 1.000 ARs registrados do teste realizado no SRD e as consultas foram feitas para um AR de testes cujo URI era conhecido e com uma restrição de contexto relativa a processamento. Na máquina de testes havia um AR de processamento sendo executado que coletava informações sobre o contexto de processamento a cada 10s.

Com esta configuração, os resultados obtidos foram: com 10 usuários o tempo médio de acesso foi de 217ms, com 50 usuários foi de 1.199ms e com 100 usuários foi de 2.145ms. Neste tempo está computado também o tempo de pesquisa junto ao SRD, para localizar os ARs do tipo de processamento que se encontram no mesmo nodo do AR de teste.

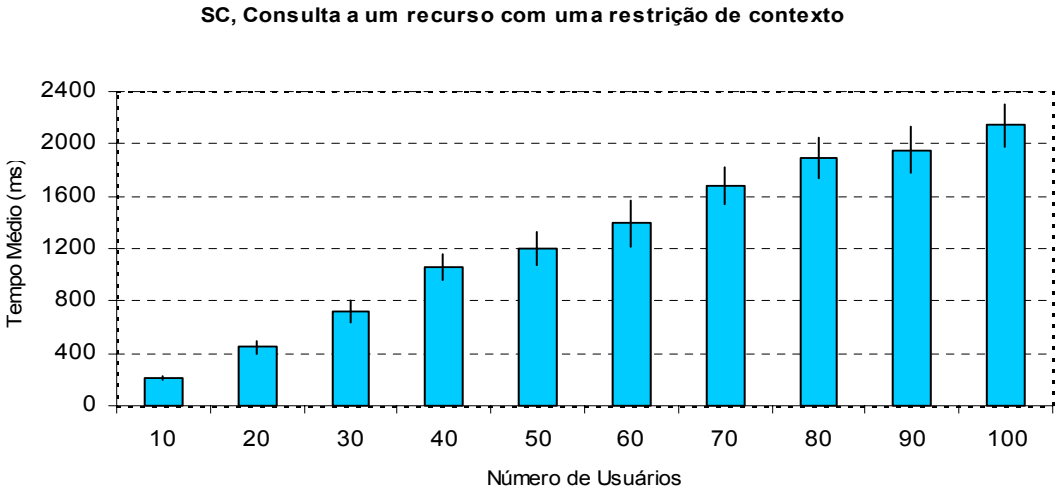


Figura 19 - Desempenho SC - Tempo de resposta médio x número de requisições concorrentes

O resultado do teste de escalabilidade referente ao Serviço de Descoberta é apresentado na Figura 20. A consulta utilizada no teste é baseada em um tipo de teste e em uma restrição de contexto relacionada a processamento. O mesmo AR desenvolvido para o teste do SC foi utilizado neste teste. Nos tempos obtidos, estão computados os tempos relativos às consultas ao SRD e ao SC. Para 10 usuários o tempo total 1.733ms, para 50 usuários foi 2.641ms e para 100 usuários foi de 4.722ms.

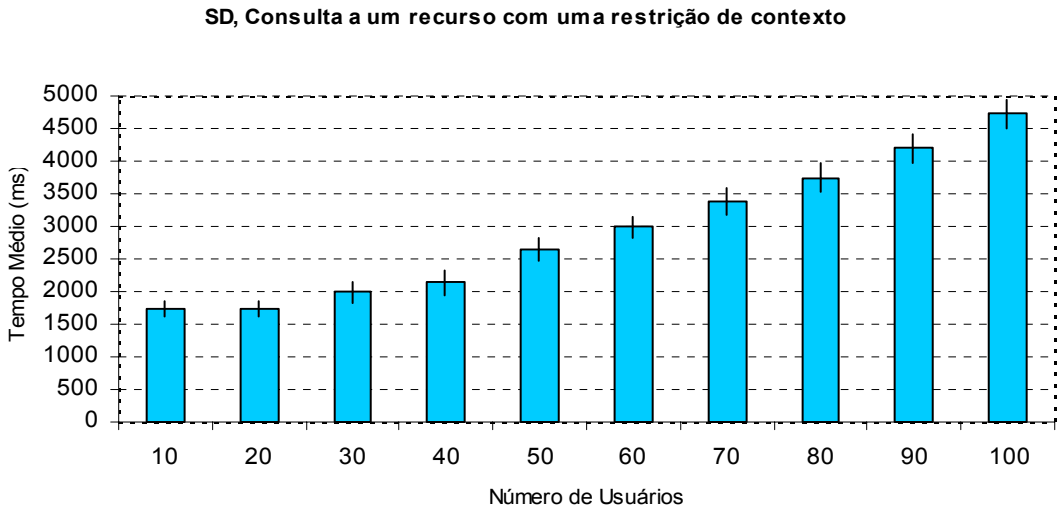


Figura 20 - Desempenho SD - Tempo de resposta médio x número de requisições concorrentes

4.7.1 Discussão

A avaliação de desempenho realizada mostrou que os serviços se comportam bem, com tempos de resposta aceitáveis para o tipo de aplicação considerado (aplicações cientes de contexto, sistemas de computação ubíqua e pervasiva).

Não realizamos testes de desempenho considerando federações de domínios, ou seja, Serviços de Registro e Diretório federados, mas acreditamos que consultas deste tipo tenham problemas de escalabilidade similares aos encontrados em redes ponto-a-ponto.

Além disso, todos os serviços foram implantados na mesma máquina, para podermos simular o pior caso de utilização dos componentes. Observamos que na implementação de referência, utilizamos o servidor HTTP que vem embutido no JDK da Sun, que é pouco otimizado na função de servidor de aplicações. Os resultados provavelmente podem ser melhorados se for utilizado um servidor de aplicações otimizado para atender requisições de Serviços Web, tais como o servidor de aplicações JBoss ou o Tomcat.

CAPÍTULO 5 APLICAÇÕES DE EXEMPLO

O modelo dos serviços propostos e a implementação de referência foram validados através de exemplos de aplicações cientes de contexto, que empregaram os mesmos como infra-estrutura de suporte. O primeiro exemplo trata de uma aplicação na área de tele-saúde cujo objetivo é monitorar pacientes idosos em suas casas. O sistema precisa obter informações do ambiente, medidas médicas e a atividade do paciente para fazer um registro do estado do mesmo e avaliar sua condição de saúde. O segundo exemplo trata de sistemas tolerantes a falhas, em que as réplicas devem ser selecionadas dinamicamente e a estratégia de replicação pode ser alterada, dependendo do contexto de operação do sistema.

5.1 Aplicação de tele-saúde

Esta aplicação está sendo desenvolvida no contexto do projeto de pesquisa "Aplicando técnicas de computação ubíqua a aplicações de tele-saúde" apoiado pela FAPERJ (Loques, 2007) e envolvendo alunos da UFF e UERJ.

A aplicação de tele-saúde desenvolvida tem por objetivo monitorar pacientes idosos em suas residências. Na literatura estas aplicações são denominadas *Remote Assisted Living Application* (Brownsell, 2007; ATSP, 2007), que traduzimos por Assistência Domiciliar de Saúde Telemonitorada. Uma instalação típica onde esta aplicação é implantada envolve sensores, que coletam dados do paciente e enviam estes dados para uma central de monitoramento (uma clínica, um médico ou hospital). Em nossa abordagem incorporamos inteligência local: na casa do paciente, sensores de ambiente e sensores portados pelo mesmo constantemente geram dados, que são coletados pelo sistema de computação local. Estes dados são interpretados pelo sistema utilizando conhecimento médico. A identificação de condições anormais no paciente pode ativar dispositivos locais (por exemplo, ligando o sistema de ar-condicionado), iniciar uma interação com o paciente (por exemplo, através da tela da TV) ou enviar uma mensagem de emergência para o centro de monitoração.

Os serviços propostos de Descoberta e Contexto foram utilizados como infra-estrutura de suporte para a descoberta, coleta e persistência das informações de contexto obtidas dos sensores de ambiente e de dispositivos médicos. Assim sendo estes elementos precisam ser especificados através de classes de recursos, com suas respectivas propriedades e precisam ser

representados por Agentes de Recursos. A Listagem 30 apresenta a descrição de um dispositivo genérico de medida de pressão arterial, cujo tipo foi nomeado *BloodPressure,* e que contém as propriedades de pressão sistólica e diastólica (linhas 7 e 8). Linhas 5 e 6 apresentam atributos específicos do aparelho, o modelo e o número de série.

```
1  <Resource>
2    <Type>BloodPressure </Type>
3    <Description>Blood Pressure Resource Agent</Description>
4    <Attributes>
5      <Attribute Name="model" Type="string" />
6      <Attribute Name="serial" Type="string" />
7      <Attribute Name="systolic" Type="integer" Units="mmHg" />
8      <Attribute Name="diastolic" Type="integer" Units="mmHg" />
9    </Attributes>
10 </Resource>
```

Listagem 30 - Descrição do recursos da classe BloodPressure

Nas próximas subseções apresentamos os elementos principais desta aplicação e como os serviços propostos foram utilizados.

5.1.1 O Home Health System

O *Home Health System* (HHS), Sistema de Saúde Domiciliar, é um sistema de software projetado para coletar, armazenar, analisar e distribuir dados de contexto relacionados à saúde. Neste caso dados de contexto compreendem medidas dos sensores ambientais, atividade atual do paciente e sua localização na residência, e dados de medidas médicas / fisiológicas. O HHS avalia os dados de contexto coletados para inferir o estado de saúde do paciente. Todos os dados de contexto e o estado de saúde inferido a partir destes dados são armazenados em uma base de dados local para manter um histórico individualizado do paciente. Este histórico também é enviado para um centro de monitoramento e disponibilizado para o médico ou outros profissionais de saúde envolvidos, e para determinados familiares do paciente.

O núcleo do sistema é o Plano de Cuidados (*Care Plan*) composto por dados de prescrição médica e rotinas direcionadas ao paciente. A Figura 21 apresenta a organização geral dos módulos do HHS.

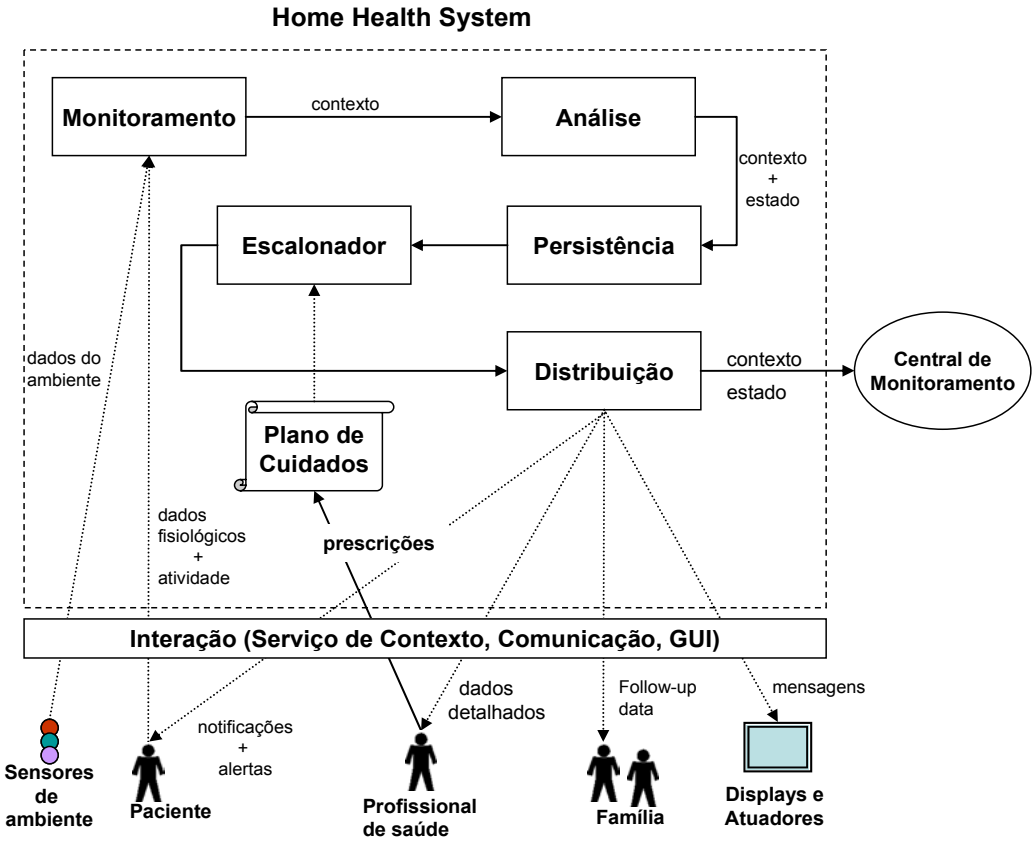


Figura 21 - Organização do Home Health System

- Monitoramento: descobre e monitora sensores e aparelhos para coletar dados de contexto;
- Análise: aplica uma série de regras além de utilizar um modelo fuzzy para inferir o estado de saúde do paciente, tendências de severidades e situações críticas;
- Persistência: utiliza um banco de dados relacional para armazenar os dados coletados e o estado de saúde inferido;
- Escalonador: pode ser programado para enviar mensagens de alerta, transmitir dados de contexto para o centro de monitoramento e para disparar ações pré-configuradas em tempos específicos;
- Distribuição: responsável por encaminhar dados de contexto para uma central de monitoramento e por enviar notificações ou dados detalhados para o médico ou profissionais de saúde;
- Interação: trata a apresentação dos dados monitorados, provendo visões apropriadas para cada ator do sistema, isto é, o paciente, membros da família e profissionais de saúde autorizados.

Maiores detalhes sobre a arquitetura do HHS e a o módulo de Análise podem ser obtidos em (Copetti, 2009).

5.1.2 O cenário

O cenário utilizado é esquematizado na Figura 22. Cada cômodo tem um conjunto de sensores de ambiente (temperatura, luminosidade e umidade). Pontos de acesso WiFi (WiFi AP) são instalados para facilitar a comunicação sem fio. O sistema de aquecimento e refrigeração da casa e as TVs Digitais (TVDi) devem ser capazes de receber e exibir mensagens e são parte da aplicação. O paciente precisa portar um *tag* de localização e o sensor de movimento (um acelerômetro).

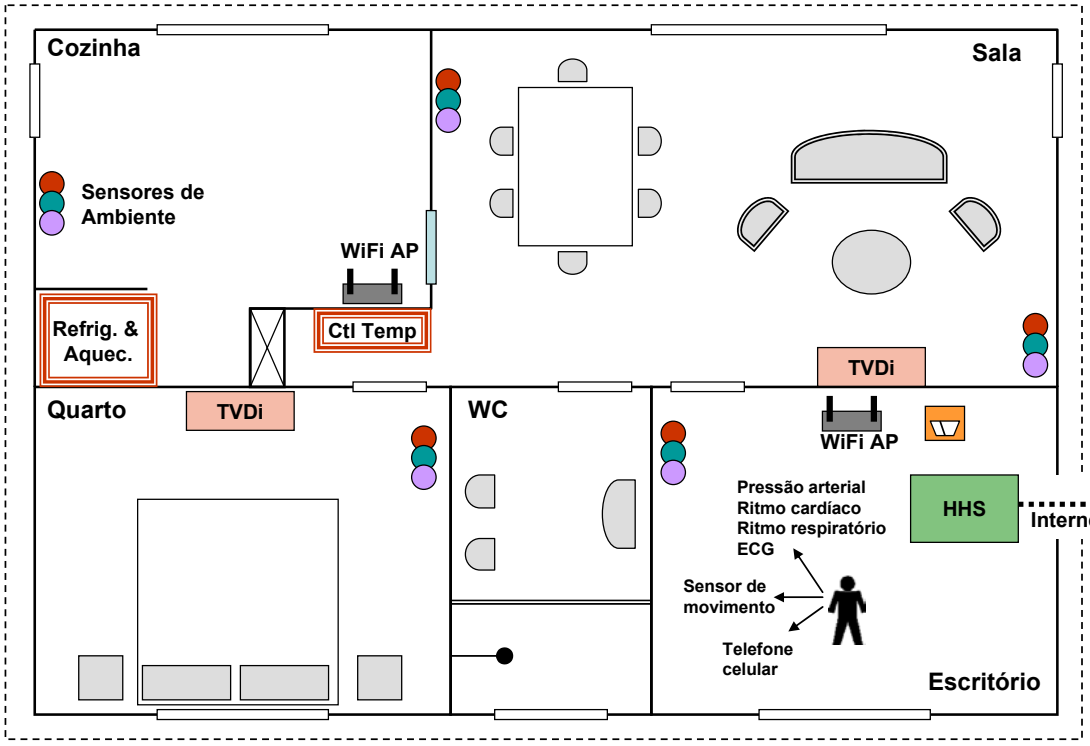


Figura 22 - Ambiente da aplicação de tele-saúde

O paciente também deverá colocar o aparelho médico quando o plano de cuidados assim indicar. Todos os dispositivos se comunicam com o sistema computacional utilizando um mix de redes com e sem fio. Uma instância do HHS implantada em um nó de computação executa um sistema de software composto de um banco de dados, um módulo de comunicação, o escalonador inteligente e um visualizador de dados.

Como dito anteriormente, cada sensor ou dispositivo é representado por um AR. O software executando no HHS utiliza o Serviço de Descoberta e o Serviço de Contexto para coletar informações de contexto dos ARs, solicitando informações de forma síncrona ou recebendo notificações de forma assíncrona.

5.1.3 Implantação

O uso cotidiano do sistema proposto deve considerar que o paciente ou o cuidador acompanhante não são especialistas em computação. Assim, o uso do sistema deve ser precedido de uma etapa de instalação e implantação, quando são realizados os seguintes passos:

Instalação básica. Os sensores de ambiente são instalados, ajustados e calibrados. O HHS é carregado no nó de computação e a segurança apropriada é configurada (senhas, chaves e certificados digitais);

Configuração do HHS. Informações básicas são configuradas: o mapa da residência (em nosso protótipo, o nome de cada cômodo) para permitir a localização do paciente, e a URI dos serviços de suporte (Registro e Diretório, Descoberta e Contexto);

Comunicação. A comunicação com o sistema central, médico e cuidadores, via rede ou telefonia celular, são configurados;

Descrição e registro de sensores. Cada sensor é descrito, de acordo com a classe do AR e registrado no Serviço de Registro e Diretório;

Seleção de equipamentos médicos. Uma enfermeira ou cuidador deve obter o Plano de Cuidados inicial e selecionar os aparelhos de medição de dados fisiológicos apropriados;

Registro dos aparelhos médicos. Após a seleção dos aparelhos, as descrições das classes dos ARs correspondentes são customizadas com a ajuda da Interface Gráfica do HHS, e os ARs correspondentes são também registrados no Serviço de Registro e Diretório.

A Listagem 31 apresenta o registro do AR *BloodPressure* (linha 2). A *tag URI* (linha 9) identifica o AR específico. Os outros atributos apresentam as informações específicas do aparelho sendo registrado. Por exemplo, o modelo do aparelho é *WristClinic* (linha 4) e o número de série é TelcoMed004E (linha 5). Ainda, é informado que os atributos da medida de pressão sistólica e diastólica são dinâmicos.

```
 1 <ResourceRegister>
 2   <Type>BloodPressure</Type>
 3   <Attributes>
 4     <Attribute Name="model" Value="WristClinic" />
 5     <Attribute Name="serial" Value="TelcoMed004E" />
 6     <Attribute Name="systolic" Value="Dynamic" />
 7     <Attribute Name="diastolic" Value="Dynamic" />
 8   </Attributes>
 9   <URI>hhs/BPAgent</URI>      <!-- identifica o recurso -->
10   <IP>152.92.155.196</IP>     <!-- onde o AR se encontra -->
11   <Tecnologies>
12     <Tecnology Type="WebService"
13                URL="http://hhs.pel.uerj.br:1986/BloodPressure?wsdl"/>
14   </Tecnologies>
15 </ResourceRegister>
```

Listagem 31 - Registro do AR BloodPressure no SRD

Uma vez completados os passos anteriores o sistema pode executar uma **varredura de sensores** (*sensor survey*). Esta varredura consiste em consultar o Serviço de Descoberta para identificar os ARs registrados no Diretório, relativos aos sensores de ambiente, que estão efetivamente disponíveis em cada cômodo. Esta informação confirma quais são os sensores que devem prover dados de ambiente em cada cômodo, e permite detectar falha de funcionamento de um sensor durante a operação normal do sistema. A Listagem 32 detalha uma consulta para localizar todos os recursos registrados na infra-estrutura ("*", linha 1). As Linhas 2 a 4 descrevem as restrições de contexto a serem satisfeitas, informado neste caso que a localização dos recursos (*LocationID*) deve ser o dormitório (*bedroom*).

```
1 <ResourceQuery type="*">
2   <Constraints>
3     <Attribute Name="LocationID" op="==" Value="bedroom" />
4   </Constraints>
5 </ResourceQuery >
```

Listagem 32 - Varredura de sensores

Uma possível resposta para a consulta anterior, com duas referências, é apresentada na Listagem 33. A *tag URI* presente em cada resposta contém a referência para os recursos localizados e os outros dados são relacionados com propriedades de contexto dos ARs descobertos (*Temperature* e *Luminosity*).

```
 1 <DiscoveryResponse>
 2   <URI>services.Luminosity</URI>
 3   <ResourceInfo From="Luminosity">
 4     <Attributes>
 5       <Attribute Name="Luminosity" Value="30" />
 6     </Attributes>
 7   </ResourceInfo>
 8 </DiscoveryResponse >
```

```
 9 <DiscoveryResponse>
11   <URI>services.Temperature</URI>
12   <ResourceInfo From="Temperature">
13     <Attributes>
14       <Attribute Name="minTemp" Value="19" />
15       <Attribute Name="maxTemp" Value="27.5" />
16     </Attributes>
17   </ResourceInfo>
18 </DiscoveryResponse >
```

Listagem 33 - Resposta à Varredura de Sensores

5.1.4 Operação

Neste ponto o sistema está configurado com as informações necessárias e pode iniciar sua operação. Quando chega o momento de realizar uma medida fisiológica, o paciente (ou o cuidador) deve colocar corretamente o aparelho e realizar a medida. Este procedimento seria desnecessário se o aparelho pudesse ser "vestido" durante todo o tempo e se o mesmo pudesse ser programado para iniciar uma medida automaticamente de acordo com a escala do Plano de Cuidados. O AR representando o aparelho médico notifica o Serviço de Contexto, que por sua vez envia uma notificação para o HHS com a nova medida.

Para que as notificações sejam enviadas para o HHS, logo após que os aparelhos médicos a serem usados sejam selecionados e registrados no Serviço de Registro e Diretório, o HHS também registra um Observador junto ao Serviço de Contexto. A Listagem 34 exemplifica tal registro. O cliente, no caso o HHS, é identificado por seu URI, e informa estar interessado em ser notificado quando uma nova medida for realizada em recursos do tipo *BloodPressure.*

```
1 <ResourceObserver>
2   <URI>hhs.BPObserver</URI> <!-- URI para callback -->
3   <Description>Blood Pressure Monitoring Device</Description>
4   <!-- <ServiceName>hhs.BPAgent</ServiceName> -->
5   <ResourceType>BloodPressure</ResourceType>
6 </ResourceObserver>
```

Listagem 34 - Registro do HHS como Observer

Desta forma, o Serviço de Contexto notifica o Observador em *hhs.BPObserver* (linha 2) assim que uma nova medida for detectada por qualquer AR *BloodPressureRAgent.* Outras informações como "bateria fraca" também poderiam ser notificadas. Caso o URI do AR fosse previamente conhecido, poderia ser utilizado o elemento que se encontra comentado na linha 4. Assim, notificações de mudanças ocorridas apenas neste AR seriam notificadas. No entanto, caso o AR fosse substituído por outro com outro URI, o HHS não seria notificado de

forma automática, o que torna o uso de notificações baseadas em URI uma desvantagem neste caso.

A Listagem 35 apresenta uma possível notificação. A informação da linha 2 permite identificar que se trata de uma notificação de um AR BloodPressure, cuja referência é a URI da linha 8. As linhas 4 a 7 apresentam as propriedades efetivamente medidas e os respectivos valores.

```
1  <ResourceState>
2    <Type>BloodPressure</Type>
3    <CollectTime>2008-09-25 T 23:24:15</CollectTime>
4    <Attributes>
5      <Attribute Name=”systolic" Values="13" Units="mmHg" />
6      <Attrib Name="diastolic" Values="6" Units="mmHg" />
7    </Attributes>
8    <URI>hhs.BPAgent</URI>
9  </ResourceState>
```

Listagem 35 - Notificação de nova medida de pressão arterial

A medida obtida é avaliada e os valores relevantes são armazenados no banco de dados. Na seqüência, o Serviço de Contexto é consultado para obter informações do AR do serviço de localização, *UserLocation*, com o objetivo de identificar em que cômodo o usuário se encontra. A Listagem 36 apresenta tal consulta. A linha 3 especifica que o AR a ser consultado, *UserLocation.* Na linha 4 o operador lógico informa que o cliente está interessado em resultados se o valor do atributo *userId* é igual a 712 (no caso, o código do paciente). Observa-se que a consulta solicita ainda que o intervalo mínimo de coleta de informações por parte do AR seja de 15 segundos (linha 5).

```
1  <ContextQuery>
2    <Target URI=”rtls.UserLocation”>
3      <Attributes From=”UserLocation”>
4        <Attribute Name=”userId” op=”==” Value=”712” />
5        <CollectInterval Min=”15” Units=”s”/>
6      </Attributes>
7    </Target>
8  </ContextQuery>
```

Listagem 36 - Consulta ao AR do serviço de Localização

A Listagem 37 apresenta uma resposta à consulta anterior, com a informação de contexto. Na linha 2, além do URI é informado que o intervalo de coleta do AR que respondeu a esta consulta é de 2 segundos, o que atende à restrição imposta na consulta. A linha de 3 a 6 contém os atributos e os respectivos valores. Na linha 5, por exemplo, é informado que o valor do atributo *currentRoom* (cômodo corrente) é igual a *office* (escritório).

```
1  <ContextResponse>
2    <ResourceInfo URI =”rtls.UserLocation”  Interval=”2”
                                             Updated=”2009-05-01T13:16:55”>
3      <Attributes From=”UserLocation “>
4        <Attrib Name=”userId” Value=”712” />
5        <Attrib Name=”currentRoom” Value=”office” />
6      </Attributes>
7    </ResourceInfo >
8  </ContextResponse>
```

Listagem 37 - Resposta à Consulta Realizada ao Serviço de Contexto

Com a localização do usuário, o HHS pode obter os valores atuais dos sensores de ambiente. Para a aplicação é importante armazenar o contexto do ambiente, pois dados fisiológicos tais como pressão arterial ou ritmo cardíaco podem sofrer influência da temperatura ambiente, por exemplo. A Listagem 38 apresenta a consulta ao Serviço de Contexto para a obtenção das medidas dos sensores do escritório (*office*).

```
1  <ContextQuery>
2    <Target URI="services.Temperature">
3      <Attributes>
4        <Attribute Name="location" op="==" Value="office"/>
5      </Attributes>
6    </Target>
7    <Target URI="services.Luminosity">
8      <Attributes>
9        <Attribute Name="location" op="==" Value="office"/>
10     </Attributes>
11   </Target>
12  </ContextQuery>

13  <ContextResponse>
14    <ResourceInfo URI="services.Temperature">
15      <Attributes Update="2009-05-05T20:32:00.0" Interval="60000">
16        <Attribute Name="location" Value="office"/>
17        <Attribute Name="measure" Value="25.5" Units="C"/>
18      </Attributes>
19    </ResourceInfo>
20    <ResourceInfo URI="services.Luminosity">
        …
21    </ResourceInfo>
22  </ContextResponse>
```

Listagem 38 - Consulta aos ARs dos sensores do escritório (office)

Em seguida o HHS ainda consulta o AR de atividade, *Activity*, para obter informações da atividade do paciente.

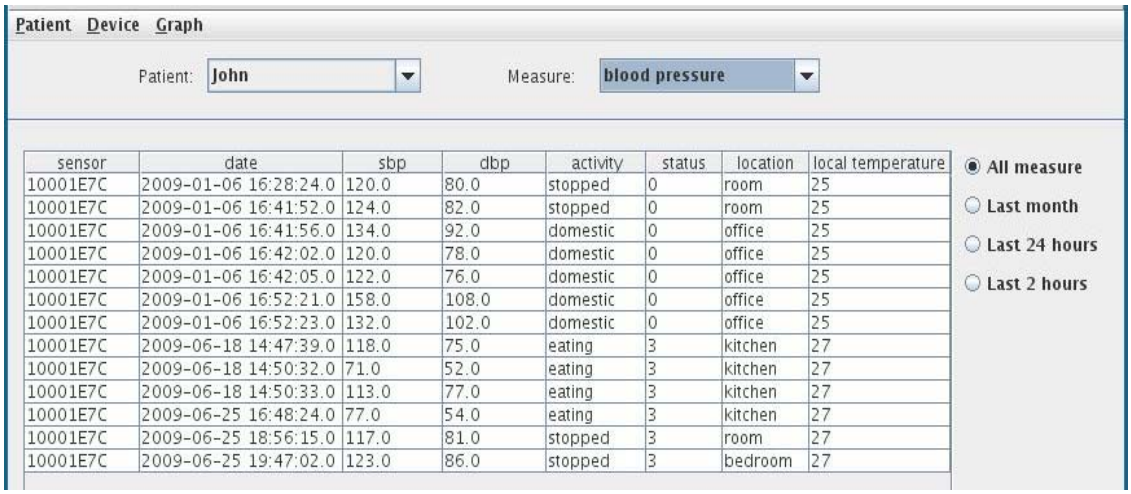
Uma vez que o módulo de Monitoramento tenha coletado todas as informações de contexto necessárias o mesmo as entrega ao módulo de Análise. O contexto composto por todos os dados coletados é avaliado como um todo. O módulo de Análise, cujas regras são baseadas em guias da área médica é capaz de inferir condições críticas e tendências. Informações de contexto obtidas em seqüência podem levar a classificação de situações

críticas e disparar procedimentos de alerta. Além disso, tais informações podem ser classificadas simplesmente como suspeitas e disparar um procedimento que tentará corrigir dados potencialmente errados (Copetti, 2009).

O gerente de Plano de Cuidados inclui um sistema de notificação programável, que pode configurar o escalonador para apresentar mensagens específicas com o objetivo de alertar o paciente ou o cuidador de que uma dada ação deve ser executada (por exemplo, tomar um remédio, executar uma medida ou mesmo recarregar a bateria de um dispositivo). Também pode ser escalada a transmissão de mensagem de alerta para o médico ou para o centro de monitoramento, quando determinadas condições ou limites sejam detectados.

5.1.5 Protótipo

Utilizando alguns elementos da implementação de referência dos serviços, um protótipo da aplicação de tele-saúde foi desenvolvido. Utilizou-se um dispositivo chamado WristClinic (TelcoMed, 2007), capaz de medir pressão arterial, ritmo cardíaco, concentração de oxigênio (SpO2), temperatura corporal e ritmo respiratório. No protótipo utilizamos medidas de pressão arterial. Além do WristClinic utilizamos *loggers* de temperatura para medir a temperatura ambiente (detalhes na próxima seção), e acelerômetros do dispositivo SunSPOT (SunSPOT, 2009) para obter informações simples sobre a atividade do paciente (em pé, deitado, sentado, correndo, andando). Baseado nestas informações um sistema de regras *fuzzy* infere o estado de saúde do paciente (Copetti, 2009). Os dados coletados são armazenados em um banco de dados PostgreSQL. A Figura 23 apresenta uma amostra da Interface Gráfica desenvolvida na forma de um Applet em Java.



sensor	date	sbp	dbp	activity	status	location	local temperature
10001E7C	2009-01-06 16:28:24.0	120.0	80.0	stopped	0	room	25
10001E7C	2009-01-06 16:41:52.0	124.0	82.0	stopped	0	room	25
10001E7C	2009-01-06 16:41:56.0	134.0	92.0	domestic	0	office	25
10001E7C	2009-01-06 16:42:02.0	120.0	78.0	domestic	0	office	25
10001E7C	2009-01-06 16:42:05.0	122.0	76.0	domestic	0	office	25
10001E7C	2009-01-06 16:52:21.0	158.0	108.0	domestic	0	office	25
10001E7C	2009-01-06 16:52:23.0	132.0	102.0	domestic	0	office	25
10001E7C	2009-06-18 14:47:39.0	118.0	75.0	eating	3	kitchen	27
10001E7C	2009-06-18 14:50:32.0	71.0	52.0	eating	3	kitchen	27
10001E7C	2009-06-18 14:50:33.0	113.0	77.0	eating	3	kitchen	27
10001E7C	2009-06-25 16:48:24.0	77.0	54.0	eating	3	kitchen	27
10001E7C	2009-06-25 18:56:15.0	117.0	81.0	stopped	3	room	27
10001E7C	2009-06-25 19:47:02.0	123.0	86.0	stopped	3	bedroom	27

Figura 23 - Amostra da Interface Gráfica do HHS

Algumas atividades de desenvolvimento do protótipo estão em andamento:

- A integração dos ARs com o banco de dados e o HHS está sendo realizada. Vários módulos já foram concluídos e testados individualmente;
- A localização do paciente é feita manualmente, transmitindo-se a informação a partir de um PDA. Um sistema automático baseado em informações de pontos de acesso WiFi está sendo incluído ao sistema;
- O AR de identificação de atividade está sendo desenvolvido e aprimorado para classificar atividades de forma mais precisa. Por exemplo, a queda do paciente pode ser detectada, atividade de subir escada ou varrer a casa também poderia ser mapeada;
- Procedimentos para reconfigurar dispositivos do ambiente também estão sendo desenvolvidos. Alguns experimentos foram realizados com dispositivos X10 (X10, 2007) para ligar e desligar ou diminuir a iluminação do ambiente de acordo com a localização atual do paciente.

Além destas adições, outras melhorias estão sendo estudadas, no âmbito do projeto FAPERJ, como, por exemplo, o sistema de regras fuzzy, no que diz respeito à precisão, e a economia de energia dos sensores baseada na localização do paciente.

5.1.6 AR de temperatura

Para a aplicação de tele-saúde foi necessário o desenvolvimento de alguns ARs para sensores de ambiente. Tais ARs são importantes nesta aplicação. Por exemplo, as medidas de batimentos cardíacos e pressão arterial podem ser influenciadas pela temperatura. O contexto composto pela temperatura, pressão arterial e atividade do paciente pode disparar diferentes ações, dependendo de regras *fuzzy* ou políticas inseridas no módulo de Análise. Esta atividade permitiu validar o padrão de projeto proposto para os ARs. Por exemplo, utilizamos um *logger* de temperatura para monitorar a temperatura ambiente.

O DS1921G Thermochron iButton da Dallas Semiconductor (Dallas, 2007) é um sensor de temperatura que mede e registra temperaturas na faixa de -40°C to 85°C. O iButton se conecta ao computador hospedeiro através de porta USB ou serial. O fabricante oferece um acionador Java que utiliza a API para comunicação serial da biblioteca RXTX (Javari, 2008). Um AR foi desenvolvido utilizando-se este acionador Java.

O AR *AbstractTemperatureAgent* (Figura 24) segue o modelo apresentado anteriormente. Basicamente a classe *IButtonTemperatureAgent* implementa o método *getResourceValues* herdado da classe abstrata *AbstractResourceAgent*. Este método chama algumas funções de API desenvolvida pela empresa Dallas para efetivamente obter os dados do sensor e verificar se houve alguma alteração de valores.

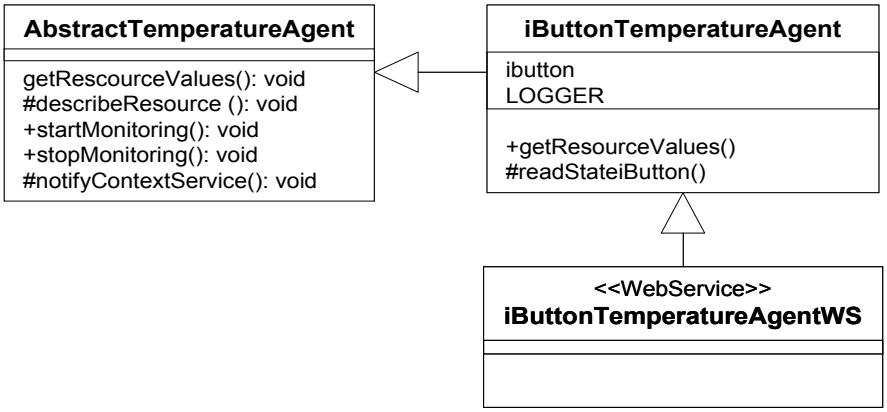


Figura 24 - Classes do AR Temperature

A Listagem 39 apresenta a descrição de um recurso do tipo *Temperature*. Além das informações-padrão como número de série (linha 5) e modelo do dispositivo (linha 6), a faixa de temperatura que o dispositivo suporta (linhas 7-8) e o intervalo de temperaturas para o qual o alarme foi configurado, também podem ser obtidos (linhas 9-10) além da temperatura corrente (linha 11).

```
 1 <Resource>
 2   <Type>Temperature</Type>
 3   <Description>Temperature Sensor</Description>
 4   <Attributes>
 5     <Attribute Name="Serial" Type="string" />
 6     <Attribute Name="Model" Type="string" />
 7     <Attribute Name="MinTemp" Type="double" Units="Celsius" />
 8     <Attribute Name="MaxTemp" Type="double" Units="Celsius" />
 9     <Attribute Name="LowAlarm" Type="double" Units="Celsius" />
10     <Attribute Name="HighAlarm" Type="double" Units="Celsius" />
11     <Attribute Name="LastTemp" Type="double" Units="Celsius" />
12     <Attribute Name="locationId" Type=”string” />
13   </Attributes>
14 </Resource>
```

Listagem 39 - Descrição do recursos da classe Temperature

Além de responder às consultas síncronas, a classe *iButtonTemperatureAgent* utiliza o modelo de *push* disponibilizado pela infra-estrutura (comunicação assíncrona) para notificar o Serviço de Contexto de qualquer mudança no estado do sensor. Esta notificação poderia ser utilizada para enviar um alarme caso um limite previamente configurado no HSS fosse ultrapassado.

5.1.7 Observações

A implementação da aplicação Assistência Domiciliar de Saúde Telemonitorada é um trabalho em andamento, embora módulos relevantes do mesmo já tenham sido implementados e testados. A integração dos dispositivos médicos está sendo realizada enquanto este texto está sendo concluído. O desenvolvimento de aspectos que depende de conhecimento médico, específico, é orientado por médicos colaboradores, da UERJ (Hospital Universitário Pedro Ernesto) e da UFF (Instituto Biomédico).

Alguns aprimoramentos para esta aplicação são planejados com a ajuda da equipe de colaboradores médicos:

- Refinamento das regras para inferir o estado de saúde do paciente, principalmente aqueles relacionados com a classificação precisa de situações críticas e alertas de emergência (parte da tese de Doutorado de Alessandro Copetti, da UFF);
- Auto-adaptação do Plano de Cuidados, em que uma nova prescrição poderia ser ativada por um médico, ou ajustes poderiam ser recomendados automaticamente pelo mecanismo de gerência do plano de cuidados baseado em uma tendência inferida a partir das informações de contexto coletadas pelo sistema.

5.2 Tolerância a faltas

Sistemas distribuídos são sujeitos a faltas decorrentes de problemas na infra-estrutura de hardware e software sobre os quais são executados. Estas faltas podem provocar falhas nestes sistemas levando à indisponibilidade dos serviços providos. O uso de técnicas de tolerância a faltas permite a recuperação destes sistemas, ainda que parcialmente, levando à continuidade dos serviços (Jalote, 1994). Usualmente a tolerância a faltas é obtida através da redundância de elementos de software e hardware. Na maioria das vezes estes recursos são alocados de forma estática podendo haver subutilização dos mesmos por um período grande de tempo. Assim sendo, o dimensionamento adequado de elementos redundantes é importante para controlar os custos finais dos sistemas.

Algumas abordagens para introduzir tolerância a faltas utilizam soluções *ad hoc*, misturando o código responsável por atender os requisitos funcionais com o código responsável pela redundância, através de replicação de componentes, e a manutenção da

consistência das réplicas. O resultado é um código com alto grau de acoplamento, o que prejudica a reutilização. Outras abordagens eliminam este problema utilizando mecanismos embutidos na infra-estrutura de suporte sobre a qual as mesmas são desenvolvidas, separando requisitos funcionais dos requisitos de tolerância a faltas. Ainda assim, a configuração destes mecanismos é definida de forma estática, forçando a pré-alocação de recursos. Exemplos que utilizam esta abordagem são o JEE e .NET.

Abordagens adaptativas para o suporte de requisitos de tolerância a faltas procuram alcançar um equilíbrio entre a robustez e a utilização eficiente dos recursos. Por um lado, técnicas de replicação e consistência diferenciadas podem ser empregadas em contextos de operação específicos, consumindo, obviamente, os recursos necessários (tempo de processamento, banda de rede, etc.). Por outro, a alocação de recursos também pode ser "verde" na medida em que os recursos redundantes passam a ser utilizados apenas quando os mesmos são realmente necessários para atender a uma dada política de tolerância a faltas de forma aceitável.

A aplicação de técnicas de tolerância a faltas adaptativas tem se tornado atraente em instalações do tipo *data centers*, onde servidores devem executar sem interrupção visível dos serviços providos e com determinado nível de qualidade. Falhas e variação de carga devem ser mitigadas, de tal forma que SLAs (*Service Level Agreement*) sejam cumpridos. Por exemplo, técnicas mais robustas, como a replicação *ativa* ou *ativa cíclica*, podem ser acionadas quando há um cenário de operação mais propenso a falhas e, no outro sentido, técnicas mais simples, como a replicação *passiva*, podem ser novamente acionadas quando o cenário de operação voltar a ser mais tranqüilo. Neste contexto, é desejável que os requisitos de tolerância a faltas e as políticas de adaptação possam ser descritas em nível alto de abstração. Também é desejável que estes requisitos possam ser implantados com separação de interesses em relação às aplicações e gerenciados em tempo de execução de forma autonômica.

Nesta aplicação se propõem o uso de contratos arquiteturais para especificar requisitos de tolerância a faltas, em que perfis quantificam propriedades como o *tipo de replicação*, o *número de réplicas* e o *intervalo de checkpointing* desejado; adicionalmente uma máquina de negociação especifica níveis de qualidade desejados e como estes são impostos. Em tempo de execução uma infra-estrutura de software permite a implantação, o monitoramento e a manutenção dos requisitos descritos no contrato de forma autonômica, reconfigurando dinamicamente a aplicação e os recursos responsáveis pela tolerância a faltas para manter os

requisitos contratados.

Na implementação desta aplicação empregamos os contratos descritos em CBabel e a infra-estrutura de suporte de CR-RIO, integrada aos serviços de Descoberta e de Contexto para prover tolerância a faltas em um cenário envolvendo um servidor HTTP Apache (Apache, 2007a), integrado a um grupo de servidores Tomcat (Apache, 2007b). A idéia é acionar dinamicamente uma técnica de replicação adequada para cada contexto de operação específico, considerando o tempo de resposta do conjunto de réplicas e a taxa de faltas associada.

5.2.1 Observações

Servidores de aplicação como o JEE e .NET oferecem mecanismos de replicação, mas não permitem configurar a técnica utilizada. No servidor de aplicação JBoss é possível utilizar a infra-estrutura JBossCache (JBoss, 2007) e AOP (*Aspect Oriented Programming*) para replicar objetos do *cache* de diferentes formas. No entanto o uso destes mecanismos é *ad hoc*. Considerando o Apache-Tomcat, é possível configurar o balanceamento de carga de forma nativa através do conector Mod_JK (Apache, 2007c). No entanto, técnicas de replicação adaptativas também não são suportadas.

Uma abordagem adaptativa para tolerância a faltas em serviços replicados também é explorada em (Kalbarczyk, 2005) numa linha semelhante à deste trabalho. A preocupação principal dos autores é a arquitetura e o desempenho do suporte. As especificações são, no entanto, *ad hoc* e embutidas no código dos serviços.

A organização dos elementos de tolerância a faltas em nossa abordagem é baseada em (Lung, 2006), uma extensão do padrão FT-CORBA da OMG. No entanto, este padrão também não é adaptativo. Uma vez definidos os requisitos de tolerância a faltas os mesmos não podem ser alterados de acordo com as necessidades da aplicação. Em (Lung, 2007) é apresentada uma infra-estrutura para tolerância a faltas adaptativa chamada GroupPac, uma implementação livre do padrão FT-CORBA. Através desta infra-estrutura é possível criar programas baseados em CORBA que mudam suas propriedades de tolerância a faltas de acordo com regras codificadas no programa. No entanto, estas regras são programadas de forma *ad hoc*. Em nossa abordagem a tolerância a faltas é especificada em nível alto e integrada a uma infra-estrutura de suporte recorrente para vários domínios de aplicação.

5.2.2 O *framework* para a implantação de contratos

O *framework* CR-RIO (*Contractual Reflective-Reconfigurable Interconnectable Objects*) é centrado em um modelo arquitetural e em uma linguagem de descrição, CBabel, para descrever a arquitetura funcional de aplicações e expressar seus requisitos não-funcionais por meio de contratos (Loques, 2004). Com base nestes elementos, desenvolveu-se uma infra-estrutura de suporte para: (i) interpretar a especificação dos contratos e armazená-la como meta-informação associada à aplicação; (ii) prover mecanismos de reflexão e adaptação dinâmica, que permitem adaptar a configuração da aplicação (incluindo os seus elementos de suporte), visando atender as exigências de contratos; e (iii) prover elementos para impor, monitorar e manter os mesmos.

5.2.3 Contratos

A configuração funcional de uma aplicação é definida através da especificação de componentes arquiteturais, que realizam atividades essenciais à mesma, não permitindo negociação. Requisitos não-funcionais de uma aplicação são definidos através de restrições operacionais ou de qualidade e podem admitir alguma negociação envolvendo os recursos utilizados. Um contrato descreve em tempo de projeto os aspectos não-funcionais da aplicação, especificando o uso a ser feito de recursos compartilhados durante a operação, e variações aceitáveis na disponibilidade destes recursos:

Categorias. Descrevem propriedades e aspectos não-funcionais específicos de componentes, recursos ou serviços. Por exemplo, características do processador, memória e comunicação. Aspectos menos tangíveis como faixa de preço (“caro”, “barato”), tolerância a faltas, ou qualidade de um componente ou serviço (“boa”, “média”, “ruim”) também podem ser descritos. Cada Categoria é associada a componentes ou serviços de suporte, que irão alocar e monitorar os respectivos recursos, podendo utilizar para isso a infra-estrutura disponível.

Perfis. Quantificam ou valoram as propriedades de uma Categoria. A quantificação restringe cada propriedade, funcionando como uma instância de valores aceitáveis para determinada Categoria. Perfis podem ser definidos, com a granularidade desejada, para indicar o nível de qualidade aceitável no contexto de operação de componentes individuais ou partes da arquitetura.

Serviços ou **configurações** arquiteturais. Especificam versões da arquitetura que vão definir os possíveis níveis de qualidade ou estados de operação da aplicação. Cada configuração contém uma descrição de componentes arquiteturais e suas interações, associados a um ou mais perfis, especializando a arquitetura básica. Assim, o nível de qualidade desejado/tolerado por um serviço ou configuração é diferenciado de outro pelo conjunto das propriedades declaradas nos perfis. Um serviço só pode ser implantado ou mantido se todos os perfis associados ao mesmo forem válidos.

Cláusula de negociação. Descreve uma política, definida por uma máquina de estados, que estabelece uma ordem arbitrária para a implantação das configurações. De acordo com o descrito na cláusula, quando uma configuração de maior preferência (como alta qualidade, por exemplo) não puder mais ser mantida, a gerência de contratos tentará implantar uma configuração de menor preferência (como menor qualidade ou que exija menos recursos). O retorno para uma configuração de maior preferência também pode ser descrito, permitindo que uma configuração de melhor qualidade seja (re)implantada, se os recursos necessários à mesma se tornarem (novamente) disponíveis.

As arquiteturas descritas em CBabel são mapeadas em um modelo de objetos (Corradi, 2005). Este modelo é refletido em um repositório de meta-nível, que pode ser atualizado durante uma (re) configuração e consultado durante a vida da aplicação. Este repositório inclui também a representação dos contratos e mantém informações relacionadas aos recursos de interesse das aplicações. A semântica definida por um contrato é imposta em tempo de operação por uma infra-estrutura de suporte composta por um conjunto de componentes formando padrões reusáveis (Lisbôa, 2004).

5.2.4 Infra-estrutura de suporte

A infra-estrutura empregada no suporte a contratos é constituída de elementos com papéis bem definidos na implantação e reconfiguração da arquitetura das aplicações e na gerência autonômica dos contratos. Para atender aos requisitos da aplicação de tolerância a falhas, integramos os serviços de Registro e Diretório, de Descoberta e de Contexto à infra-estrutura (Figura 25).

Configurador. Elemento responsável por mapear as descrições arquiteturais (em CBabel) em ações que efetivam as configurações requeridas nos sistemas nativos. O Configurador 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 para implantar e reconfigurar a aplicação. Estas operações são atômicas e refletidas no repositório persistente de meta-nível, que pode ser consultado através da API de reflexão arquitetural.

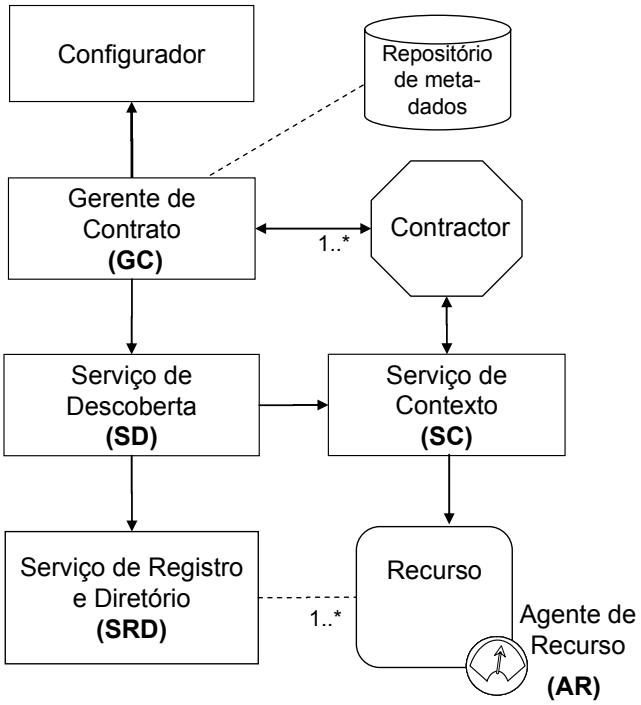


Figura 25 - Infra-estrutura CR-RIO

Gerente de Contratos (GC). Responsável pela implantação dos serviços diferenciados e pela gerência das políticas descritas no contrato. Para implantar uma das configurações do contrato, o GC (i) envia o conjunto de perfis de todas as configurações aos *Contractors*, (ii) solicita a verificação das restrições exigidas pelos perfis associados à configuração a ser implantada e (iii) aguarda a notificação dos mesmos. Se todos os *Contractors* responderem positivamente, a configuração pode ser implantada. Caso contrário, outra configuração deve ser selecionada, de acordo com a cláusula de negociação, e o procedimento de implantação é, então, reiniciado. Se nenhuma das configurações descritas no contrato puder ser implantada, a aplicação é terminada. O mesmo ocorre se durante a operação de uma dada configuração o GC receber uma notificação de perfil inválido. Para efetivar a implantação dos componentes arquiteturais da configuração selecionada (recém negociada com os *Contractors*) o GC utiliza o Configurador. O GC pode também iniciar uma nova negociação, quando os recursos para a implantação de uma configuração de maior preferência se tornam disponíveis (os respectivos perfis são válidos), mesmo se a configuração corrente se mantiver válida.

Contractor. Coordena o processo de alocação e monitoração das propriedades de componentes básicos (mecanismos, recursos ou serviços). Uma aplicação distribuída necessita de um *Contractor* em cada nó do domínio, que recebem do GC o conjunto de todos os perfis. Quando cada *Contractor* inicia o processo de monitoramento, solicita ao Serviço de Contexto os valores monitorados de propriedades de interesse e compara os mesmos com as restrições descritas nos perfis. Em seguida cada *Contractor* se registra junto ao SC como observadores dos recursos que monitoram sendo informados pelo SC de mudanças ocorridas nos recursos. O *Contractor* notifica o GC que a configuração atual não é mais válida se, pelo menos, um dos perfis foi violado. Informa também a lista de todos os perfis que estão válidos, incluindo os relacionados à configuração atual.

Observamos que a semântica dos contratos aciona de forma consistente os elementos da infra-estrutura. Por exemplo, se a arquitetura da aplicação especifica um determinado módulo *TomCat*, estaticamente (cuja referência é previamente conhecida), o GC simplesmente aciona o Configurador, para implantar e alocar o componente específico. Por outro lado, se o módulo é especificado através de uma atribuição dinâmica (cuja referência ainda não é conhecida), o GC aciona o SD para descobrir e selecionar um componente *TomCat* (possivelmente o melhor) antes de solicitar a alocação do mesmo ao Configurador.

5.2.5 Arquitetura, categoria e perfis do Serviço de Replicação

Consideramos a replicação, em nossa abordagem, um serviço de suporte que pode ser utilizado e referenciado em um contrato. Assim, este serviço deve seguir a arquitetura padrão que foi descrita na Seção 5.2.4. Os elementos de suporte à replicação de componentes, comuns em várias propostas, como (Gorender, 2005) e (Lung, 2006) foram integrados à infra-estrutura de suporte de CR-RIO: (a) um Gerente de Replicação (GR), (b) o grupo de réplicas e (c) Controladores de Réplica (CTL-R) para cada réplica individual.

As propriedades de replicação e faltas no nível arquitetural são descritas por Categorias. A categoria *Replication* (Listagem 40) define as propriedades do serviço de replicação, inspiradas em (Lung, 2006), indicando o que pode ser requerido deste serviço, e também o que pode ser monitorado, independentemente da técnica utilizada. As propriedades desta categoria são: (i) o número de réplicas; (ii) o intervalo de *checkpoint* e acionamento do protocolo de consistência (linha 3); (iii) o intervalo de monitoramento de cada réplica (linha 4), e (iv) o tempo limite para que cada réplica responda quando monitorada (linha 5). Com

estas propriedades um *Contractor* que atua como um elemento global da infra-estrutura doravante denominado Gerente de Replicação (GR) pode identificar que o número de réplicas está fora da especificação. Por exemplo, o perfil *ActiveCP* indica que cada réplica deverá responder ao processo de monitoramento, a cada 20s, e em até 200ms (linha 8-9). Uma réplica será considerada indisponível se não responder dentro deste intervalo. O perfil *ActCNRep*, separado por questões de modularidade, indica que o grupo deve ter 4 réplicas.

```
 1  category Replication {
 2     numberOfReplicas: numeric;
 3     checkPointInterval: numeric s;
 4     monitoringInterval: numeric s;
 5     timeoutInterval: numeric ms;
 6  };

 7  profile{
 8     Replication.monitoringInterval = 20;
 9     Replication.timeoutInterval = 200;
10  } ActiveCP;

11  profile{
12     Replication.numberOfReplicas = 4;
13  } ActCNRepP;
```

Listagem 40 - Categoria para replicação, perfil para replicação ativa cíclica

A categoria *Faults* (Listagem 41) é proposta para especificar propriedades relacionadas à faltas: (i) o número de faltas tolerado antes da configuração se tornar inválida (linha 2); o intervalo em que as faltas podem ocorrer (linha 3); e o período de tempo mínimo para que o grupo de réplicas seja considerado estável. A propriedade *stableInterval*, adequadamente utilizada em um contrato, permite que uma técnica menos robusta seja usada, no lugar de uma mais robusta, dado que o número de faltas está, por algum tempo, abaixo do especificado. Além disso, ela aplica certo retardo à decisão de controle, evitando instabilidades devido a condições transitórias. Por exemplo, o perfil *ActiveCFaults* especifica que são toleradas 2 faltas a cada 15seg, e o grupo é considerado estável se não apresentar faltas durante 60s.

```
 1  category Faults{
 2     numberOfFaults: decreasing numeric;
 3     faultInterval: decreasing numeric s;
 4     stableInterval: increasing numeric s;
 5  };

 6  profile{
 7     Faults.numberOfFaults=2;
 8     Faults.faultInterval=15;
 9     Faults.stableInterval=60;
10  } ActiveCFaults;
```

Listagem 41 - Categoria para faltas e perfil de faltas para replicação ativa cíclica

Os perfis baseados nas categorias *Replication* e *Faults* serão usados em conjunto para especificar em um contrato o nível de tolerância a faltas requerido e, em tempo de execução, para avaliar se este nível está sendo respeitado. É necessário, então, incluir os componentes de software responsáveis efetivamente por gerenciar estas propriedades.

As especificações de categorias e perfis são mapeadas em descrições de recursos e as requisições ao Serviço de Descoberta e Contexto são utilizadas para a seleção dos componentes replicados e para o monitoramento destes componentes por parte da infra-estrutura do CR-RIO. Na Listagem 42 apresentamos a representação do serviço de replicação, linhas 1-10, e do sensor de falhas, linhas 11-16.

```
 1  <Resource>
 2    <Type>Replication</Type>
 3    <Description>CR-RIO replication service</Description>
 4    <Attributes>
 5      <Attribute Name="numberOfReplicas" Type="integer"/>
 6      <Attribute Name="checkPointInterval" Type="integer" Units="s"/>
 7      <Attribute Name="monitoringInterval" Type="integer" Units="s"/>
 8      <Attribute Name="timeoutInterval" Type="integer" Units="ms"/>
 9    </Attribute>
10  </Resource>

11  <Resource>
12    <Type>Faults</Type>
13    <Description>CR-RIO sensor faults</Description>
14    <Attributes>
15      <Attribute Name="numberOfFaults" Type="integer" />
16      <Attribute Name="faultInterval" Type="integer" Units="s"/>
17      <Attribute Name="stableInterval" Type="integer" Units="s"/>
18    </Attributes>
19  </Resource>
```

Listagem 42 - Descrição de recursos Replication e Faults

Sob o ponto de vista de arquitetura (Figura 25) o papel do Gerente de Replicação (GR) é encapsulado em um *Contractor*. Baseado nos perfis de replicação e faltas, ele controla a qualidade do serviço, avaliando através do Serviço de Contexto se as réplicas estão "vivas", se o número de réplicas está dentro do especificado e se o número de faltas do grupo está dentro dos parâmetros. O GR realiza suas atividades independentemente da técnica de replicação. As interações entre um módulo cliente e os módulos replicados são realizadas por comunicação de grupo. Como este é um papel de interação, um conector[1] de grupo realiza a difusão seletiva de mensagens para os membros, e um AR associado a este conector monitora a comunicação e a qualidade da difusão.

[1] Módulos representam componentes funcionais da aplicação e conectores a mediação ou interconexões entre os módulos (considerados de meta-nível). Cadeias de conectores podem ser interpostas na rota de interação entre módulos, permitindo filtrar, restringir ou mesmo distribuir as interações.

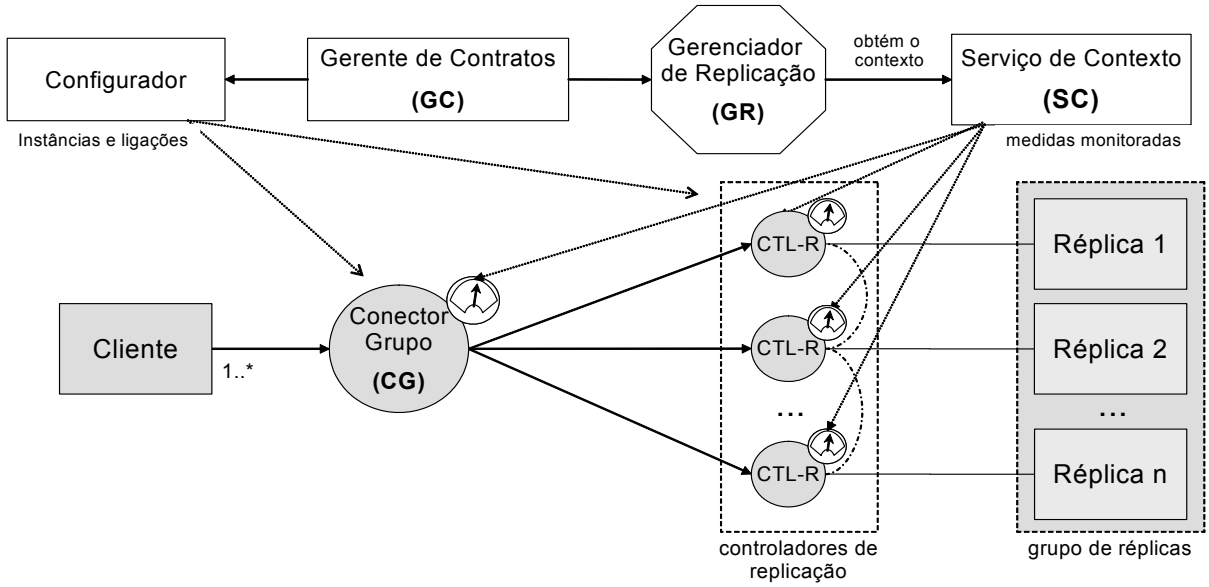


Figura 26 - Estrutura do serviço de replicação

Cada módulo do conjunto de réplicas tem a sua interação com outros módulos da aplicação interceptada por um conector Controlador de Réplica (CTL-R). Este elemento, essencial em nossa abordagem, dá suporte às várias estratégias de manutenção da consistência das réplicas sem interferir diretamente nos módulos replicados. Cada técnica de replicação é associada a um CTL-R especializado. A seleção de um CTL-R específico determina a técnica de replicação a ser utilizada. O CTL-R recebe um perfil contendo as propriedades de monitoração (*interval* e *timeout*). Um AR associado realiza os testes pró-ativamente e registra os resultados no SC. Desta forma é possível ao GR verificar se cada réplica atende aos perfis de replicação. O CTL-R é autônomo para realizar procedimentos de eleição, quando necessário, e se comporta adequadamente quando o mesmo é eleito primário (respondendo as requisições, persistindo o estado e mantendo a consistência do grupo).

Uma vez selecionada uma técnica de replicação e implantada a configuração correspondente, o conjunto de CTL-R realiza os procedimentos para atingir a consistência pós-reconfiguração e os ARs passam a monitorar as propriedades de interesse, declaradas nos perfis. Periodicamente, o GR é informado pelo SC sobre o estado do contexto de execução (através do método de comunicação assíncrona) e consolida os dados a partir dos vários ARs. O GR verifica, então, se os intervalos de tempo e o número de réplicas se mantêm dentro dos perfis da configuração atual e se os mesmos atendem aos perfis de outras configurações. Ao receber uma notificação do GR o GC decide o que deve ser feito. No caso de uma invalidação da configuração de replicação atual, outra versão da configuração da aplicação é selecionada segundo a política descrita na cláusula de negociação do contrato, por exemplo, com uma

técnica de replicação mais robusta, e a rotina para sua implantação é iniciada.

No caso da notificação de configuração válida o GC pode decidir mudar a configuração atual. Por exemplo, se não ocorrerem faltas durante o intervalo especificado pela propriedade *stableInterval,* isso indica que o número de faltas baixou para patamares mais adequados para um serviço que possui menores exigências. Neste caso, o contrato pode especificar, por exemplo, que uma configuração com replicação passiva pode substituir a configuração atual, digamos, ativa cíclica. Caso contrário, nenhuma ação é disparada e a técnica de replicação atual é mantida.

5.2.6 Exemplo de Aplicação

Consideramos um cenário usualmente encontrado em *data centers*: um servidor HTTP Apache e um conjunto de servidores de aplicações Tomcat. O uso de um conjunto replicado de servidores pode ter o objetivo de melhorar a escalabilidade do serviço (ou diminuir o tempo de resposta) e tornar o sistema mais robusto, mitigando falhas através de redundância. Em nosso exemplo, a preocupação está relacionada aos requisitos não-funcionais de tempo de resposta médio do grupo de módulos *TomCat* e a tolerância a faltas:

(a) Sob carga de acesso normal, com tempo de resposta abaixo de 200ms, apenas 2 servidores Tomcat serão utilizados com a técnica de replicação passiva. O objetivo é diminuir o consumo de recursos enquanto as requisições forem tratadas a tempo pela réplica primária;

(b) Se a carga de acesso aumentar e o tempo de resposta aumentar para mais de 200ms, até 4s, 4 servidores redundantes serão alocados e a técnica de replicação ativa cíclica será utilizada. A política é aumentar o número de réplicas para aumentar a vazão de requisições tratadas e utilizar uma técnica de replicação mais robusta, mesmo com consumo maior de recursos.

A Figura 27 apresenta o diagrama geral da arquitetura da aplicação. Uma requisição vinda de um cliente Web é processada pelo módulo *Apache* que identifica através da URL que se trata de conteúdo dinâmico, que deve ser processado por um módulo *TomCat*. Em um cenário sem replicação a comunicação Apache-Tomcat é realizada por dois elementos: o Mod_JK e o AJP, conectores disponibilizados pelos respectivos produtos. Em nossa arquitetura este fluxo é interceptado pelo conector *MOD_JK_G*, que provê a comunicação de grupo.

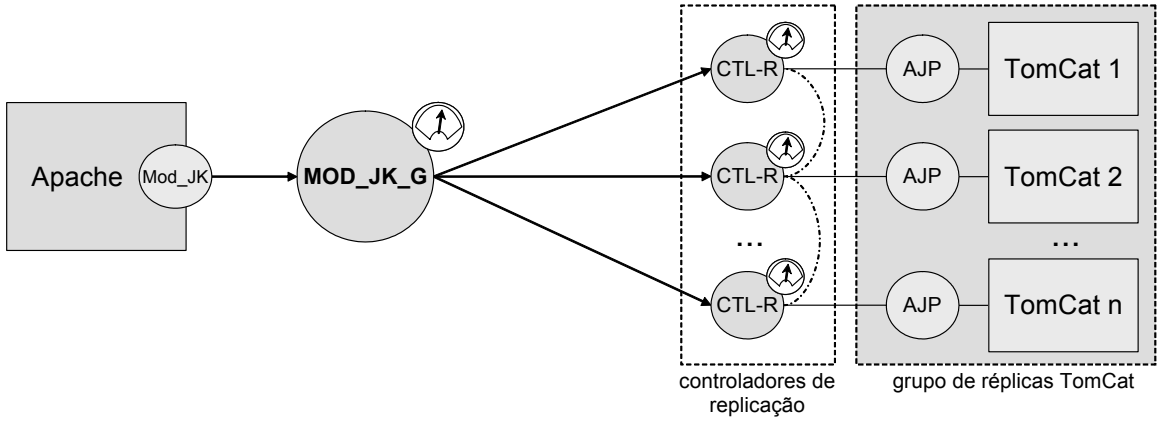


Figura 27 - Arquitetura do exemplo utilizando Tomcat e HTTP Apache

Ao receber uma requisição do Apache o conector *MOD_JK_G* difunde a requisição para os conectores CTL-R de cada réplica *TomCat.* Estes realizam os procedimentos de consistência adequados à técnica de replicação estabelecida, e encaminham a requisição se for apropriado. Por exemplo, no caso de replicação ativa cíclica o CTL-R com a posse da ficha encaminha a requisição para processamento no módulo *TomCat* correspondente, via conector AJP e as demais descartam a mesma. Após processar a requisição, o módulo *TomCat* devolve a resposta ao seu respectivo CTL-R que coloca a mesma no seu caminho de volta. No caso da replicação passiva, o CTL-R da réplica primária repassaria a requisição para o módulo *TomCat*, e enviaria o estado para as réplicas secundárias de acordo com o a propriedade *checkPointInterval*.

5.2.7 Arquitetura e contrato

Uma vez traçada a política de tolerância a faltas, a arquitetura da aplicação é descrita (Listagem 43). As classes de módulo são listadas: *Apache*, que atuará como um cliente, e *TomCat*, que será replicado (linha 1). O mesmo se dá com os conectores específicos utilizados na arquitetura da aplicação (linha 2).

```
1   module Apache, TomCat;
2   connector Mod_JK_G, AJP, CTLRp, CTLRs, CTRLa;
3   module{
4      group TCGroup; // referência ao grupo de réplicas TomCat
5      instantiate Apache as ap, Tomcat as tc; // alocação de módulos
6      join tc to TCGroup;
7      link ap to TCGroup by Mod_JK;
8   }  webApp;
9   start webApp under webContract;
```

Listagem 43 - Descrição da arquitetura de exemplo utilizando Apache e Tomcat

Uma referência para o grupo de réplicas *TomCat* é criada (linha 4), e as referências para as instâncias dos módulos são declaradas. Instâncias de conectores são criadas automaticamente. O módulo *tc* é incluído no grupo *TCGroup* (inicialmente o grupo tem apenas um elemento). Por fim, a topologia é descrita, ligando-se o módulo *ap* aos elementos do grupo *TCGroup* através do conector *Mod_JK_G*. A linha 9 declara que o módulo *webApp* deve ser iniciado sob o contrato *webContract* (descrito adiante). Detalhes da sintaxe e semântica de CBabel podem ser obtidos em (Sztajnberg, 2002) e (Rademaker, 2005). Vale observar que a descrição é declarativa. Ações de configuração são necessárias, em tempo de implantação, para carregar a aplicação de acordo com esta descrição.

O próximo passo é a descrição do contrato que especifica no nível arquitetural os requisitos descritos coloquialmente. No exemplo, cada regra descrita anteriormente dá origem a um serviço da arquitetura contendo uma configuração diferenciada:

passServ, para a replicação passiva com "reserva quente" (*hot standby*) onde uma réplica é eleita primária e apenas esta processa efetivamente as requisições. As réplicas secundárias são atualizadas a cada *checkpoint;*

actCServ, para replicação ativa cíclica, estilo *round-robin*, onde as várias réplicas se revezam como primária, circulando uma ficha. Neste caso não existe atualização de estado baseado em *checkpointing*.

```
1  profile {
 2     Replication.checkPointInterval = 10;
 3     Replication.monitoringInterval = 20;
 4     Replication.timeoutInterval = 200;
 5  } PassiveP;

 6  profile{
 7     Replication.numberOfReplicas = 2;
 8  } PassNRepP;

 9  profile {
10     Faults.numberOfFaults=2;
11     Faults.faultInterval=15;
12 } PassiveFaults;
```

Listagem 44 - Perfis para a replicação passiva

Cada configuração é associada a perfis relacionados às categorias *Replication* e *Faults*. Os perfis para a replicação ativa cíclica foram descritos na Seção 5.2.5. Para a configuração com replicação passiva (Listagem 44) o perfil *PassiveP* descreve que o intervalo de *checkpointing* da réplica primária é de 10s (linha 2), os conectores CTL-R serão monitorados a cada 20 s (linha 3) e a resposta deve ser dada em até 200ms (linha 4). O perfil *PassNRepP* indica que 2 réplicas são necessárias. O perfil *PassiveFaults* indica um máximo de 2 faltas a

cada 15s (linha 8-9). Como *stableInterval* não foi especificada, se as outras propriedades estiverem válidas, a configuração será considerada estável.

Para contemplar o aspecto de tempo de resposta do grupo, uma categoria relacionada à comunicação é utilizada. Os perfis presentes na Listagem 45 que correspondem à categoria *Communicaton* representam os requisitos de tempo de resposta para cada técnica de replicação. O tempo de resposta é monitorado, em nosso exemplo, pelo AR associado ao conector de grupo *MOD_JK_G*.

```
1 category Communication {
2    responseTime: decreasing numeric ms;
3 }
4 profile {
5    Communication.responseTime >= 200 and <= 4000;
6 } comActCP;
7 profile {
8    Communication.responseTime < 200;
9 } comPassP;
```

Listagem 45 - Categoria de comunicação e perfis de tempo de resposta

A Listagem 46 apresenta os perfis (do tipo *Processing*) que representam os requisitos relacionados a processamento para cada técnica replicação. A categoria de processamento possui um AR associado do tipo *SimpleProcessing* que é idêntico ao apresentado na Seção 3.3 e que deverá estar instalado e em execução em cada um dos servidores nos quais se encontra o TomCat.

```
 1 category Processing {
 2    CPUClock: increasing numeric MHz;
 3    TotalMemory: increasing numeric MB;
 4    FreeMemory: increasing numeric MB;
 5    CPUIde: increasing numeric %;
 6 }
 7 profile {
 8    Processing.FreeMemory > 1024MB;
 9    Processing.CPUIdle > 60%
10 } procActCP;
11 profile {
12    Processing.FreeMemory >= 512MB and <= 1024;
13    Processing.CPUIdle >= 0% and <=60%
14 } procPassP;
```

Listagem 46 - Categoria de características de processamento

Neste ponto descreve-se o contrato *webContract* (Listagem 47). Em cada serviço, *passServ* (linha 2-8) e *actCServ* (linha 9-15), estruturas da arquitetura são especializadas para incorporar os elementos arquiteturais da replicação e para associar as mesmas aos perfis adequados. No serviço *passServ* o grupo *TCGroup* passa a ser restrito pelos perfis *PassNRepP, PassiveFaults e PassiveP* (linha 3) e somente será válido se todas as propriedades destes estiverem válidas. Um vetor de módulos *TomCat* é estruturado com o

número de réplicas desejado, sendo que cada instância selecionada é restrita pelo perfil *ProcPassP* (linha 4-5). Em seguida, o módulo é incorporado ao grupo (linha 6). Por último, o módulo Apache, *ap*, é ligado ao grupo *TCGroup* por uma composição dos conectores Mod_JK_G, *CTLRp*, um CTL-R especializado para replicação passiva, e o conector *AJP* para adaptar a interface dos módulos *TomCat.* Esta composição é associada ao perfil *comPassP*, que restringe o tempo de resposta (linha 7).

```
 1  contract{
 2    service{
 3      group TCGroup with PassNRepP, PassiveFaults;
 4      for (i=0; i < PassiveP.Replication.numberOfReplicas; i++) {
 5        instantiate tc[i]=select*(TomCat) with ProcPassP, PassiveP;
 6        join tc[i] to TCGroup;                                    }
 7        link ap to TCGroup by Mod_JK_G > CTLRp > AJP with comPassP;
 8    } passServ;
 9    service{
10      group TCGroup with ActNRepP, ActiveCFaults;
11      for (i=0; i < ActiveCP.Replication.numberOfReplicas; i++ {
12        instantiate tc[i]=select*(TomCat) with ProcActCP, ActiveCP;
13        join tc to TCGroup;                                      }
14      link ap to TCGroup by Mod_JK_G > CTLRa > AJP with comActCP;
15    } actServ;
16    negotiation {
17       not passServ -> (actCServ || out-of-service);
18           actServ  ->  passServ;                    };
19  } webContract;
```

Listagem 47 - Contrato da aplicação de exemplo

A cláusula de negociação efetivamente mapeia os requisitos de tolerância a faltas em uma máquina de estados, que determina a política de implantação, a prioridade e as possíveis transições entre as configurações previamente descritas. Uma vez a aplicação em operação, esta política é gerenciada autonomicamente. O sistema só voltará a sofrer intervenção manual se nenhuma das configurações previstas no contrato puder ser implantada ou mantida. A ordem das regras de negociação (linhas 16-18) determina sua prioridade e o serviço da parte esquerda é aquele a ser implantado e monitorado. A configuração prioritária (linha 17) é a descrita no serviço *passServ*. Se esta não puder ser implantada ou mantida (daí o *not*) porque um dos perfis foi violado, por exemplo, o GC tentará implantar o serviço *actCServ*. Isto é, se o número de faltas aumentar, uma configuração de replicação mais robusta será utilizada. Caso nenhuma das configurações possa ser implantada, um serviço especial, *out-of-service*, é implantado, indicando que a aplicação não pode executar com a qualidade requerida. Já na regra da linha 18, não existe a condição "not". Isso quer dizer que se a configuração atual é a do serviço *actCServ* (ou seja os perfis deste serviço estão válidos), e o serviço *passServ* Listagem 47puder ser implantado, ou seja, os perfis deste serviço também são válidos, então a transição para este é incondicional. Com isso podemos expressar o requisito de retornar o

sistema a uma técnica de replicação que exija menos recursos (com menos réplicas). A cláusula de negociação da Listagem 47 pode ser representada pela máquina de estados da Figura 28.

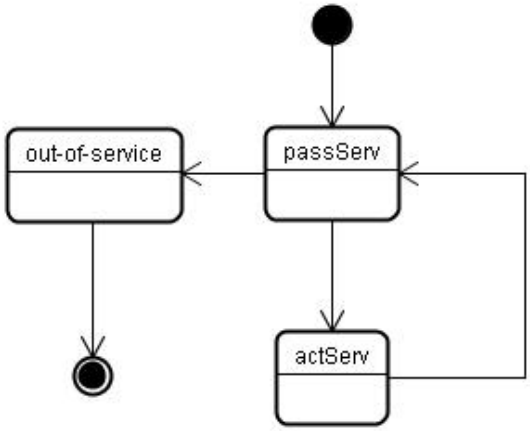


Figura 28 - Máquina de estado de negociação

O serviço preferencial é o de replicação passiva (*passServ*). O sistema pode efetuar uma transição da técnica de replicação passiva para a técnica de replicação ativa (*actServ*) e desta técnica de volta para a replicação passiva. Adicionalmente, pode haver uma transição do estado *passServ* para o estado *out-of-service* que representa o término do processo de adaptação efetuado pela infra-estrutura do CR-RIO.

A construção *select* indica que o módulo específico vai ser selecionado dinamicamente através do SD. A consulta ao SD é parametrizada pela classe do módulo (*TomCat*, no exemplo) e pelos perfis associados à declaração *instantiate*. A Listagem 48 apresenta o mapeamento da construção *select* da linha 5 e os perfis associados *ProcPassP* e *PassiveP* em uma consulta ao SD. O perfil *ProcPassP* apresentado na Listagem 46 está associado ao recurso *Processing* presente na Listagem 48 e o perfil *PassiveP* (Listagem 44) estão associados ao recurso

```
1 <ResourceQuery Type="TomCat">
2   <Constraints From="Processing">
3     <Attribute Name="FreeMemory" op="between" Value="512" Value="1024"/>
4     <Attribute Name="CPUIdle" op="between" Value="0" Value="60"/>
5   </Constraints>
6   <Constraints From="PassiveP">
7     <Attribute Name="timeoutInterval" op="==" Value="200"/>
8   </Constraints>
9 </ResourceQuery>
```

Listagem 48 - Consulta ao SD para obter um servidor TomCat com replicação passiva

O uso do símbolo "*" na construção *select* indica que o SD será continuamente monitorado e se outra instância, mais apta, da classe requerida estiver disponível, o módulo

atual será substituído. Com isso é possível realizar reparos, localizados e atômicos (Cardoso et al., 2006), na configuração sem a necessidade da intervenção do GR. Por exemplo, no caso iminente de uma falha em um nó, monitorado pelo *timeoutInterval*, a referência de um novo módulo, mais apto pode ser descoberta, substituindo o módulo atual. Com isso evita-se que a seqüência de notificações de perfil e serviço inválidos, e que uma nova negociação, e implantação de outra configuração sejam disparadas.

5.2.8 Implantação e aspectos de implementação

Para validar a abordagem e o contrato *webContract*, desenvolvemos alguns componentes específicos da aplicação e integramos os mesmos à infra-estrutura previamente desenvolvida (Corradi, 2005) e (Santos, 2006). Desenvolvemos em Java as classes do conector de grupo *MOD_JK_G,* os conectores CTL-R e customizamos as classes abstratas de ARs para as categorias *Replication e Faults*.

A comunicação original entre o Apache e o Tomcat se dá através do conector Mod_JK (Apache, 2007c), que encaminha as requisições para o conector AJP, componente padrão do Tomcat. Em nossa solução, o conector *MOD_JK_G* é interposto neste caminho para receber do Mod_JK as requisições do Apache e realizar a comunicação de grupo, encaminhando estas requisições para o grupo de CTL_Rs. Foi necessário também contemplar os protocolos específicos do Mod_JK e do AJP dentro do código. A Figura 29 apresenta o diagrama de interações simplificado desta composição de elementos. A interação *1.2.1.1.1* é justamente a comunicação de *1:n* e o retorno correspondente é realizado de *n:1*.

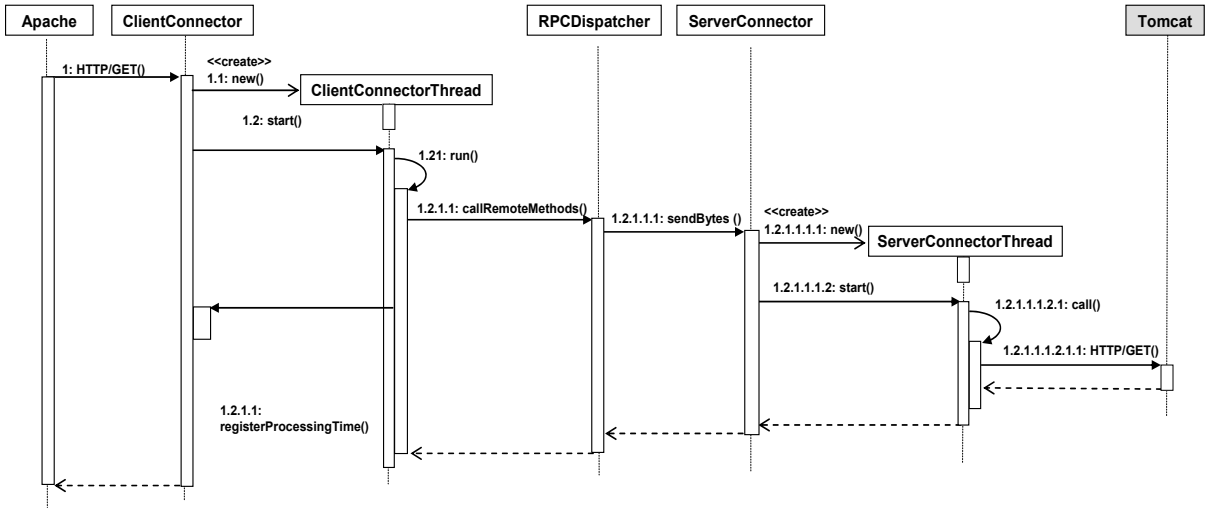


Figura 29 - Diagrama de interação dos conectores MOD_JK_G > CTL-R > AJP

A comunicação de grupo foi implementada com o pacote JGoups (Ban, 2007), através da classe *RPCDispatcher*, que provê um mecanismo de invocação dinâmica no cliente e a chamada de procedimentos nos servidores remotos (um pouco mais complexo que um RPC para grupo). Observe no trecho de código da Listagem 49 que o segundo parâmetro, "*sendBytes",* é o nome do método remoto a ser invocado em cada réplica, *buffer* contém os dados encaminhados pelo Apache, *class* é um vetor com os tipos dos parâmetros em *buffer* (usado para remontar a invocação por reflexão no lado remoto).

```
1 RspList rsp_list = disp.callRemoteMethods((Vector) null, "sendBytes",
2       new Object[] { buffer }, new Class[] { byte[].class },
3       GroupRequest.GET_ALL, 0L);
```

Listagem 49 - Envio de dados através da classe RPCDispatcher da API JGroups

O lado "servidor" (*ServerConnector*, na Figura 29) encapsula a funcionalidade dos conectores CTL-Rs, implementando as características específicas de cada técnica de replicação, e faz interface com o conector AJP. Foi necessário adaptar esta interface ao esquema de reuso de conexões abertas entre o Apache e o Tomcat. Para isso fizemos uso de *threads*, através do pacote JNIO (Sun, 2007). O suporte provido pelo JNIO para o uso de *threads* e para chamadas de E/S não bloqueantes é mais escalável. Para construir as especializações de CTL-R para cada técnica de replicação optou-se por usar do padrão *strategy* (Gamma et al., 1995) ao invés de classes separadas.

Construímos as técnicas de replicação ativa e ativa cíclica. Um teste simples de desempenho foi realizado com a ferramenta JMeter (JMetter, 2007), também utilizada nos testes de desempenho da implementação de referência dos serviços propostos, para "estressar" o servidor Apache. Neste teste, configuramos o JMeter para simular 10 usuários simultâneos solicitando documentos de 8Kbytes a duas réplicas Tomcat executando na mesma máquina, com replicação ativa. O tempo medido no teste foi aproximadamente igual a 4s, contra aproximadamente 400ms do teste realizado com apenas uma instância do Tomcat e sem a infra-estrutura de replicação (ordem de grandeza similar a encontrada em (Favarim, 2004)). Além de concluir e refinar a implementação, mais testes podem ser realizados em cenários distribuídos, como pede o exemplo, também considerando o número de adaptações realizadas. Com isso será também possível detectar limitações da abordagem. Por outro lado, a aplicação implementada permitiu validarmos o uso dos serviços de Descoberta e Contexto propostos nesta dissertação.

CAPÍTULO 6 CONCLUSÕES

6.1 A proposta

Foi proposto um modelo para elementos importantes usados no suporte de aplicações cientes de contexto: (i) Serviço de Descoberta, que permite a descoberta dinâmica de recursos, considerando restrições de contexto nesta descoberta; (ii) um Serviço de Contexto, que coletam informações de recursos das aplicações e do ambiente de execução. A integração destes serviços com aplicações pervasivas e ubíquas é baseada em descrições de meta-nível das classes de recursos, as quais são registradas em um Serviço de Registro e Diretório, e em um protocolo de interação independente de plataforma e de mecanismos de comunicação. Estes serviços são facilmente integrados a infra-estruturas de *middleware* devido à sua abordagem reflexiva.

Baseado no modelo, uma implementação de referência foi desenvolvida. Além de classes abstratas e descrições em XML, são empregadas boas técnicas de engenharia de software e padrões de projeto. Todos os serviços são implantados como Serviços Web e a interação entre eles baseada em SOAP. Isso possível, mais uma vez, usando mecanismos de extensão, conferindo independência aos serviços. Em particular foi desenvolvido um padrão de projeto para os Agentes de Recursos, o ponto de extensão mais freqüentemente utilizado, de forma que novos recursos podem ser integrados com certa facilidade à infra-estrutura. O programador precisa montar uma descrição e implementar a interface *ResourceAgent*. A implementação de referência disponibiliza a classe abstrata *AbstractResourceAgent* que disponibiliza pontos de extensão cujo objetivo é facilitar o desenvolvimento de novos Agentes de Recursos através do uso do padrão de projeto *Template Method* largamente utilizado.

A proposta dos serviços e a construção dos mesmos foram validadas através de exemplos. Alguns Agentes de Recursos foram implementados, entre eles agentes para processamento (CPU, memória e disco), para *loggers* de temperatura da empresa iButton da empresa Maxim, e sensores SunSPOT da SunMicrosystems (contendo sensores de luminosidade e acelerômetros). Enquanto o presente texto estava sendo finalizado, agentes para equipamentos médicos da TelcoMed e equipamentos de automação residencial da X10 também estavam sendo integrados. Estes agentes foram utilizados nas aplicações de tele-saúde e tolerância a faltas e projetados para usar os Serviços de Descoberta e Contexto. Cada

exemplo, de domínios diferentes, utilizou os serviços propostos de maneiras diferentes. Realizamos ainda alguns experimentos para medir o desempenho da implementação com relação a tempo de resposta e escalabilidade. Para isso utilizamos o JMeter e Agentes de Recursos de testes, que não realizam efetivamente um sensoriamento de recursos, mas respondem à interface *ResourceAgent*. A avaliação que temos do uso dos serviços como infra-estrutura de suporte a aplicações cientes de contexto, bem como seu desempenho é satisfatória. A maior dificuldade em fazer extensões aos serviços, mais especificamente a criação de um novo Agente de Recursos, está na programação dos acionadores ou características específicas dos recursos a serem monitorados. Este tipo de dificuldade, em realidade, está ligado à falta da adoção, por parte dos fornecedores, de uma padronização para tratar o problema da monitoração, como discutimos anteriormente.

Algumas das contribuições deste trabalho foram publicadas no *26º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos* (SBRC'09) (Rodrigues, 2009a), no *IEEE/IFIP 7th International Conferences on Embedded and Ubiquitous Computing* (EUC-09) (Rodrigues, 2009b) e no *International Workshop on Ubiquitous Computing, Management, and Embedded Applications in Healthcare* (UbiHealth-09) (Rodrigues, 2009c).

6.2 Trabalhos futuros

Como trabalho de continuação mais imediata, alguns aprimoramentos podem ser realizados na implementação de referência. O primeiro seria especializar o Serviço de Registro e Diretório para utilizar o LDAP ou um banco de dados relacional. Com isso informações de contexto poderiam ser armazenadas por prazos longos de tempo, permitindo o uso da infra-estrutura proposta em um serviço de apoio a gerência de recursos. O segundo aprimoramento seria ampliar os testes de desempenho para avaliar mais amplamente a escalabilidade dos serviços. Além destes aprimoramentos gostaríamos de desenvolver outras aplicações com o suporte dos serviços propostos e explorar melhor o mecanismo de federação já contemplado no modelo e na implementação de referência.

Como proposta para continuação deste trabalho, gostaríamos de explorar duas linhas de investigação. Primeiro, o uso de ontologias para organizar o Serviço de Registro e Diretório e aprimorar a capacidade de mineração do Serviço de Descoberta. Com o uso de ontologias, classes descritas de formas diferentes poderiam ser semanticamente relacionadas,

ampliando as opções de seleção de recursos de uma determinada classe. A idéia seria adicionar uma camada de ontologia ao Serviço de Registro e Diretório. A outra linha é relacionada à heurística de seleção. Ao ser consultado, o Serviço de Descoberta retorna uma lista de recursos da classe requerida que atendem às restrições de contexto. Cabe ao cliente selecionar o recurso, o que pode ser feito ao acaso ou utilizando uma heurística simples, como ordenar os recursos selecionados segundo o valor de uma das propriedades. Uma solução mais interessante seria o emprego de funções de utilidade, por exemplo, que poderiam ser guiadas por restrições de contexto determinadas na consulta ao Serviço de Diretório, acrescida de informações de peso ou prioridade para determinadas propriedades do recurso sendo descoberto. Uma camada de software adicional poderia ser adicionada para fornecer mais este serviço. Uma possibilidade de integração deste tipo de funcionalidade foi apresentada na aplicação de tolerância a falhas, Seção 5.2, através do mecanismo de suporte à construção *select* da descrição arquitetural em CBabel.

Além destas linhas, identificamos ao longo do desenvolvimento deste trabalho uma oportunidade de ampliar a funcionalidade dos serviços propostos. Através deles é possível descrever e monitorar recursos diversos, mas não conseguimos atuar ou utilizar serviços destes recursos. A integração dos dispositivos X10 na aplicação de tele-saúde é um exemplo deste ponto. Os ARs desenvolvidos possibilitam a obtenção da informação se determinado ponto de luz está ligado ou desligado e qual a intensidade da iluminação, mas não oferece o serviço para que a aplicação interaja com o dispositivo para ligar ou desligar a luz. Desta forma o programador deve utilizar diretamente os acionadores do dispositivo ou métodos do recurso, diminuindo as possibilidades de reuso. Uma extensão no modelo para descrever recursos poderia adicionar a descrição dos serviços oferecidos e um elemento similar ao AR (ou o próprio AR poderia cuidar de acionar os recursos quando a solicitação de um serviço fosse recebida), como se dá em sistemas de middleware para objetos distribuídos como CORBA, RMI ou mesmo Serviços Web. Desta forma a aplicação poderia utilizar a mesma infra-estrutura de suporte a aplicações com restrições de contexto para a descoberta de recursos que oferecessem funções com determinadas características e também utilizar os mecanismos semelhantes para acionar e monitorar o contexto dos recursos.

REFERÊNCIAS

Apache.org (2007a). Apache HTTP Server Project. Acesso em novembro de 2007. Disponível em: http://httpd.apache.org.

Apache.org (2007b). Apache Tomcat. Acessado em novembro de 2007. Disponível em: http://tomcat.apache.org.

Apache.org (2007c). The Apache Tomcat Connector - Reference Guide. Acesso em novembro de 2007. Disponível em: http://tomcat.apache.org/connectors-doc/reference/apache.html.

ATSP. Association of Telehealth Service Providers. Acesso em fevereiro de 2009. Disponível em: http://www.atsp.org/.

Baker M, Smith G. Jini meets the grid. Proceedings of IEEE International Conference on Parallel Processing Workshop; setembro de 2001; Valencia, Espanha. p. 193-198.

Balakrishnan D, Barachi M, Karmouch A, Glitho R. Challenges in modeling and disseminating context information in ambient networks. Proceedings of 2º International Workshop on Mobility Aware Technologies and Applications; outubro de 2005; Montreal, Canadá. p. 32-42.

Baldauf M, Dustdar S, Rosenberg F. A survey on context-aware systems. International Journal of Ad Hoc and Ubiquitous Computer; junho de 2007; v. 2. p. 263-277.

Ban B et al. JGroups - A toolkit for reliable multicast communication. Acesso em outubro de 2007. Disponível em: http://www.jgroups.org/javagroupsnew/docs/index.html.

Bardram J. The Java Context Awareness Framework (JCAF) – A service infrastructure and programming framework for context-aware applications. Lecture Notes in Computer Science (LNCS); maio de 2005. v. 3468. p. 98-115.

Brownsell S, Aldred H, Hawley MS. The role of telecare in supporting the needs of elderly people. Journal of Telemedicine and Telecare; setembro de 2007. v. 13. p. 293-297(5).

Capra L, Zachariadis S, Mascolo C. Q-CAD: QoS and context aware discovery framework for mobile systems. Proceedings of IEEE International Conference on Pervasive Services (ICPS'05); julho de 2005; Santorini, Grécia. p. 453-456.

Cardoso, LT. Integração de serviços de monitoração e descoberta de recursos a um suporte para arquiteturas adaptáveis de software [dissertação]. Niterói: Universidade Federal Fluminense. Instituto de Computação; 2006.

Cardoso LT, Sztajnberg A, Loques OG. Self-adaptive applications using ADL contracts. Proceedings of 2º IEEE International Workshop on Self-Managed Networks Systems & Services; 2006, Dublin, Irlanda. LNCS, vol 3996, p. 87-101.

Chen G, Kotz D. A survey of context-aware mobile computing research. Technical report TR2000-381. Department of Computer Science. Dartmouth College; novembro de 2000.

Cheng S, Garlan D, Schmerl B, Sousa J, Spitznagel B, Steenkiste P. Using architectural style as a basis for system self-repair. Proceedings of 3º IEEE/IFIP Conference on Software Architecture: System Design, Development and Maintenance; agosto de 2002; Deventer, Holanda. p. 45-59.

Cheng S, Huang A, Garlan D, Schmerl B, Steenkiste P. Rainbow: architecture-based self-adaptation with reusable infrastructure. Proceedings of the International Conference on Autonomic Computing; maio de 2004; Washington, USA. p. 276-277.

Copetti A, Loques O, Leite J, et al. Intelligent context-aware monitoring of hypertensive patients, situation recognition and medical data analysis. Pervasive Health Environments. 2009. Londres.

Corradi A. Um framework de suporte a requisitos não-funcionais de nível alto [dissertação]. Niterói: Universidade Federal Fluminense. Instituto de Computação; 2005.

Curty R. Uma proposta para descrição e implantação de contratos para serviços com qualidade diferenciada [dissertação]. Niterói: Universidade Federal Fluminense. Instituto de Computação; novembro de 2002.

Dallas Maximm Semiconductor. iButton: Touch the Future. Acesso em: março de 2007. Disponível em: http://www.maxim-ic.com/products/ibutton/

DAML. The DARPA Agent Markup Language. Acesso em: abril de 2009. Disponível em: http://www.daml.org.

Dey A. Providing architectural support for context-aware applications. [tese]. Georgia Institute of Tecnology; novembro de 2000.

Droms R. Dynamic Host Configuration Protocol. IETF, RFC 2131, 1997. Acesso em: abril de 2008. Disponível em: http://www.ietf.org/rfc/rfc2131.txt.

Fahy P, Clarke S. CASS - a middleware for mobile context-aware applications. Proceedings of Workshop on Context Awareness – MobiSys; junho de 2004; Massachusetts, USA.

Favarim F, Fraga J, Lung LC, Siqueira F. Suporte de tolerância a faltas adaptativa para aplicações desenvolvidas em CCM. Anais do Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC'04), SBC / LARC; 2004; Gramado, Brasil.

Foster I. The anatomy of the grid: enable scalable virtual organizations. International Journal of High Performance Computing Applications. 2001. v. 15 nº 3. p. 200-222.

Frølund S, Koistinen J. Quality-of-service specifications in distributed object systems. Proceedings of IEEE Distributed Systems Engineering. 1998. nº 5. p. 179-202.

Furmento N, Lee W, Mayer A, Newhouse S, Darlington J. ICENI: an open grid service architecture implemented with Jini. Proceedings of ACM/IEEE Conference on High Performance Networking and Computing. 2002. p. 1-10.

Gaia. Active Spaces for Ubiquitous Computing. Acesso em: abril de 2009. Disponível em: http://gaia.cs.uiuc.edu.

Gamma, E, et al Design Patterns - Elements of Reusable Object-Oriented Software. Addison Wesley, 1ª edição. 1995.

Ganglia. Ganglia Monitoring System. Acesso em: dezembro de 2008. Disponível em: http://ganglia.info.

Garlan D, Cheng SW, Huang AC, Schermerl B, Steenkiste P. Rainbow: architecture-based self-adaptation with reusable infrastructure. Computer. IEEE Computer Society Press; outubro de 2004; Los Alamitos, USA. vol 37, nº 10, p 46-54.

Google. Acesso em: janeiro de 2008. Disponível em: http://www.google.com

Gorender S, Cunha PR, Macedo, RJ. The implementation of a distributed system model for

fault tolerance with QoS. Anais do 23º Simpósio Brasileiro de Computadores (SBRC05); 2005; Fortaleza, Brasil. p. 827-840.

Gu T, Pung HK, Zhang DQ. A middleware for building context-aware mobile services. Proceedings of Vehicular Technology Conference; maio de 2004. v. 5. p. 2656-2660.

Guttman E, Perkins C, Kempf J. Service Templates and Service: Schemes. IETF, RFC 2609, 1999. Acesso em: abril de 2007. Disponível em: http://tools.ietf.org/html/rfc2609.

Hofer T, Schwinger W, Pichler M, Leonhartsberger G, Altmann J, Retschitzegger W. Context-awareness on mobile devices - the hydrogen approach. Proceedings of the 36º Annual Hawaii International Conference; janeiro de 2003; Havaii, USA. p 10.

Jalote P. Fault Tolerance in Distributed System. Prentice-Hall. 1994.

JAX-WS. Sun Microsystems. Java API for XML Web Services (JAX-WS). Acesso em: julho de 2008. Disponível em: https://jax-ws.dev.java.net/.

JBoss.org. JBoss Cache. Acesso em: maio de 2007. Disponível em: http://labs.jboss.com/jbosscache/

JEE. Sun Microsystems. Java EE: Do more with less work. Acesso em: maio de 2008. Disponível em: http://java.sun.com/javaee/.

Jini. Jini Specifications and API Archive. Acesso em: abril de 2009. Disponível em: http://java.sun.com/products/jini.

JMetter. Apache.org (2007d). Apache JMeter. Acesso em: fevereiro de 2008. Disponível em: http://jakarta.apache.org/jmeter.

Kalbarczyk Z, Iyer RK, Wang L. Application fault tolerance with armor middleware. IEEE Internet Computing; março e abril de 2005. v. 9. nº. 2. p. 28-37.

Keane J, Trent J. RXTX API. Acesso em: maio de 2008. Disponível em: http://www.rxtx.org/.

Korpipää P, Mantyjarvi J, Kela J, Keranen H, Malm EJ. Managing context information in mobile devices. IEEE Pervasive Computer. julho de 2003. v 2. p. 42-51.

Lacoste M, Privat G, Ramparany F. Evaluating confidence in context for context-aware security. Ambient Intelligence. LNCS; novembro de 2007; v. 4794. p. 211-229.

Lindholm T, Yellin F. The Java Virtual Machine Specification. Prentice Hall PTR. 2a Edição. 1999.

Lisbôa JC, Loques OG. Um padrão arquitetural para gerenciamento de qualidade de serviço em sistemas distribuídos. In: 4º Latin American Conference on Pattern Languages of Programming, SugarLoafPLoP; 2004; Fortaleza, Brasil.

Loques O, Sztajnberg A, Ansaloni, S, Cerqueira RC. A contract-based approach to describe and deploy non-functional adaptations in software architectures. Journal of the Brazilian Computer Society; 2004; Porto Alegre, Brasil. v. 10. nº 1. p. 5-18.

Loques O et al. Aplicando técnicas de computação ubíqua a aplicações de tele-saúde. Projeto de Pesquisa. Edital Programa Prioridade Rio - Apoio ao estudo de temas prioritários para o Governo do Estado do Rio de Janeiro. 2007-2009.

Loyall JP, Schantz RE, Zinky JA, Bakken DE. Specifying and measuring quality of service in distributed object systems. Proceedings of 1º IEEE International Symposium on Object-Oriented Real-Time Distributed Computing; abril de 1998; Kyoto, Japão. p. 43.

Lung LC, Favarim F, Santos GT, Correia, MP. An infrastructure for adaptive fault tolerance on FT-CORBA. Proceedings of 9º IEEE International Symposium on Object-oriented Ral-time Distributed Computing; 2006; Gyeongju, Coréia do Sul. V. 1. p. 504-511.

Lung LC, Padilha R. GroupPac. Acesso em setembro de 2007. Disponível em: http://sourceforge.net/projects/grouppac.

Massie, ML, Chun, BN, Culler, DE. The ganglia distributed monitoring system: design, implementation, and experience. Parallel Computing. julho de 2004. v. 30, p. 817-840.

Mühl G, Ulbrich A, Herrmann K, Weis T. Disseminating information to mobile clients using publish-subscribe. IEEE Internet Computing. junho de 2004. p. 46-53.

MyProxy. Acesso em: julho de 2009. Disponível em: http://grid.ncsa.illinois.edu/myproxy.

OSGi. The Dynamic Module System for Java. Acesso em: fevereiro de 2009. Disponível em: http://www.osgi.org/Main/HomePage.

Pal P, Loyall J, Schantz R, Zinky J, Shapiro R, Megquier J. Using QDL to specify QoS aware distributed (QuO) application configuration. Proceedings of 3º IEEE International Symposium on Object-Oriented Real-Time Distributed Computing; março de 2000; Newport Beach, USA. p. 310-319.

Pascoe J. Adding generic contextual capabilities to wearable computers. Wearable Computers. Digest of Papers. Proceedings of Second International Symposium; outubro de 1998. p.92-99.

Rademaker A. Uma semântica em lógica de reescrita para a ADL CBabel. Niterói: Universidade Federal Fluminense. Instituto de Computação; 2005.

Rialle V, Duchene F, Noury N, Bajolle L, Demongeot J. Halth “smart” home: information technology for patients at home. Telemedicine Journal and e-Health. dezembro de 2002. p. 395-409.

RMI. Remote Method Invocation. Acesso em: dezembro de 2008. Disponível em: http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp.

Rodrigues ALB, Sztajnberg A, Loques O. (2008a) Auto-adaptação de requisitos de tolerância a faltas através de contratos. Anais do 28º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC’08); maio de 2008; Rio de Janeiro, Brasil. v. 1. p. 973-986.

Rodrigues ALB, Bezerra LN, Sztajnberg A, Loques O. (2009a) Self-adaptation of fault tolerance requirements using contracts. Proceedings of IEEE/IFIP 7º International Conferences on Embedded and Ubiquitous Computing (EUC-09); agosto de 2009; Vancouver, Canadá.

Rodrigues ALB (2009C) et al. Using discovery and monitoring services to support context-aware remote assisted living applications. Proceedings of International Workshop on Ubiquitous Computing, Management, and Embedded Applications in Healthcare (UbiHealth-09): Strategies and Application Case Studies; agosto de 2009; Vancouver, Canadá.

Rodrigues, ALB (2009D). Context Discovery Resource Framework. Acesso em: maio de 2008. Disponível em: http://www.ime.uerj.br/~albr.

Rodrigues, ALB. Uma interface para gerência e manutenção de serviços JINI [monografia]. Rio de Janeiro: Universidade do Estado do Rio de Janeiro. Instituto de Matemática e Estatística. 2003.

Roman M, Hess C, Cerqueira R, Ranganathan A, Campbell RH, Nahrstedt K. A middleware infrastructure for active spaces. IEEE Pervasive Computing. 2002.v.1, p. 74-83.

RPC. Remote Procedure Call. Acesso em: dezembro de 2008. Disponível em: http://tools.ietf.org/html/rfc707.

Sacramento V, Endler M, Rubinsztejn H, Lima L, Gonçalves K, Nascimento F, Bueno G. MoCA: A middleware for developing collaborative applications for mobile users. IEEE Distributed Systems Online; outubro de 2004. v.5, p. 2.

Santos ALG, Leal DA, Loques O G (2006) Um suporte para adaptação dinâmica de arquiteturas ubíquas. In: XXXII Conferência Latinoamericana de Informática; 2006; Santiago, Chile.

Schilit B, Adams N, Want R. Context-aware computing applications. Proceedings of IEEE 1º Workshop on Mobile Computing Systems and Applications; dezembro de 1994; Santa Cruz, USA. p. 85-90.

Schmidt A, Beigl M, Gellersen HW. There is more to context than location. Computers and Graphics; dezembro de 1999. v. 23. p. 893-901.

Sheng QZ, Benatallah B. ContextUML: a UML-based modeling language for model-driven development of context-aware Web services. Proceedings of IEEE International Conference on Mobile Business; julho de 2005. p. 206-212.

Stefanov DH, Bien Z, Bang W. The smart house for older person sans persons with physical disabilities: structure, technology arrangements, and perspectives. IEEE Transactions on Neural Systems and Rehabilitation Engineering; junho de 2004. v. 12. p. 228-250.

Strang T, Linnhoff-Popien C. A context modeling survey. 1º International Workshop on Advanced Context Modeling, Reasoning and Management; setembro de 2004; Nottingham, England.

Sun Microsystems, Inc. New I/O APIs. Acesso em julho de 2007. Disponível em: http://java.sun.com/j2se/1.5.0/docs/guide/nio.

SunSPOT. Sun Microsystems. Sun SPOT – Small Programmable Object Technology. Acesso em Mario de 2009. Disponível em: http://www.sunspotword.com.

Sztajnberg A. Flexibilidade e separação de interesses para a concepção e evolução de aplicações distribuídas [tese]. Rio de Janeiro: UFRJ/ COPPE/PEE. Maio de 2002.

Telcomed. WristClinic™ – The all-in-one wireless remote medical monitoring revolution, A Medic4All Group Company. Acesso em novembro de 2007. Disponível em: http://www.telcomed.ie/index.html.

Tierney B et al, A grid monitoring architecture. Tech. Rep. GWD-PERF-16-2. Global Grid Forum. janeiro de 2002.

UML. Unified Modeling Language. Acesso em: março de 2009. Disponível em: http://www.uml.org.

Veizades J, Guttman E, Perkins C, Kaplan S. Service Location Protocol. IETF, RFC 2165. Acesso em: junho de 1997. Disponível em: http://www.ietf.org/rfc/rfc2165.txt.

W3C1. Extensible Markup Language. Acesso em: dezembro de 2008. Disponível em: http://www.w3c.org/XML.

W3C2. WebServices Activity. Acesso em: dezembro de 2008. Disponível em: http://www.w3c.org/2002/ws.

W3C3. Resource Description Framework. Acesso em dezembro de 2008. Disponível em: http://www.w3.org/RDF.

W3C4. RDF Vocabulary Description Language 1.0: RDF Schema. Acesso em: fevereiro de 2008. Disponível em: http://www.w3.org/TR/rdf-schema.

W3C5. Naming and Addressing: URIs, URLs. Acesso em: novembro de 2007. Disponível em: http://www.w3.org/Addressing.

Wahl M, Howes T, Kille S. Lightweight Directory Access Protocol. IETF, RFC 2251. Acesso em: abril de 2009. Disponível em: http://www.ietf.org/rfc/rfc2251.txt.

Weiser M. The computer for the 21st century. Scientific American. Setembro de 1991.

Wolski R, Spring T N, Hayes J. The network weather service: a distributed resource performance forecasting service for metacomputing. Proceedings of Future Generation Computer Systems; 1997; San Jose, USA. vol. 15, n. 5-6. p. 757-768.

X10. Active Home Professional. Acesso em: dezembro de 2007. Disponível em: http://www.activehomepro.com/.