UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"

FCT – Faculdade de Ciências e Tecnologia
DMEC – Departamento de Matemática, Estatística e Computação
Bacharelado em Ciência da Computação

Trabalho de Conclusão de Curso
(Modalidade Trabalho Acadêmico)

Arquitetura Orientada a Serviços baseada na Tecnologia Jini: um caso de teste em Computação Ubíqua.

Prof. Dr.: Ronaldo Celso Messias Correia

Autor: Lucas Roberto Thomaz

2010

LUCAS ROBERTO THOMAZ

Arquitetura Orientada a Serviços baseada na Tecnologia Jini: caso de teste em Computação Ubíqua.

Monografia apresentada ao Departamento de Matemática, Estatística e Computação (DMEC), da Faculdade de Ciências e Tecnologia (FCT), UNESP, como parte das atividades da disciplina Trabalho de Conclusão de Curso, necessária para a para obtenção do título de Bacharel em Ciências da Computação.

Orientador: Prof. Dr. Ronaldo Celso Messias Correia

Presidente Prudente
2011

Dedicatória

Dedico esse trabalho aos meus pais Luiz e Beatriz, meus irmãos Vitor e Mariana, minha namorada Fernanda, meus familiares e meus amigos. Foram indispensáveis na minha vida acadêmica e pessoal, me apoiando e me dando forças para seguir em frente.

Agradecimentos

Agradeço: Luiz e Beatriz, que me guiaram e me apoiaram pelas estradas da vida, sempre me aconselhando e me dando conforto em difíceis escolhas que tenha feito, sem eles, nada disso seria possível, os quais tenho orgulho de dizer que são meus Pais.

Meus irmãos, Vitor e Mariana, com quem compartilho momentos felizes e que são integrantes da maioria das minhas boas recordações, desde sempre.

Meus familiares, que sempre estiveram ao meu lado, me ajudando e torcendo por mim.

Minha namorada, Fernanda, que considero a minha maior conquista durante a graduação, foi parte essencial na entrega desse trabalho, me orientando e confortando, durante todo o processo.

Meus amigos, Ricardo Birigui, Éverton Toicinho, Diogogomes, Ademarcos, Fernando Codorna, Allampada, Moller da Bateria e Rafael Kaótico, com quem morei junto por tantos anos, e aos amigos, Caio Mogle, Gustavo Dida da Bateria, Felcarlos e Pedro Doido, com quem criei a mais pura forma de amizade, aquela que aprende nas diferenças a respeitar o limite de cada um e é a eles que dedico o meu maior aprendizado, aquele que não são livros que trazem, a eles dedico o meu maior crescimento pessoal.

Tantos outros amigos feitos na universidade, os quais será impossível nomear aqui, mas que certamente foram peças importantes para o quebra-cabeça que foi a minha graduação. A memória de Alexandre Roberto Wakabayashi.

Amigos da minha cidade natal, Botucatu, que estiveram sempre junto comigo, mesmo com a distancia de 400 quilômetros que nos separavam.

Meu orientador Ronaldo Celso Messias Correia, pelo apoio, dedicação e conhecimento, sem o qual, esse trabalho não seria possível.

Meus professores, por todo o conhecimento que compartilharam comigo e que me fizeram sair da Universidade melhor do que entrei.

Índice

Índice iii
Lista de Figuras vi

1 Introdução **1**
1.1 Considerações Iniciais 1
1.2 Objetivo 1
1.3 Organização do Trabalho 2

2 Computação Ubíqua **3**
2.1 Apresentação 3
2.2 Características Principais 4
2.3 Desafios da Computação Ubíqua 5
2.4 Aplicações 5
2.4.1 Ensino 6
2.4.2 Trabalho Colaborativo 6
2.4.3 Aplicativos que auxiliam a movimentação do usuário . . . 6
2.4.4 Residência Inteligente 6

3 Arquitetura Orientada a Serviços **7**
3.1 Introdução 7
3.2 Conceitos Básicos 8
3.2.1 Serviços 8
3.2.2 Interoperabilidade 8
3.2.3 Baixo Acoplamento 9
3.3 Ingredientes 9
3.3.1 Infraestrutura 9
3.3.2 Arquitetura 9
3.3.3 Processos 9
3.3.4 Governança 9
3.4 Serviços 10
3.4.1 Reusabilidade 11
3.4.2 Abstração de lógica de negócio 11
3.4.3 Composição 11

3.4.4 Autonomia . . . 11
3.4.5 Capacidade de ser descoberto . . . 11
3.4.6 Classificação . . . 12
3.4.7 Serviços Básicos . . . 12
3.4.8 Serviços Compostos . . . 12
3.4.9 Serviços Processo . . . 13
3.5 Segurança . . . 13

4 Jini **14**
4.1 Introdução . . . 14
4.1.1 Componentes . . . 15
4.1.2 Registro de Serviço . . . 15
4.1.3 Busca do Cliente . . . 17
4.2 Proxies . . . 17
4.3 Lookup Service . . . 18
4.3.1 Descoberta Unicast . . . 18
4.3.2 Descoberta Multicast . . . 19
4.3.3 Grupos . . . 19
4.4 Registrando um serviço . . . 21
4.5 Busca do Cliente . . . 22
4.6 Leasing . . . 23
4.7 Métodos Remotos . . . 24

5 Caso de Teste **25**
5.1 Sistema . . . 26
5.1.1 Usuários do Sistema . . . 27
5.2 Implementação . . . 27
5.2.1 Modelagem do sistema de Gerência do Campeonato de Futebol . . . 28
5.2.2 Base de dados . . . 29
5.2.3 Serviços e Clientes . . . 30
5.3 Definição e criação das Classes . . . 32
5.3.1 Classes do sistema de gerência e suas interfaces gráficas . . . 34
5.3.2 Classes dos Serviços e Clientes . . . 39
5.4 Funcionamento do sistema . . . 48
5.4.1 Iniciando o Servidor . . . 49
5.4.2 Iniciando os Serviços . . . 49
5.4.3 Iniciando os Clientes . . . 49
5.4.4 Utilizando o cliente . . . 51

6 Considerações finais **53**
6.1 Dificuldades Enfrentadas 54
6.2 Perspectivas Futuras 54

Lista de Figuras

3.1 Serviços encapsulando vários conceitos lógicos. Adaptado de ERL (2005). 10
3.2 Camadas fundamentais dos serviços. Adaptado de JOSUTTIS (2007). 12

4.1 Componentes de um sistema Jini conectados via uma rede, por onde o código é movido e as requisições são feitas. 15
4.2 Diagrama de Sequência para a busca do serviço 16
4.3 Diagrama de Sequência para a busca do cliente 17
4.4 Processo de registro do objeto do serviço. Adaptada de NEWMARCH (2001) 21
4.5 Objetos com o sistema de *lease*. Figura adaptada de NEWMARCH (2001) 23

5.1 Relação entre as camadas nas quais o sistema se divide 28
5.2 Modelo das entidades da base de dados do sistema 31
5.3 Modelo simplificado da relação entre Clientes e Serviços. 33
5.4 Interface gráfica de utilização do cliente da Comissão de Arbitragem 33
5.5 Interface gráfica de para os clientes Torcedor e Imprensa. Cadastro de opniões e listagens de dados da partida 34
5.6 Interface gráfica de Cadastro de Dados do Campeonato. 37
5.7 Interface gráfica de manipulação de dados das Equipes e Clubes. 38
5.8 Interface gráfica de manipulação das Disputas. 38
5.9 Interface gráfica de manipulação dos Estádios. 38
5.10 Interface gráfica de manipulação dos Jogadores. 39
5.11 Interface gráfica de central da Gerência do Sistema. 39
5.12 Diagrama de Sequência para o Serviço de Cadastro de Opinião e o Cliente Torcedor. 50
5.13 Interface gráfica do Cliente Torcedor. 51

Resumo

O cotidiano das pessoas está cercado de dispositivos computacionais, cada vez com mais recursos (sensores) e, com um processamento cada vez maior. A maneira como esses dispositivos se comunicam ainda não é natural e isto retarda o crescimento da Computação Ubíqua. Este trabalho apresenta uma maneira com a qual estes dispositivos possam se comunicar, utilizando a tecnologia Jini e os conceitos da Arquitetura Orientada a Serviços, aplicando estes conceitos em um caso de teste de Computação Ubíqua. Para a realização do estudo de caso foi construído um sistema fictício de gerência de um campeonato de futebol, no qual usuários poderão interagir entre si e, com o sistema de forma simplificada, ter acesso aos dados do campeonato em tempo real, durante o evento de uma partida. Essa comunicação é realizada por meio de serviços construídos com a tecnologia Jini, os quais foram baseados nos principais conceitos SOA, como a modularidade e a reusabilidade.

Palavras-chave: SOA, Jini, Computação Ubíqua.

Abstract

The people's daily lives are surrounded by computing devices, with increasing resources (sensors) and, with increasing processing. How these devices communicate is still not natural and This retards the growth of Ubiquitous Computing. This paper presents a way in which these devices can communicate using Jini technology and concepts of Service Oriented Architecture, applying these concepts in a test case of Ubiquitous Computing. To conduct the test case was constructed a fictitious system for management of a soccer championship, where users can interact with each other and with the system in a simplified way, have access to data in real time of the championship during the event. This communication is performed by services built using Jini technology, which were based on key SOA concepts, such as modularity and reusability.

Keywords: *Service-Oriented Archtecture, Ubicomp, Ubiquitous Computing, Jini .*

Capítulo 1

Introdução

1.1 Considerações Iniciais

O notável aumento das tecnologias que envolvem dispositivos móveis, particularmente nas características físicas, como o aumento de recursos e a maior capacidade de processamento, aliado ao avanço nas pesquisas em computação distribuída, naturalmente exige a necessidade da computação modular e federativa.

É importante perceber que o reuso dos recursos e a interligação entre eles colaboram para o aumento da produtividade e da facilidade em adequar as ferramentas e serviços para novos usos e aplicações exigidas pelo público.

Nesse contexto, é possível observar a aplicação da tecnologia para criação de softwares baseados em serviços, **SOA** (*Service-oriented Architecture*). Essa abordagem possibilita maior facilidade de inserção destes softwares (*serviços*) em outras áreas, contribuindo e muito para a **Computação Ubíqua**.

O termo Computação Ubíqua ainda não é tão difundido, assim como seu conceito. É comum encontrar diversos aplicativos em pequenos dispositivos realizando funções computacionais, isto acontece no cotidiano das pessoas, sem que elas percebam. Adequar os softwares para esse novo contexto é indispensável para a evolução da computação móvel e deve ser levado em conta desde o início de sua construção.

Os conceitos de SOA podem ser utilizados de maneiras distintas e em tecnologias diferentes, dependendo da finalidade e do contexto da aplicação. A tecnologia **Jini** foi escolhida para esse trabalho por ser baseada nos princípios em questão e por estender os recursos da solida tecnologia **Java**

1.2 Objetivo

O projeto visa estudar e analisar os conceitos envolvidos na arquitetura de desenvolvimento de softwares orientados a serviços (SOA) aplicado no contexto

da Computação Ubíqua.

Estes conceitos estão muito próximos e, portanto, será analisado se estes podem ser naturalmente complementares, a partir de um caso de teste no qual será criada uma solução utilizando destes conceitos com o apoio da tecnologia Jini, que é baseada na linguagem Java, a qual é muito popular atualmente e pode ser executada nos mais diversos dispositivos, móveis ou não.

O caso de teste, que é atual, reflete uma situação do mundo da Computação Ubíqua: informações atuais e instantâneas, de acesso fácil a todos os tipos de usuários envolvidos e conectados em uma rede, em um local onde essa informação pode ser muito interessante e naturalmente aceita.

1.3 Organização do Trabalho

O trabalho foi organizado em capítulos, sendo os Capítulos 2, 3 e 4 utilizados como embasamento teórico dos conceitos utilizados.

No Capítulo 5 é apresentado o caso de teste e a motivação do uso dos conceitos na situação escolhida. A Seção 5.2 explicita detalhes da implementação e o modo como as tecnologias foram utilizadas.

Por fim, no Capítulo 6 são discutidos os resultados e observações do trabalho, apontado as contribuições que foram feitas, as limitações enfrentadas e um olhar para futuros trabalhos que podem ser realizados a partir destes conceitos.

Capítulo 2

Computação Ubíqua

2.1 Apresentação

Este capítulo está baseado no trabalho de ARAUJO (2003), que é referência para vários outros trabalhos.

O termo "‘ubíquo"’ é pouco comum, contudo o uso desse conceito na computação é cada vez maior. A palavra pode ser definida como aquilo que está presente em todos os lugares, aquilo que é onipresente. Na computação isso reflete à dispositivos presentes no cotidiano das mais variadas formas e de maneira imperceptível para as pessoas.

O conceito básico da computação ubíqua é que a computação move-se para fora das estações de trabalho e computadores pessoais e torna-se pervasiva em nossa vida cotidiana. WEISER (1991), considerado o pai da computação ubíqua, vislumbrou ha uma década que, no futuro, computadores habitariam os mais triviais objetos: etiquetas de roupas, xícaras de café, interruptores de luz, canetas, entre outros, de forma invisível para o usuário. Neste mundo de Weiser, devemos aprender a conviver com computadores, e não apenas interagir com eles.

A invisibilidade ao usuário requer que os serviços estejam profundamente embarcados no meio e possuam uma alta colaboração. Ser invisível implica que a tecnologia seja usada de maneira tão natural que se dá de forma despercebida.

Atualmente existem dispositivos móveis com processamento de dados muito superior em relação aos super-computadores da década passada e o mais relevante, com mais recursos de conexão em rede e sensores para os mais variados usos, estas características possibilitam cada vez mais o emprego da computação ubíqua.

A computação móvel baseia-se no aumento da capacidade de mover fisicamente serviços computacionais conosco, ou seja, o computador torna-se um dispositivo sempre presente que expande a capacidade de um usuário utili-

zar os serviços que um computador oferece, independente de sua localização. Combinada com a capacidade de acesso a computação móvel tem transformado a computação em uma atividade que pode ser carregada para qualquer lugar.

O conceito de computação móvel é mais abrangente que o simples fato de poder mover um dispositivo que possa processar dados para diferentes lugares. Deve-se levar em conta a capacidade do dispositivo em interagir com o ambiente e de acordo com as necessidades e características do local onde se encontra reagir de maneira diferente.

A computação ubíqua surge na necessidade de se integrar mobilidade com a funcionalidade de qualquer dispositivo computacional, enquanto em movimento conosco, pode construir, dinamicamente, modelos computacionais dos ambientes nos quais nos movemos e configurar seus serviços dependendo da necessidade.

2.2 Características Principais

Existem pelo menos três princípios na computação ubíqua:

- **Diversidade:** É a capacidade de cada dispositivo oferecer uma solução mais adequada a situação, ou seja, telefones são especialistas em chamadas de voz, computadores talvez sejam os melhores para se acessar a internet e máquinas fotográficas são as que possuem capacidade de obter as melhores fotos. Dispositivos ubíquos acenam com uma nova visão da funcionalidade do computador, que é o propósito especifico, que atende necessidades específicas de usuários particulares. ARAUJO (2003) diz ainda que apesar de vários dispositivos poderem oferecer funcionalidade que se sobrepõem, um pode ser mais apropriado para uma função do que outro;

- **Descentralização:** Na computação ubíqua as responsabilidades são distribuídas entre vários dispositivos pequenos que assumem e executam certas tarefas e funções. Estes dispositivos cooperam entre si para a construção de inteligência no ambiente, que é refletida nas aplicações. Para isso uma rede dinâmica de relações é formada, entre os dispositivos e entre dispositivos e servidores do ambiente, caracterizando um sistema distribuído.;

- **Conectividade:** Na computação ubíqua, tem-se a visão da conectividade sem fronteiras, em que dispositivos e suas aplicações movem-se juntamente com o usuário, de forma transparente, entre diversas redes heterogêneas, tais como as redes sem fio de longa distância e redes de média

e curta distância. Para que se atinja a conectividade e interoperabilidade desejada é preciso basear as aplicações em padrões comuns, levando ao desafio da especificação de padrões abertos.

2.3 Desafios da Computação Ubíqua

Há vários desafios a serem superados na computação ubíqua nos níveis tecnológico, social e organizacional. No nível social, pesquisadores afirmam que a computação ubíqua trará problemas de segurança e privacidade e mudará a forma como os trabalhadores e empresas interagem entre si, o que gera novas preocupações, por exemplo, como por exemplo, o empregador supervisiona os empregados, já que computadores estarão por toda parte conectados uns aos outros por redes. Igualmente desafiador é entender como os avanços tecnológicos podem ajudar ou prejudicar o ser humano e o bem-estar social (ARAUJO, 2003).

A Computação Ubíqua evolui naturalmente com o avanço dos dispositivos computacionais, contudo mesmo sendo cada vez mais comum o uso das tecnologias da computação no cotidiano das pessoas e, mesmo assim, ainda existem muitos desafios, alguns exemplos são:

- Tratamento de Contexto: coletar dados de diversos sensores, processar esses dados e dissemina-los para os mais diversos dispositivos;
- Melhor utilização dos recursos de dispositivos pessoais;
- Realização da tarefa apropriada no dispositivo apropriado;
- Criação de metodologias de desenvolvimento de aplicações em que a aplicação se move juntamente com o usuário.

2.4 Aplicações

O número de aplicações que envolvem computação ubíqua é cada vez maior, devido a diversos fatores. O avanço tecnológico dos dispositivos móveis, por exemplo, é um fator que está facilitando este crescimento.

A computação ubíqua pode ser encontrada nas mais diversas áreas, são exemplos: ensino, trabalho colaborativo, residências e automóveis inteligentes, dentre outras.

Algumas das aplicações descritas por ARAUJO (2003) estão exemplificadas a seguir:

2.4.1 Ensino

O objetivo é dar suporte aos professores em suas atividades de produção de material didático e aos alunos na anotação de aulas de forma personalizada. São aplicativos que capturam as aulas presenciais e disponibilizam este material sob a forma de hiperdocumentos multimídia, que posteriormente, são reaproveitados e mesclados com outras tecnologias como por exemplo técnicas de visão computacional e combinação de mais de uma fonte de vídeo para a produção dos hiperdocumentos multimídia.

2.4.2 Trabalho Colaborativo

Oferece suporte a reuniões formais e/ou informais entre participantes de um mesmo grupo. Aplicativos que capturam atividades de uma reunião informal, registrando os desenhos feitos em uma lousa eletrônica compartilhada e a voz de cada membro da reunião, e disponibiliza essa informação sob a forma de hiperdocumentos multimídia sincronizados são exemplos de aplicações nessa área.

2.4.3 Aplicativos que auxiliam a movimentação do usuário

Guias turísticos eletrônicos trabalham normalmente com informações de localização dos usuários e utilização de dispositivos portáteis para conferir maior mobilidade aos usuários. Os principais objetivos dessa classe de aplicações são registrar os locais visitados e identificar a posição atual do usuário dentro de um espaço de interação, como campus universitário, museu, etc.

2.4.4 Residência Inteligente

Basicamente, as aplicações de computação ubíqua que abrangem o domínio doméstico têm por objetivo conhecer as atividades dos moradores de uma casa e fornecer serviços que aumentem a qualidade de vida deles. Os aplicativos para essa área focam no desenvolvimento de arquiteturas e tecnologias para ambientes inteligentes. Um exemplo de aplicativo controlaria o ambiente que contém um computador, telas eletrônicas (incluindo uma tela grande), caixas de som, sofás e mesa de café, entre outros itens. Os serviços são fornecidos para melhorar o ambiente, como por exemplo, automatizar o controle de luz, tocar música baseado na localização (dependendo da preferência do usuário), e ainda transferir automaticamente o conteúdo de uma tela para outra.

Capítulo 3

Arquitetura Orientada a Serviços

3.1 Introdução

Este capítulo é baseado no estudo de JOSUTTIS (2007), que é bastante completo e serve de referência para vários outros autores.

O ambiente de negócios é muito dinâmico e necessita de constantes análises para ser corretamente entendido, isso acarreta constantes mudanças nas tecnologias existentes, criando um ambiente caótico para a produção e manutenção de softwares. É necessário um planejamento que compreenda as mudanças repentinas do ambiente e esteja preparado para adequar suas soluções a essas mudanças, sem comprometer a integridade e qualidade de seus produtos.

Os anos de evolução tecnológica não acompanham a velocidade com a qual o mercado muda suas características. Um arquiteto de sistemas hoje tem como objetivo estender a vida de uma solução já existente dentro da empresa através da implementação de uma nova lógica ou processo do negócio que manipule um repositório de dados existente. Isso é custoso e compromete a integridade e qualidade do sistema. Faz-se necessário uma diferente abordagem.

Arquitetura Orientada a Serviços ou **SOA** é um paradigma que reúne vários conceitos e práticas para um objetivo maior: desenvolver sistemas modularizados. SOA é um paradigma para a compreensão e manutenção de processos de negócio que abrangem sistemas grandes.

Para sobreviver, os negócios de TI necessitam de uma arquitetura na qual possam basear seus produtos. Essa necessidade naturalmente leva a escolha da arquitetura SOA, segundo OASIS (2006):

A SOA é um paradigma para organização e utilização de competências distribuídas que estão sob controle de diferentes domínios proprietários. Em geral, as entidades (pessoas e organizações) criam competências para resolver ou suportar uma solução para problemas que encontram no decorrer de seus

negócios. É natural pensar que as necessidades das pessoas podem ser compatíveis com as competências oferecidas por alguém ou, em outras palavras no mundo da computação distribuída, um requisito de um agente computador pode ser compatível com o de outro pertencente a outro proprietário.

SOA é uma abordagem que ajuda sistemas a se manterem escaláveis e flexíveis enquanto crescem. Essa característica é muito procurada pelo mercado e se torna indispensável para muitos sistemas adequarem suas soluções aos negócios dinâmicos das empresas.

A única maneira de se manter um grande sistema distribuído flexível é fazê-lo suportar heterogeneidade, descentralização e tolerância a falhas, características estas, supridas pela tecnologia SOA.

3.2 Conceitos Básicos

Existem três conceitos básicos e técnicos, existentes em qualquer abordagem SOA, são eles:

- Serviços;
- Interoperabilidade;
- Baixo Acoplamento.

3.2.1 Serviços

Essencialmente um serviço é uma representação de uma funcionalidade do sistema. O objetivo do SOA é estruturar um grande sistema distribuído, baseado nas abstrações das regras de negocio e funcionalidades.

A lógica de uma regra de negócio deve estar definida e disponibilizada como um serviço que pode ser reutilizado por outros sistemas.

3.2.2 Interoperabilidade

A implementação dos serviços deve ser realizada de forma independente da plataforma de desenvolvimento,tecnologias de implementação e linguagens de programação.

A arquitetura deve ser hábil a conectar sistemas de maneira simples. Interoperabilidade é a base na qual inicia-se a implementação das funcionalidades (serviços) que são disseminadas para vários sistemas distribuídos.

3.2.3 Baixo Acoplamento

O conceito visa minimizar as dependências entre os serviços, permitindo assim, flexibilidade na mudança das regras do negócio, se adaptando mais fácil e rapidamente na nova abordagem. Os serviços deverão ser projetados para interagir. Vários sistemas estão envolvidos, não sendo portanto, momento para robustez. Cita ainda que é necessária atenção à essas três características: flexibilidade, escalabilidade e tolerância a falhas

3.3 Ingredientes

Não é possível comprar SOA. É necessário os conceitos dessa arquitetura sejam introduzidos no ambiente de forma adequada, ou seja, descobrindo o nível certo de descentralização e que os outros conceitos estejam naturalmente integrados ao projeto.

JOSUTTIS (2007) distribui alguns itens, chamados de ingredientes por ele, que complementam a adequação do problema real nos projetos SOA.

3.3.1 Infraestrutura

É a parte técnica do SOA que permite a alta operabilidade. Tem a responsabilidade de transformar dados, roteamento inteligente, segurança, confiabilidade, gerenciamento de serviços, monitoramento e *logging*.

3.3.2 Arquitetura

Necessária para restringir todas as possibilidades do SOA de modo que o sistema se mantenha estável e passível de manutenção, adequando as necessidades do sistema e colaborando para tomada de decisões de modo que não comprometa a integridade.

3.3.3 Processos

Técnicas para que mais de um responsável possa gerir o sistema, visto que rotineiramente não é uma única pessoa responsável por tudo e sim várias, cada qual no seu setor.

3.3.4 Governança

Modo pelo qual todos os processos são administrados e como as estratégias SOA adotadas interagem. É necessário configurar corretamente os processos de modo que estabilize o conceito SOA na organização. Normalmente existe

um time central de competência em SOA em uma organização e é ele o responsável por lidar com o entendimento entre a infraestrutura, arquitetura e processo.

3.4 Serviços

Parte fundamental dentro do SOA, encapsulam uma função de negócio que pode ser reutilizável, como evidenciado na Figura 3.1. Podem ser entendidos como uma função do sistema computacional construída de forma a ser facilmente vinculada a outros componentes de software, que podem ser outros serviços.

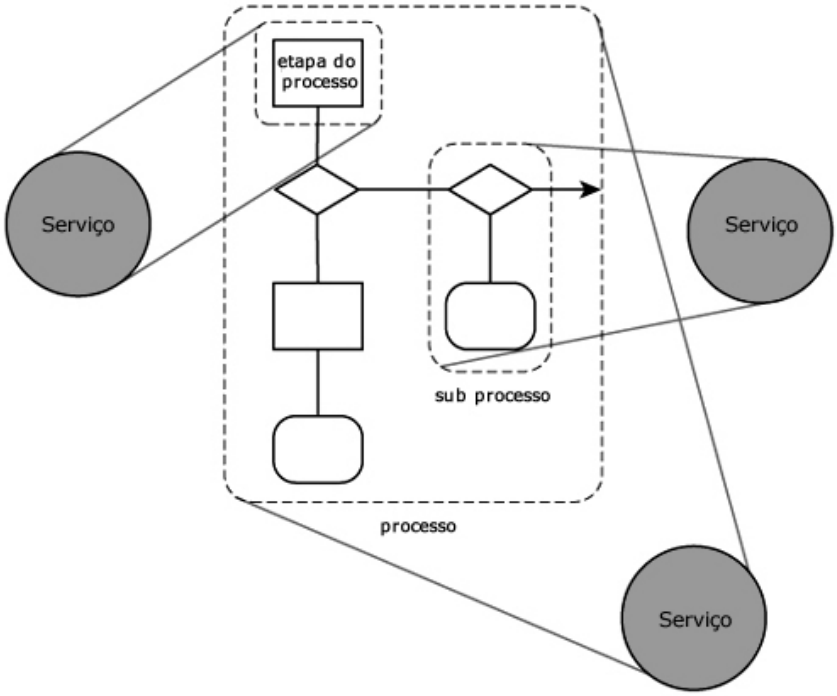

Figura 3.1: Serviços encapsulando vários conceitos lógicos. Adaptado de ERL (2005).

Serviço é um mecanismo para habilitar o acesso a um conjunto de competências, onde o acesso é provido usando uma interface já descrita e é consistentemente exercitada com restrições e políticas como especificados pela descrição de serviço (OASIS, 2006).

JOSUTTIS (2007) diz que, na essência, um serviço é uma representação da TI de alguma funcionalidade do negócio, ou seja, serviço é a representação de uma abstração.

Os serviços devem possuir algumas características complementares as descritas nos conceitos do SOA, e que segundo ERL (2005) são: reusabilidade, Abstração de lógica de negócio, composição, autonomia e a capacidade de ser

descoberto.

3.4.1 Reusabilidade

Utilização de padrões de projeto para aumentar as chances de utilizar determinada funcionalidade em situações distintas, muitas vezes não consideradas no momento da criação. Principal objetivo é reduzir esforço no desenvolvimento, reduzindo o custo e aumentando a produtividade.

3.4.2 Abstração de lógica de negócio

Escondem detalhes da implementação das funcionalidades do mundo exterior, fazendo com que a lógica se mantenha segura mesmo quando utilizada por outros meios. Normalmente a lógica do negócio é irrelevante para o consumidor do serviço, facilitando ainda mais se não necessitar conhecimento prévio para o seu uso.

3.4.3 Composição

Serviços tem a autonomia e poder de compor outros serviços, permitindo à logica ser representada em diferentes níveis de granularidade, além de promover a reusabilidade e a criação de camadas de abstração.

3.4.4 Autonomia

A lógica governada por um serviço é circundada por limites explícitos. O serviço possui controle dentro destes limites e não depende de outros serviços para executar sua governança.

Esse não limita a aplicação de um serviço a outro ou até a outra aplicação, fazendo com que ele não seja limitado na implementação e uso.

3.4.5 Capacidade de ser descoberto

Serviços devem permitir às suas descrições serem descobertas e entendidas por humanos e requisitantes de serviços que possam utilizar sua lógica.

Esta característica é importante para que seja utilizado o serviço mais adequado para determinada situação e, até mesmo, para que não existam serviços redundantes por falta de conhecimento sobre os já existentes.

3.4.6 Classificação

Os serviços podem ser classificados, tecnicamente, em três categorias: serviços básicos, serviços compostos e serviços processos.

Esta classificação da visibilidade para três camadas de serviços e seus estágios de expansão, conforme ilustrado na Figura 3.2:

- SOA fundamental: possui somente uma camada de serviço, com serviços básicos;
- SOA federado: além da camada de serviços básicos possui uma camada com serviços compostos;
- SOA habilitado a processos: possui uma terceira camada com serviços processo.

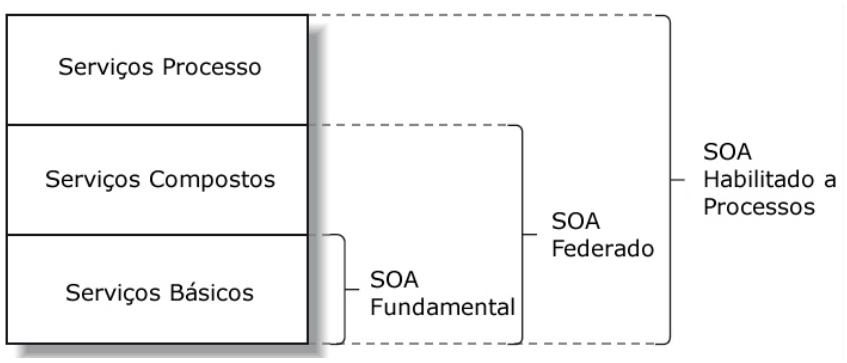

Figura 3.2: Camadas fundamentais dos serviços. Adaptado de JOSUTTIS (2007).

3.4.7 Serviços Básicos

São aqueles que apenas provem funcionalidades básicas, que não façam sentido se forem divididos em múltiplos serviços. Se referem a primeira camada de expansão.

3.4.8 Serviços Compostos

Segundo estágio de expansão, os quais representam serviços compostos por outros serviços (básicos e/ou serviços compostos) processo também chamado de orquestração.

3.4.9 Serviços Processo

O terceiro passo da expansão representa os *workflows* ou processos de negócio de longo prazo. Diferente dos serviços básicos e compostos, um serviço de processo tem um estado que se mantém estável durante múltiplas chamadas.

3.5 Segurança

Fator importante a ser considerado em uma implantação SOA. JOSUTTIS (2007) define a seguinte categorização para os aspectos de segurança para SOA:

- **Autenticação:** verificar identidades de usuários, dispositivos ou serviços externos;
- **Autorização:** controlar restrições para uma determinada identidade;
- **Confidencialidade:** assegurar que ninguém, além do fornecedor e consumidos tenha acesso aos dados durante a transferência;
- **Integridade:** garantir que os dados não sejam manipulados ou falsificados, que contenham erros ou lixos eletrônicos;
- **Disponibilidade:** manter o sistema sempre disponível;
- **Contabilização:** monitorar chamadas de serviço para gerenciamento, planejamento ou outros propósitos;
- **Auditoria:** monitorar e rastrear todo o fluxo de dados relevantes para a segurança, seja para melhorar sua confiabilidade, seja para atender a determinações legais.

Capítulo 4

Jini

4.1 Introdução

Jini é uma tecnologia criada para facilitar a implementação de sistemas distribuídos. Responsável pelo tratamento de uma complexa e burocrática rede, facilitando o ingresso de um dispositivo, seja ele um hardware ou um software.

NEWMARCH (2001) define Jini como sendo o nome de um ambiente de computação distribuída que oferece uma rede "plug and play", ou seja, um dispositivo pode ser conectado a uma rede e sua presença é anunciada para que clientes possam localizar esse serviço e utilizar suas funcionalidades.

A tecnologia Jini oferece um serviço que centraliza as principais funcionalidades de uma rede dinâmica, tratando eventos como a inserção e remoção de dispositivos em uma rede, tornando a aplicação e implementação muito mais simples. Ela forma uma rede que se adapta as mudanças no ambiente e proporciona alta integração entre os dispositivos conectados.

Jini foi construído no topo da consistente tecnologia Java, utilizando-se assim de poderosos recursos já existentes como serialização de objetos, soquetes e *RMI (Remote Method Invocation)*. Esse fator facilita a criação e desenvolvimento de sistemas distribuídos organizados em uma federação de serviços. Um serviço pode se conectar a tudo que está presente em uma rede e está pronto para exercer uma função útil.

Para OAKS (2000) Jini pode ser visto como um conjunto de classes Java e serviços que possuem potencial para criar a sua própria solução, pois a tecnologia pode ser explorada de maneiras diferentes.

Outra característica importante do Jini é que ele fornece a infraestrutura necessária para que, em tempo de execução, clientes e consumidores de serviços possam se comunicar e desempenhar suas funções.

Jini permite que dispositivos co-relacionados, serviços e aplicações (e há pouca distinção entre estes) possam acessar um ao outro sem maiores problemas para se adaptar a um ambiente em constante mudança, e possam com-

partilhar códigos e configurações de forma transparente. Jini tem potencial para alterar radicalmente a maneira como a utilização das redes de serviços de informática atualmente trabalha, pois permite e incentiva novos tipos de serviços e novos usos das redes existentes (OAKS, 2000).

4.1.1 Componentes

Jini está estruturado em três principais componentes, evidenciados na Figura 4.1:

- Serviços;
- Clientes;
- *Lookup Service*.

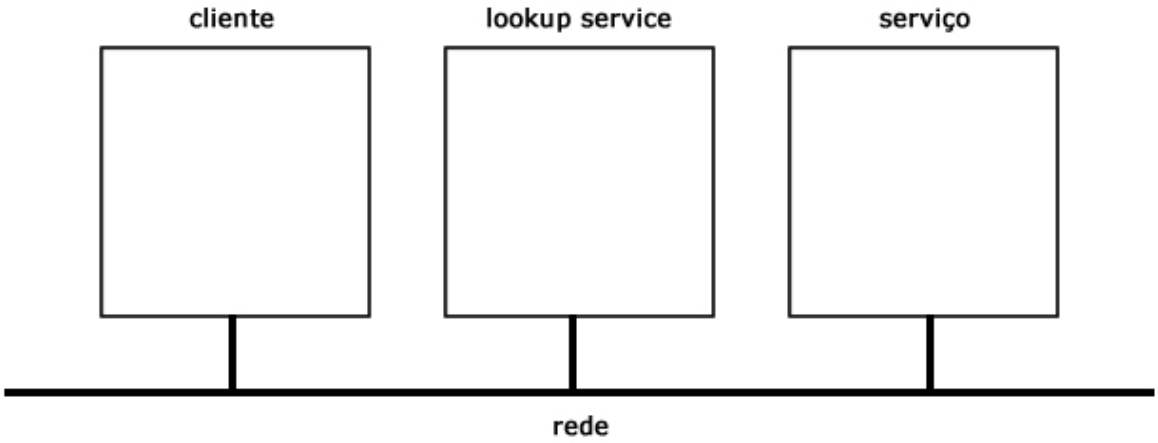

Figura 4.1: Componentes de um sistema Jini conectados via uma rede, por onde o código é movido e as requisições são feitas.

O código é movido em torno dessas três entidades, o que requer que os objetos sejam projetados adequadamente, aptos a serialização.

NEWMARCH (2001) exemplifica os componentes de forma que o serviço possa ser algo como uma impressora, um tostador de pão ou uma agencia de matrimônio. O cliente é quem consome esses serviços, ou seja, faz uso deles e o *lookup service* atua como o negociador, o localizador, o corretor entre o serviço e o cliente. Ele ainda evidencia que existe um componente adicional: a rede, conectando esses componentes.

4.1.2 Registro de Serviço

Os serviços mostram-se como programas em forma de objetos na linguagem Java, possivelmente constituídos por outros objetos. Eles são a base da intera-

tividade para os sistemas Jini. O serviço é um conceito lógico, como um liquidificador, um serviço de chat ou um disco rígido. Ele é ser definido como um interface Java e essa característica é comumente utilizada pra identifica-lo.

Um serviço é criado por um *service provider*, que segue algumas regras:

- Cria os objetos que executam o serviço;
- Registra um objeto, *service object*, utilizando outro serviço, o *Lookup Service*. O *service object* é a parte que fica publicamente visível e que é adquirida pelo cliente;
- Mantém o serviço ativo no servidor.

Para que o *service provider* possa registrar o objeto no *lookup service* é necessário que ele encontre o *lookup service* na rede. Esse processo pode ser feito de duas maneiras:

- Descoberta *Unicast*: detalhes na Seção 4.3.1.
- Descoberta *Multicast*: detalhes na Seção 4.3.2.

Quando um *lookup service* inicia um pedido para registro do serviço, ele envia o objeto *registrar* ao *service provider*. Esse objeto age como um intermediário entre o serviço e o *lookup service*.

NEWMARCH (2001) narra este processo: quando o *lookup service* recebe uma notificação, ele envia para o servidor um objeto registrar. Esse objeto fica alojado na JVM do serviço. Qualquer pedido que o *service provider* necessite ao *lookup service* será realizado utilizando esse objeto. O *service provider* registra o serviço com o *lookup service* e para isso ele faz uma cópia do objeto do serviço e armazena no lookup servico, como visto na Figura 4.2, que descreve o diagrama de sequência para esse processo.

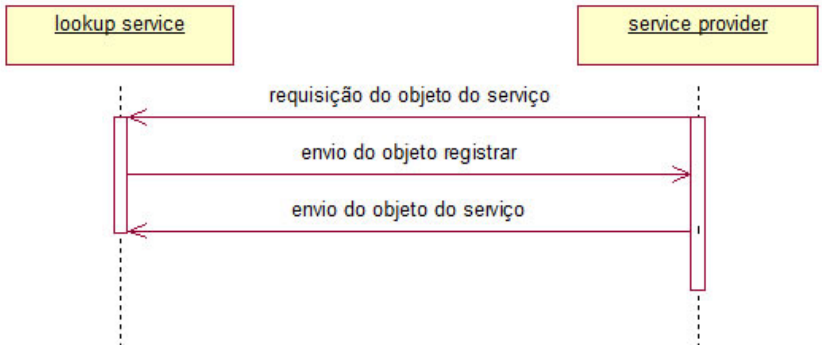

Figura 4.2: Diagrama de Sequência para a busca do serviço

Outros detalhes desse processo serão discutidos na Seção 4.4

4.1.3 Busca do Cliente

O cliente necessita de uma cópia do serviço em sua JVM e esse mecanismo ocorre de forma análoga ao mecanismo de registro do serviço, como visto na Figura 4.3, que descreve o diagrama de sequência para esse processo.

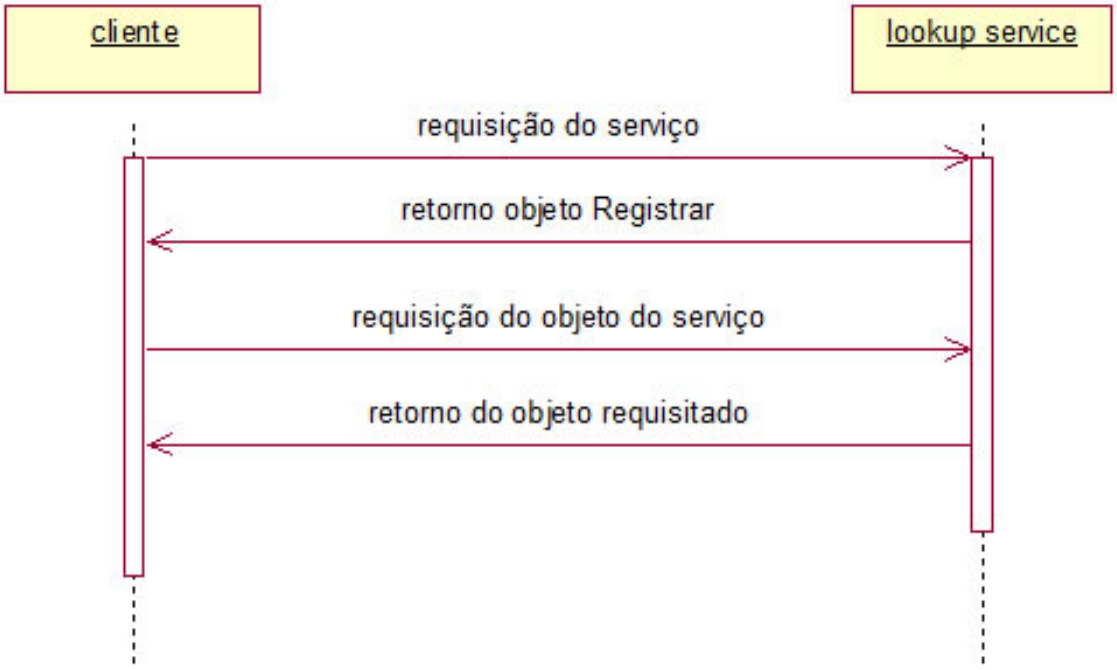

Figura 4.3: Diagrama de Sequência para a busca do cliente

Na ultima etapa, o objeto original está rodando no servidor, com uma cópia do objeto do serviço armazenada no *lookup service* e outra cópia do objeto do serviço rodando na JVM do cliente. Já nesse contexto, o cliente faz requisições ao objeto do serviço que está rodando em sua própria JVM.

Esse tópico será melhor abordado no Capítulo 4.5.

4.2 Proxies

NEWMARCH (2001) diz que alguns serviços podem ser executados por um único objeto, o *service object*. Ele é indispensável por não existir outro modo de controlar o serviço de maneira completa sem ele. Nesse cenário, a implementação do serviço deve ser composta de pelo menos dois objetos: um objeto que esteja rodando no cliente e outro, distinto, que esteja rodando no *service provider*.

Esse *service object* é, na realidade, um *proxy* , que se comunicar com os outros objetos no *service provider*. NEWMARCH (2001) descreve o proxy como sendo a parte do serviço que é visível para os clientes, mas a sua principal

função será passar as chamadas dos métodos para o restante dos objetos, de forma que o serviço seja totalmente implementado.

Implementar e entender o funcionamento do *proxy* é muito importante, principalmente quando o objeto do serviço necessita controlar remotamente uma funcionalidade do serviço que não pode ser acessado diretamente, ou seja, que seja acessível ao *service provider* mas não aos objetos que estão rodando no cliente.

4.3 Lookup Service

O *lookup service* é essencial para a estrutura Jini. Um serviço é registrado em um *lookup service* e os clientes o utilizam para buscar os serviços que estão interessados.

Clientes localizam um serviço com uma requisição ao *lookup service*. Para realizar isso, os clientes necessitam primeiramente encontrar o *lookup service*. Da mesma maneira, um serviço para se tornar disponível necessita localizar-lo para depois poder se registrar.

NEWMARCH (2001) cita que a Sun oferece um *lookup service* chamado *reggie* como parte do padrão Jini de distribuição. Como a especificação de um *lookup service* é aberta, se espera que surjam outras implementações no futuro.

Em uma rede podem existir vários *lookup services* e é possivel que qualquer cliente ou serviço possa acessá-lo, dependendo do nível das permissões de acesso. Por esse motivo, existem duas principais maneiras de se descobrir um *lookup service* em uma rede:

- Descoberta *Unicast*
- Descoberta *Multicast*

4.3.1 Descoberta Unicast

Utilizada quando, previamente, é conhecida a localização do *lookup service* na rede, ou seja, em qual máquina ele reside. Dessa maneira o acesso é feito diretamente. NEWMARCH (2001) define como sendo mais comum o uso quando o *lookup service* reside fora de sua rede local.

A conexão com o *lookup service* é realizada pelo método *getRegister* da classe *LookupLocator*. A classe possúi dois principais construtores:

```
package net.jini.core.discovery;
public class LookupLocator
LookupLocator(java.lang.String url) throws
java.net.MalformedURLException;
LookupLocator(java.lang.String host,int port);
```

Para o primeiro construtor é necessário que se informe a URL, que pode ser *jini://host/* ou *jini://host:port/*.

NEWMARCH (2001) destaca dois métodos do *LookupLocator*:

- *String getHost();*
- *int getPort();*

Esses métodos retornam informações sobre o servidor, que são muito úteis para certificar que tudo está ocorrendo como é esperado.

4.3.2 Descoberta Multicast

Se a localização do looup service é desconhecida na rede, é necessário realizar uma busca para que possamos encontrá-lo. Essa busca é conhecida como **Descoberta Multicast**.

OAKS (2000) define a descoberta *multicast* como sendo uma requisição que o cliente envia em um formato específico. Todos os *lookup services* que estiverem na rede irão receber e responder a esse pedido, informando ao cliente que o *lookup service* foi descoberto.

Pode existir um grande número de serviços de *lookup service* funcionando na mesma rede acessível a busca *multicast*. Desta maneira é necessário um modo de filtrar essas requisições, uma maneira é utilizar **Grupos**.

4.3.3 Grupos

É uma maneira de restringir o acesso aos serviços disponíveis em uma rede. Apenas serviços de um mesmo grupo é que irão se comunicar. Essa característica não restringe a distância nem a quantidade de requisições entre os serviços, apenas melhora o agrupamento entre eles.

NEWMARCH (2001) cita os grupos como sendo departamentos de uma empresa, como o departamento de engenharia ou o departamento de recursos humanos ou até mesmo o departamento de publicidade. Alguns serviços de lookup podem conter mais de um grupo ou ser apenas para um único grupo.

Quando um *lookup service* é iniciado, pode-se informar uma lista de grupos aos quais ele pertence, podemos fazer esse procedimento via linha de comando ou no próprio código, como a seguir:

```
String [] groups = "Departamento de Engenharia;
```

Jini possui duas principais classes que são necessárias no processo de descoberta *multicast*, que NEWMARCH (2001) define como sendo:

- **LookupDiscovery:** Utilizada para a descoberta em *multicast*. Ela possúi um único construtor: lookupDiscovery. Esse construtor é o responsável pela busca e se caso ele receba o parâmetro nulo significa que ele descobre todos os serviços ativos na rede. Se receber uma lista vazia de valores significa que o objeto é criado mas não realiza nenhuma busca, apenas ocorrendo quando o método setGroups() é chamado. A última situação é caso uma lista não vazia de strings é recebida, isso fará com que ele trabalhe nos grupos que forem identificados nessa lista.

- **DiscoveryListener:** Uma transmissão em modo *multicast* é uma chamada através da rede, esperando por respostas de serviços de lookup. Assim que essas respostas correm um listener é ativado. Esse listener é registrado da seguinte maneira:

```
public void addDiscoveryListener(DiscoveryListener l)
```

Esse listener deve implementar a interface DiscoveryListener:

```
package net.jini.discovery;
public abstract interface DiscoveryListener
public void discovered(DiscoveryEvent e);
public void discarded(DiscoveryEvent e);
```

O método *discovered()* é invocado quando um serviço é descoberto e o método *discarted()* é chamado assim que uma aplicação descarta o serviço, através do método *discart()* no objeto registrar.

4.4 Registrando um serviço

Para que um serviço seja visto por um cliente, ou seja, para que ele possa ser utilizado, é necessário que ele seja registrado no *lookup service* de maneira correta. Se for analizado pelo ponto de vista do *lookup service*, o serviço primeiramente obtem um objeto da classe ServiceRegistrar. Com esse objeto o serviço pode invocar o método register(), que efetivamente registra-o na rede.

A maneira como ele é anunciado diferencia na classe que ele recebe do servidor, se for um unicast lookup ele trabalha com o objeto LookupLocator. Se for utilizado o sistema multicat lookup, será utilizada o objeto ServiceRegistrar. Independente da situação, ele receberá um objeto ServiceRegistrar, o qual trabalhará como um proxy para o *lookup service*. Nesse momento ele estará apto a utilizar o método registrar, ilustrado na Figura 4.4.

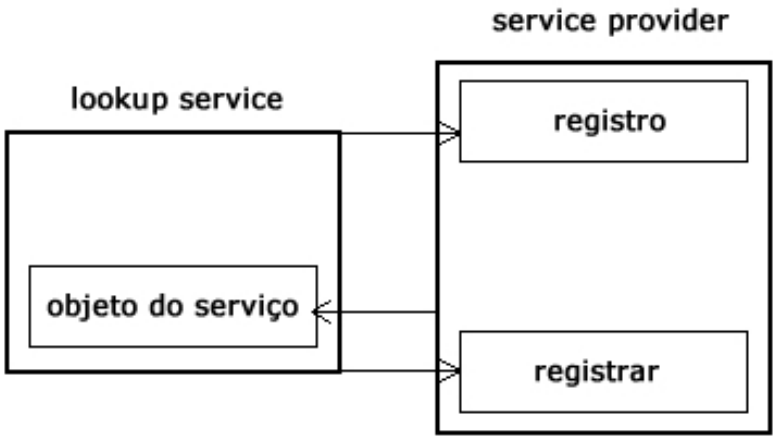

Figura 4.4: Processo de registro do objeto do serviço. Adaptada de NEWMARCH (2001)

O *lookup service* necessita de uma classe para agir como proxy em suas comunicações, essa classe é implementada a partir de uma classe abstrata chamada **ServiceRegistrar**.

Esse proxy, segundo NEWMARCH (2001), roda na aplicação, que pode ser tando um cliente como um serviço. Essa classe é enviada para a aplicação que procura pelo *lookup service*. Quando encontra a aplicação, é executada como um objeto no espaço de endereço da aplicação e a aplicação realiza suas chamadas de métodos através dela. Quando existe necessidade é essa classe que comunica novamente com seu *lookup service*. A implementação dessa classe é feita normalmente pelo *reggie* da Sun. O *reggie* utiliza RMI para comunicação, o que não é importante para a aplicação, pois ela apenas utiliza sua funcionalidade, que pode ser implementada de forma diferente, se necessário.

NEWMARCH (2001) evidencia que o objeto do proxy não deve armazenar

nenhuma informação no lado da aplicação mas deve manter ativa a informação do *lookup service*, que pode ser utilizado caso haja necessidade. Essa funcionalidade, segundo NEWMARCH (2001) é obtida na execução do *lookup service* fornecido pela Sun.

O objeto ServiceRegistrar possui dois principais métodos, um deles é utilizado por um serviço que tenta registrar (register()) e o outro por um cliente que necessita encontrar um serviço (lookup()).

Quando um service locator é descoberto ele envia um objeto ServiceRegistrar para ser rodado no cliente ou no serviço. Esse objeto pode ser implementado utilizado por dois grandes consumidores: serviços que são registrados e clientes que estão buscando por serviços.

4.5 Busca do Cliente

Depois de encontrar um *lookup service*, um cliente necessita encontrar um serviço. Do serviço ele obtém o objeto ServiceRegistrar. Para encontrar um serviço no *lookup service*, o cliente necessita descrever o serviço, utilizando o objeto ServiceTemplate. O cliente requisitaa ao objeto que recebeu do *lookup service* um dos dois métodos que pode para encontrar um ou mais de um serviços, são eles:

- *java.lang.Object lookup(ServiceTemplate templ)* : retorna um serviço relacionado ao template
- *ServiceMatches lookup(ServiceTemplate templ, int maxMatches)* : retorna um grupo de serviços relacionados ao template.

Para especificar o serviço, o *lookup service* utiliza a clase ServiceTemplate:

```
package net.jini.core.lookup;
public Class ServiceTemplate
public ServiceID serviceID;
public java.lang.Class[] serviceTypes;
public Entry[] attributeSetTemplates;
ServiceTemplate(ServiceID serviceID,
java.lang.Class[] serviceTypes,
Entry[] attrSetTemplates);
```

A classe utilizada para definir qual o serviço que o cliente necessita, deve ser, obviamente, conhecida pelo cliente a priori. O cliente pode requisitar a

cada *lookup service* uma lista com os serviços que ele possui que são relacionados a essa classe previamente conhecida.

O objeto que essa classe retorna caso a busca seja bem sucedida, pode ser utilizado como a classe ao qual ele foi descoberto. Os métodos desse objeto podem ser então, invocados.

4.6 Leasing

Segundo NEWMARCH (2001), nas aplicações distribuídas, podemos nos deparar com inúmeras falhas na rede ou nos componentes que estão presentes nela. *Leasing* é o modo com o qual Jini consegue uma tolerância maior a falhas, ou seja, é uma maneira pela qual pode-se dizer que um componente está ativo. Esse método garante acesso aos recursos da rede por um período conhecido de tempo, estipulado no momento em que o acordo entre o componente e a rede são conectados.

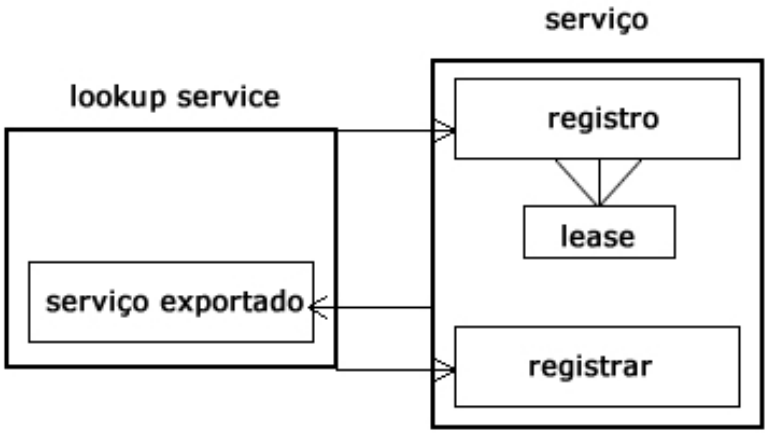

Figura 4.5: Objetos com o sistema de *lease*. Figura adaptada de NEWMARCH (2001)

Leases são garantidos aos serviços por intermédio dos lookups services. Os lookups services, segundo NEWMARCH (2001), não mantem um lease por tempo indeterminado, isso seria arriscado pois, por inúmeras causas, um serviço pode desaparecer. Outro ponto negativo é manter uma lista de serviços que não estão mais ativos e o *lookup service* ainda não saiba disso. É um desperdício de recursos, tanto do *lookup service* quanto dos cliente que tentarão acessar esses serviços inativos. Para que essa situação seja evitada, o *lookup service* garante um conhecido período de tempo para que o serviço seja dado como ativo. O modo como esse sistema de lease funciona está demonstrado na Figura 4.5.

4.7 Métodos Remotos

Jini usa uma técnica especial para lidar com invocação remota, que é chamada Jeri (Jini Extensible Invocation). NEWMARCH (2001) define como sendo uma alternativa para o Java Remote Method Protocol (JRMP) que é utilizado pelo tradicional RMI (Remote Method Invocation). NEWMARCH (2001) diz que esse novo método incorpora lições aprendidas do RMI sobre IIOP e outros transportes.

OAKS (2000) define RMI como sendo um mecanismo que permite que objetos que residam em diferentes JVM possam invocar métodos uns dos outros. OAKS (2000) diz ainda que RMI é um sistema de objetos distribuídos e nele o servidor exporta certas referências dos objetos. Esses objetos exportados são conhecidos como *remote objects*. O cliente pode obter uma referência a um objeto remoto e, uma vez com ela, pode invocar métodos nesse objeto da mesma maneira que invocaria métodos em outros objetos não remotos.

OAKS (2000) diz que a diferença entre métodos acessados remotamente e acessados localmente é que acessos remotos necessitam que a invocação ocorra no servidor com a máquina virtual do servidor. Os parâmetros são empacotados pelo RMI e enviados do cliente para o servidor, onde o método será executado. O valor retornado do método é empacotado novamente pelo RMI e enviado de volta para o cliente.

Essas chamadas remotas são síncronas, ou seja, a execução que fez a requisição no cliente é bloqueada até que o retorno da chamada seja recebido. Isso segundo OAKS (2000) faz com que o acesso aos métodos remotos pareçam ainda mais com o método padrão de invocar métodos.

Capítulo 5

Caso de Teste

Para correlacionar todos os conceitos abordados nesse trabalho, foi necessário, primeiramente, encontrar uma situação na qual a Computação Ubíqua fosse aplicável. Apesar de existirem diversas, foi priorizada uma situação que colaborasse com a evolução de alguma área ainda pouco explorada por esse ramo de estudo e que apoiasse as atividades das comunidades brasileiras.

O Brasil irá sediar dois importantes eventos esportivos nos próximos anos, um deles, em 2014, é a Copa do Mundo FIFA de futebol, evento esse que segundo Wikipedia (2011a), não ocorre no país há 36 anos. Reunirá equipes do mundo todo e será realizado nas cidades mais importantes do Brasil, reunindo milhões de pessoas.

O outro evento é oficialmente denominado Jogos da XXXI Olimpíada que, segundo Wikipedia (2011b), é um evento multi esportivo e será realizado no segundo semestre de 2016, no Rio de Janeiro. Esse evento reúne diversos atletas das mais variadas modalidades esportivas e atrai grande público para acompanhar todos os momentos dos jogos.

Neste cenário é observado que existe uma grande procura por informações sobre os jogos bem como maior interatividade com os usuários que, nesse caso, podem ser atletas, comissões técnicas e organizadoras ou até mesmo pessoas comuns interessadas nos eventos.

Ambos os eventos funcionam como uma vitrine internacional, apresentando as qualidades do país, sendo que quanto maior o valor agregado aos eventos, melhor será a imagem passada para todos que acompanharão os jogos. Portanto, certamente, qualquer atividade que colabore com a organização e facilite a maneira como as informações são expostas, agrega um valor elevado ao evento.

Com base nestes argumentos, o foco do caso de teste foi expor como a Computação Ubíqua, SOA e Jini podem colaborar no cadastro, processamento e divulgação de informações relevantes a usuários envolvidos em um campeonato de futebol, estando presente ou não em um estádio.

Este sistema, então, prove aos usuários maior interação e acesso fácil e rápido a informações que são relevantes e, em tempo real da execução de uma partida de futebol, disponibilizando também dados sobre todo o campeonato e as entidades relacionadas.

Torcedores, imprensa e até mesmo a comissão técnica da equipe serão consumidores de dados da partida e do campeonato, dados esses que podem se transformar em informações preciosas e, ainda, ajudar na tomada de decisões durante a partida.

5.1 Sistema

Essa seção descreve o sistema como um todo, desde a sua implementação ao funcionamento. Existem dois conceitos de sistema definidos nesse trabalho, o sistema em si, englobando todos as características e serviços e o sistema de gerência de um campeonato de futebol, que, por si só, não exemplifica uma aplicação dos conceitos estudados nesse trabalho, contudo, deverá ser o suporte na qual a parte mais importante do estudo será estruturada.

Por esse motivo não foi exigido que se abordasse todos os requisitos que um sistema de gerencia deste porte necessita e, muito menos, foram analisadas todas as possíveis situações. Algumas mesmo analisadas, não foram implementadas para que o trabalho não se estendesse e, consequentemente, fugisse do foco principal. Desta forma quando for referenciado, o sistema é o todo, englobando o sistema de gerência.

Foi então construído um sistema simples, com a limitada função de gerenciar os dados que possuem informações sobre:

- Estádios;
- Campeonato;
- Equipes;
- Atletas;
- Partidas e;
- Opiniões sobre as partidas.

Estes dados podem ser acessados pelos mais variados sistemas e usuários, que podem ou não estar presente em uma partida de futebol do campeonato. O sistema fornecerá interação entre os usuários em tempo real e este assunto será mais bem detalhado na Seção 5.2. A implementação restringe algumas funcionalidades, o foco será apenas voltado para aqueles que estiverem conectados a rede local do evento.

5.1.1 Usuários do Sistema

Como possíveis usuários, podemos relacionar:

- **Comissão Organizadora:** é a responsável por todas as informações necessárias ao sistema. Pode cadastrar, alterar e remover qualquer informação, tendo o papel de super usuário.
- **Comissão Arbitragem:** tem a função de cadastro dos dados oficiais de uma partida, que são coletados por seus árbitros. Esta informação é coletada em tempo real e pode ser vista por usuários interessados.
- **Imprensa e torcedores:** são os principais consumidores das informações, desde as obtidas em tempo real durante a partida, como também, dados sobre o campeonato e tudo que ele compreende. Poderão expor suas opiniões e listar as opiniões cadastradas sobre a partida em andamento.

Em uma abordagem maior e mais detalhada, talvez, fosse possível identificar mais usuários, mas, com esses citados já é possível representar o escopo do trabalho.

5.2 Implementação

A implementação foi realizada com o uso da tecnologia Jini e tornou-se natural a utilização da tecnologia Java e o ambiente de desenvolvimento *Eclipse Helios*.

Pode-se visualizar a implementação do sistema em camadas e, dessa forma, analisar a relação entre elas. Na camada mais interna estão as definições das classes e interfaces, sobre ela está a camada Jini, servindo como ponte para a próxima camada, Clientes e Serviços, e na camada mais externa, está o tratamento para a interface com usuário final, facilitando o seu acesso. Todo esse processo pode ser visto na Figura 5.1.

A camada mais interna, das classes e interfaces, é acessível aos serviços e aos clientes por meio do Jini e é por meio dele que os clientes podem requisitar aos serviços as funcionalidades dos objetos que estão definidos nas classes e disponibilizados pelos serviços definidos no servidor Jini.

O usuário final é um cliente que pode acessar as funcionalidades dos objetos por meio de uma interface gráfica, representada na Figura 5.1 como a ultima camada.

O início da implementação se deu ao obter uma cópia do *Jini Starter Kit v2.1* [1]. Esse kit acompanha todas as bibliotecas necessárias para a imple-

[1] Obtido no portal do projeto: http://www.jini.org/wiki/Category:Jini_Starter_Kit, acessado em 10/1/2011

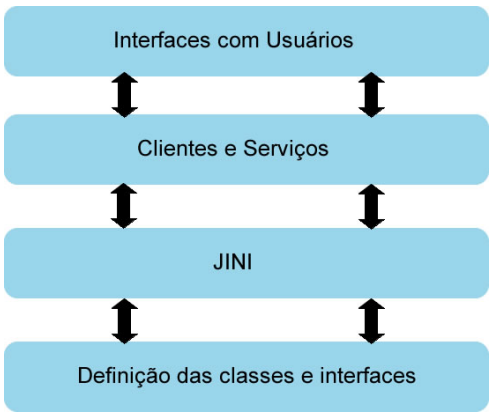

Figura 5.1: Relação entre as camadas nas quais o sistema se divide

mentação, alguns exemplos de pequenos sistemas Jini e toda a documentação necessária.

O kit também possui outras importantes ferramentas necessárias para o desenvolvimento, como os arquivos de configuração do servidor que hospedarão o ambiente Jini e o código fonte das classes do sistema Jini, possibilitando a alteração ou consulta do seu funcionamento.

A modelagem do sistema de gerência do campeonato de futebol será abordada na Seção 5.2.1. Na Seção 5.2.3 são definidos os serviços disponibilizados e os clientes consumidores destes serviços. Posteriormente será abordada a criação do banco de dados, Seção 5.2.2, e, por fim, na Seção 5.3, como foram definidas e criadas as classes necessárias para a execução do sistema.

5.2.1 Modelagem do sistema de Gerência do Campeonato de Futebol

Tendo em vista as restrições do projeto, objetivou-se a implementação do requisitos para que um sistema desse contexto suportasse naturalmente as tecnologias e conceitos estudados.

O sistema possui algumas entidades, as principais são:

- **Clube:** entidade base, possui outras entidades e representa a instituição que pode ou não estar participando de um campeonato;
- **Equipe:** entidade agregada a um clube e um campeonato. Um clube pode possuir várias equipes que participam de diferentes campeonatos. É na equipe que a entidade *Jogadores* é relacionada;
- **Campeonato:** entidade que é base para todas as que participam da competição. Um sistema pode administrar mais de um campeonato;

- **Jogadores:** é a entidade que representa os jogadores que são atribuídos a uma *Equipe*;
- **Estádios:** entidades que representam os locais onde podem ocorrer as disputas. São atrelados aos Clubes;
- **Evento da Disputa:** entidade que armazena as informações registradas durante a execução de uma partida, tais eventos podem ser, por exemplo, gols, cartões e faltas. Esses eventos são relacionados a um jogador e uma disputa;
- **Disputa:** entidade que representa a disputa e está relacionada a duas equipes, um campeonato e possui eventos. Uma disputa é realizada em um estádio e possui informações particulares como, por exemplo, data e horário da partida;
- **Opiniões:** são vinculadas em uma disputa e representam informações a respeito da partida e são enviadas, por exemplo, por torcedores e imprensa.

O relacionamento entre elas será melhor abordado na Seção sobre a base de dados (Seção 5.2.2).

A modelagem dos dados dividiu o sistema em dois grupos principais: de um lado, o sistema de gerência em si e do outro os clientes e serviços. É importante ter conhecimento que tudo no projeto foi realizado tentando aproveitar o máximo das tecnologias SOA e Jini, portanto as classes de manipulação de dados servem tanto para o sistema de gerência, pensado para ser utilizado em um ambiente fixo e local, quanto para serem utilizadas por clientes por intermédio de serviços disponíveis na rede.

O projeto roda em cima de bibliotecas e arquivos de configurações pré-definidos pelo padrão do Jini sendo um deles, particularmente importante: **Jini Starter Service**. Foi necessário criar uma classe Java que executasse um arquivo de configuração para ser possível iniciar os serviços Jini e manter o *lookup service* e a rede disponível. Esse processo será melhor representado na Seção 5.3, mas para ter uma visão geral é essencial saber que foi utilizado desta forma.

5.2.2 Base de dados

A modelagem utilizada para a criação da base de dados possui as seguintes entidades:

- clube;

- equipe;
- campeonato;
- jogadores;
- estadios;
- curiosidade_estadio;
- evento_partida;
- tipo_eventos;
- disputa;
- opinioes;
- tipo_opniao.

Utilizando o programa *MySQL Workbench* pode-se extrair uma imagem contendo as entidades da base de dados, como visto na Figura 5.2.

Os dados são todos gerenciados pela classe java *SistemaCadastroDados*, pertencente ao pacote *br.organizacao.sistema*. Ela disponibiliza métodos para que outras classes possam fazer uso do banco.

5.2.3 Serviços e Clientes

O sistema de gerência dos campeonatos trabalha integrado com o sistema de serviços e clientes. Os serviços desse projeto servem, basicamente, para possibilitar aos clientes acesso aos dados do sistema, criando, alterando e listando. Os serviços acessam as funções implementadas nas classes e disponibilizam-nas na rede em forma das funcionalidades dos serviços.

Os serviços que foram implementados são:

- Serviço de Cadastro de dados do Campeonato;
- Serviço de Cadastro de dados do Estádio;
- Serviço de Cadastro de dados da Partida;
- Serviço de Cadastro de Opinião;
- Serviço de Listagem de dados do Campeonato;
- Serviço de Listagem de dados do Estádio;
- Serviço de Listagem de dados da Partida;

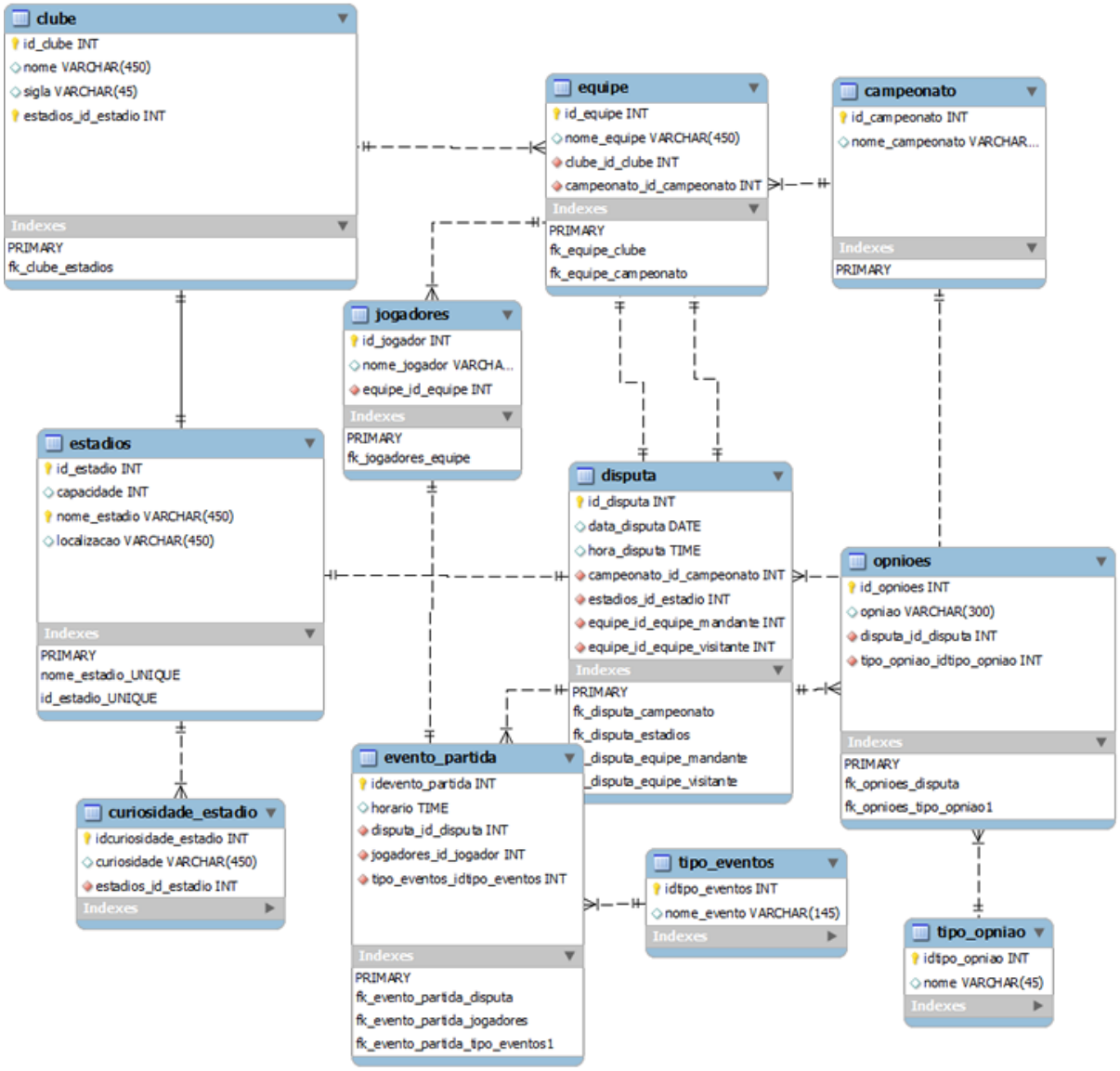

Figura 5.2: Modelo das entidades da base de dados do sistema

- Serviço de Listagem de Opiniões.

Os clientes conhecem a *interface* das classes e com esse conhecimento, buscam na rede serviços que disponibilizam objetos dessas *interfaces*.

Os clientes consumidores desse serviço são:

- Cliente de Listagem de dados do Estádio;
- Cliente de Cadastro da Comissão de Arbitragem;
- Cliente da Imprensa;
- Cliente do Torcedor.
- Cliente Estadio

Os clientes consomem um ou mais serviços, respeitando assim mais um aspecto do SOA, ou seja, reutilizamos os serviços para construir outros serviços maiores(Composição de Serviços).

Detalhes da implementação dos serviços e clientes são listados na Seção 5.3. Na Figura 5.3 é mostrado como que as classes implementadas se relacionam com os clientes e serviços.

Os clientes possuem interface gráfica com o usuário e são elas: Figura 5.4 e Figura 5.5.

5.3 Definição e criação das Classes

As classes mais importantes para o funcionamento do sistema e para a utilização dos serviços e clientes estão evidenciadas a seguir. De maneira geral, as classes foram implementadas com empenho em utilizar os conceitos SOA, principalmente os conceitos de modularidade e reusabilidade.

A implementação foi realizada no ambiente Eclipse e possuiu um principal projeto, que foi dividido em pacotes, os quais são aglomerados de classes logicamente organizados. Essa divisão foi feita da seguinte forma (listados por ordem alfabética):

- **br.arbitragem:** Representando a comissão técnica, responsável pelo abastecimento do sistema das informações oficiais da disputa.
- **br.comum.cliente:** Possui as classes comuns e públicas, reservadas para que, por exemplo, torcedores e imprensa pudessem cadastrar opiniões e acessar informações sobre o campeonato e a disputa.
- **br.imprensa:** Cliente que especifica a invocação de serviços comuns como sendo para a imprensa.

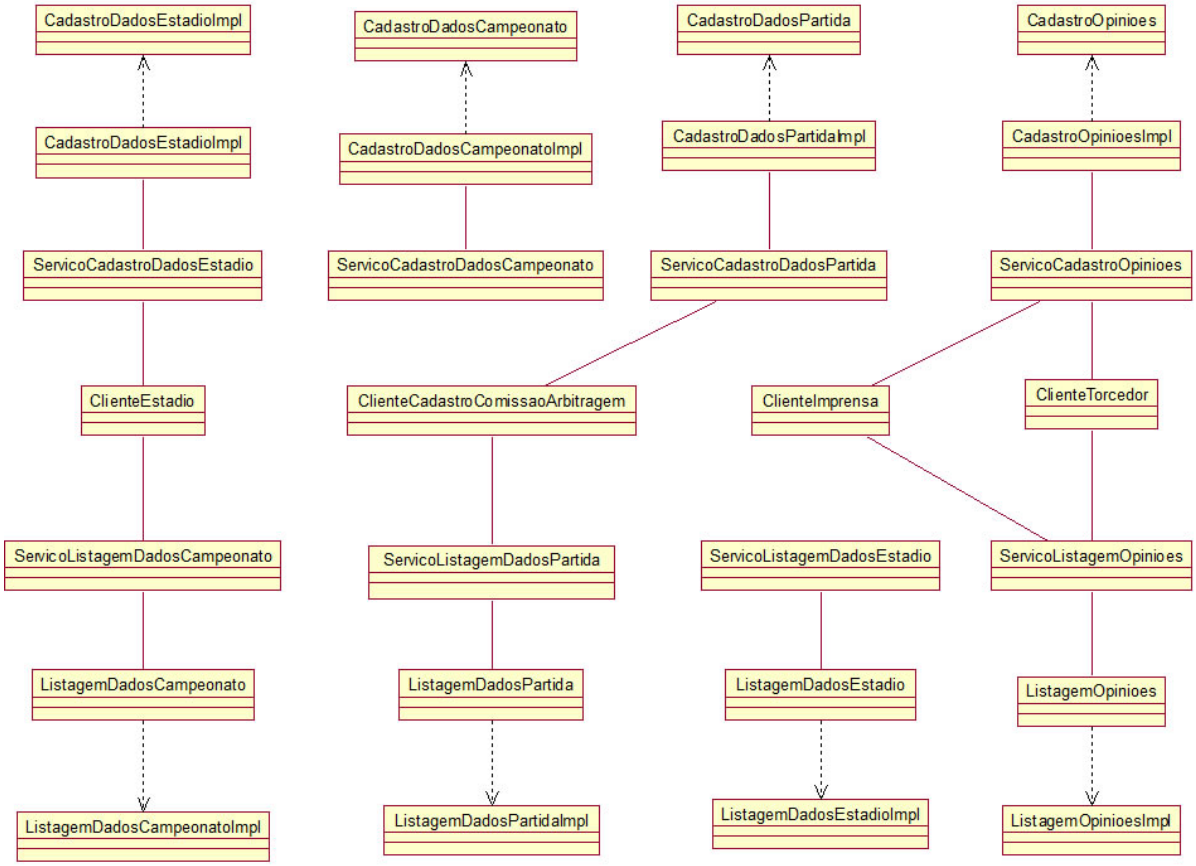

Figura 5.3: Modelo simplificado da relação entre Clientes e Serviços.

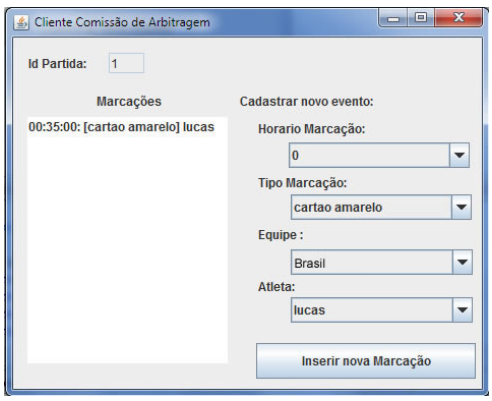

Figura 5.4: Interface gráfica de utilização do cliente da Comissão de Arbitragem

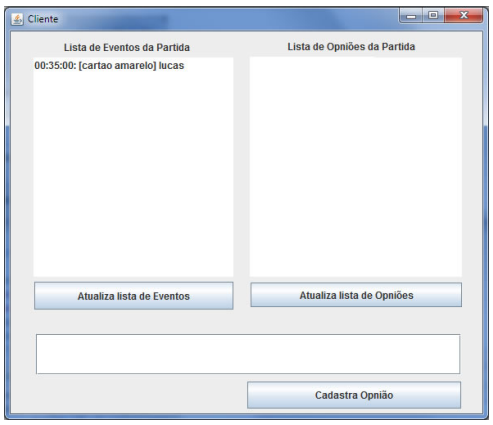

Figura 5.5: Interface gráfica de para os clientes Torcedor e Imprensa. Cadastro de opniões e listagens de dados da partida

- **br.organizacao:** Pacote macro, que armazena várias classes relacionadas a organização do campeonato e seus dados.
- **br.organizacao.cadastro:** Todas as classes responsáveis pelo cadastro de dados, não só do campeonato, mas também de opiniões.
- **br.organizacao.cadastro.servico:** Todos os serviços usados para cadastro de dados.
- **br.organizacao.cadastro.listagem:** Todas as listagens e dados existentes no sistema.
- **br.organizacao.cadastro.listagem.servico:** Serviço para que os dados possam ser listados nos clientes.
- **br.organizacao.sistema:** Classes responsáveis pelo sistema em si, possuindo todas as classes de cadastro,alteração e exclusão de dados.
- **br.torcedor:** similar ao *imprensa*, identifica o acesso aos dados comuns pelos torcedores.

Com essa descrição já é possível perceber a divisão entre serviços e clientes, que será melhor discutida na Seção 5.2.3.

Fica evidenciado, por meio de sua estrutura, que o sistema possui classes específicas para manipular dados e essas são reutilizadas por outras, evidenciando assim uma características de SOA, a reusabilidade.

5.3.1 Classes do sistema de gerência e suas interfaces gráficas

A criação e o funcionamento das principais classes responsáveis pelo sistema e também suas interfaces gráficas são descritas abaixo. A ordem em que

aparecem será alfabética pelo pacote em que estão organizadas. Não são evidenciados métodos comuns, apenas os interessantes para entendimento do sistema e os conceitos do projeto.

Existem classes que possuem interfaces e implementações, essas classes são as que são disponibilizadas pelos serviços e, portanto, necessitam de uma interface que seja conhecida pelo cliente e uma implementação que será disponibilizada para que o mesmo possa usufruir de suas funções. As interfaces são *serializadas*, pois são trafegadas na rede entre os componentes Jini.

br.organizacao.cadastro.CadastroDadosEstadio:

Responsável pelo cadastro de informações e curiosidades sobre o estádio. Possui interface e implementação.

```
public interface CadastroDadosEstadio extends Serializable
       public void cadastroCuriosidadesEstadio(Integer idEstadio,
       String curiosidade);
       public void cadastroEstadio (Integer capacidade,
       String nomeEstadio, String localizacao);
```

br.organizacao.cadastro.CadastroDadosPartida:

Responsável pelo cadastro de dados sobre a partida, que são fornecidos principalmente, pelo cliente *Comissão de Arbitragem*. Possui interface e implementação.

```
public interface CadastroDadosPartida extends Serializable
public Integer getIdPartidaAtual();
public Vector<String> listagemEventosPartida(Integer idPartida);
public Vector<String> listaJogadoresEquipe (String nomeEquipe);
public Vector<String> listaEquipesDisputa(Integer idPartida);
public Vector<String> listaEventos ();
public void cadastroEventoPartida (Integer codDisputa,
Time horario, String tipoMarcacao, String atletaInfrator);
```

br.organizacao.cadastro.CadastroOpnioes:

Responsável pelo cadastro de opiniões durante uma partida,que são fornecidas, principalmente, pelos clientes: *Torcedores* e *Imprensa*. Possui interface e implementação.

```
public interface CadastroOpnioes extends Serializable
public Integer getIdPartidaAtual();
public void cadastroOpniaoTorcedor(String opniao);
public void cadastroOpniaoImprensa(String opniao);
```

br.organizacao.listagem.ListagemDadosEstadio:

Responsável pela listagem de dados sobre o estádio, podendo ser tanto sobre sua localização física, sua capacidade ou até mesmo uma lista de curiosidades sobre um estádio. Possui interface e implementação.

```
public interface ListagemDadosEstadio extends Serializable
public Vector<String> listaEstadios();
public Vector<String> curiosidadesEstadio(String nomeEstadio);
public Integer getCapacidadeEstadio(String nomeEstadio);
public String localizacaoEstadio (String nomeEstadio);
```

br.organizacao.listagem.ListagemDadosPartida:

Responsável pela listagem de dados da partida. Talvez seja a classe mais utilizada e consumida pelos clientes desse sistema. Possui interface e implementação.

```
public interface ListagemDadosPartida extends Serializable
public Vector<String> listagemGols();
public Vector<String> listagemCartoes();
public Vector<String> listagemSubstituicoes();
public Vector<String> listagemFaltas();
public Vector<String> listagemEventosPartida (Integer idPartida);
```

br.organizacao.listagem.ListagemOpnioes:

Realiza a listagem de opiniões da imprensa ou dos torcedores, retornando uma lista de *String*. Possui interface e implementação.

```
public interface ListagemOpnioes extends Serializable
public Vector<String> listagemOpnioesTorcedores();
public Vector<String> listagemOpnioesImprensa();
```

br.organizacao.sistema.SistemaCadastroDadosCampeonato:

Classe responsável pela manipulação do banco de dados. Todas as classes que necessitam acessar ao banco realizam por intermédio desta classe. Ela possui métodos simples, mas extremamente importantes para o funcionamento do sistema e não são listados, já que não possuem nenhuma programação específica para este trabalho, são funções de cadastro, edição e listagem e que são em outros momentos citados como parte de outra classe que utilize estes serviços. Em sequência são analisadas as interfaces gráficas do sistema de gerência que foram construídas visando facilitar a manipulação dos dados e são utilizadas pelo administrador do sistema.

br.organizacao.sistema.JanelaDadosCampeonato:

Interface gráfica para manipulação de campeonatos, visando o cadastro, edição e exclusão de um campeonato, como visto na Figura 5.6.

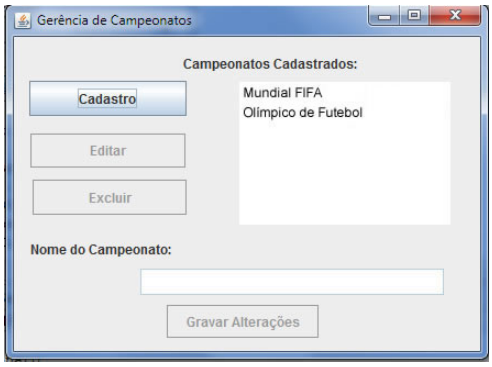

Figura 5.6: Interface gráfica de Cadastro de Dados do Campeonato.

br.organizacao.sistema.JanelaDadosClubesEquipes:

Interface gráfica para manipulação dos dados das equipes e clubes. Lembrando que um clube pode possuir mais de uma equipe e essa equipe necessita estar vinculada a um campeonato. Um clube necessariamente possuí um estádio, que será o utilizado em jogos nos quais o clube sediará, como visto na Figura 5.7.

br.organizacao.sistema.JanelaDadosDisputas:

Interface gráfica para manipulação das disputas, atrelando um campeonato e duas equipes, definindo o horário e o dia em que ela ocorrerá. Ao lado é possível verificar uma lista de disputas já cadastradas, como pode ser observado na Figura 5.8.

br.organizacao.sistema.JanelaDadosEstadios:

Interface gráfica para manipulação dos dados dos Estádios, cadastrando e editando informações sobre sua localização e capacidade, evidenciado na Figura 5.9.

br.organizacao.sistema.JanelaEdicaoDadosJogadores: e

br.organizacao.sistema.JanelaListagemDadosJogadores:

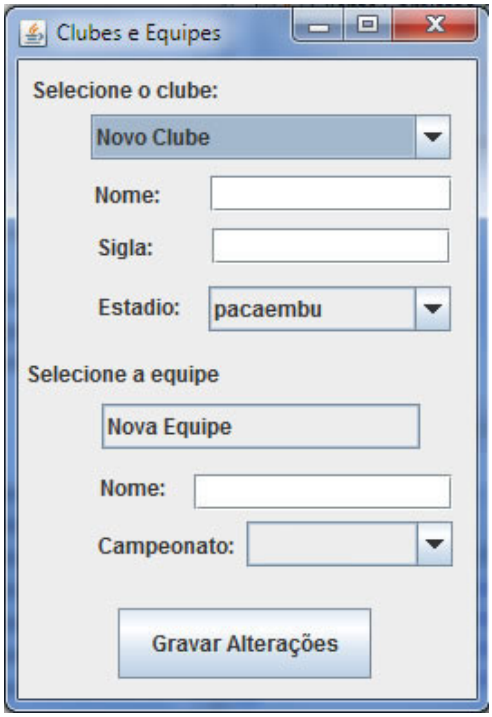

Figura 5.7: Interface gráfica de manipulação de dados das Equipes e Clubes.

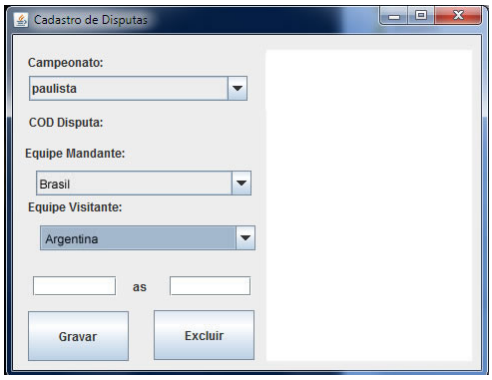

Figura 5.8: Interface gráfica de manipulação das Disputas.

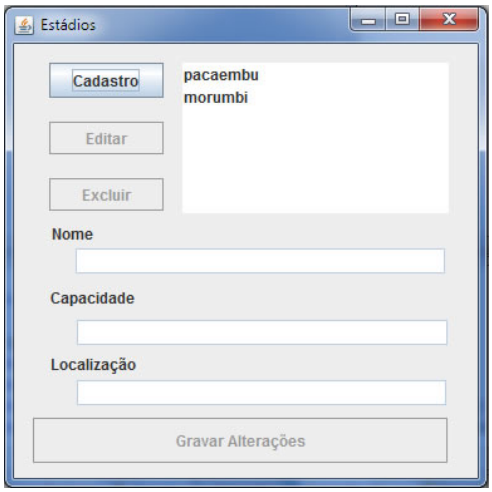

Figura 5.9: Interface gráfica de manipulação dos Estádios.

Interface gráfica para manipulação dos dados dos Jogadores, podendo incluir, remover e listar dados, como observado na Figura 5.10.

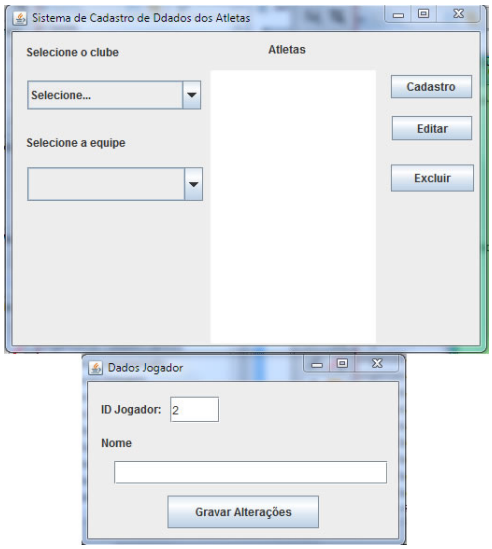

Figura 5.10: Interface gráfica de manipulação dos Jogadores.

br.organizacao.sistema.JanelaSistemaCadastroDadosCampeonato:
Interface gráfica central, para acesso as demais interface gráficas de gerência, como pode ser visto na Figura 5.11.

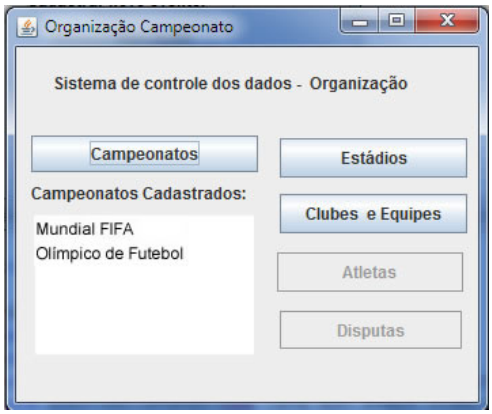

Figura 5.11: Interface gráfica de central da Gerência do Sistema.

5.3.2 Classes dos Serviços e Clientes

Essa é a implementação mais importante para esse projeto, portanto será bem detalhada, visando abordar todos os requisitos necessários para o funcionamento do sistema de serviços e clientes.

Primeiramente, é necessário iniciar o *lookup service* e para isso, foi criada uma pequena classe: *JiniServiceStarter*. Essa classe possui apenas algumas

linhas e sua principal função é ativar os programas que já veem junto com o Kit Jini.

```
import com.sun.jini.start.ServiceStarter;
public class JiniServiceStarter
public static void main(String[] args)
String configFileName;
if ( args.length!=0 )
configFileName = args[0];
 else
configFileName = "start-services.config";

String[] options =  configFileName ;
// Inicía um conjunto de serviços
ServiceStarter.main( options );
System.out.println( "Serviços iniciados...");
```

Esse arquivo *start-services.config* é chave para iniciar os sistemas básicos e seu conteúdo é evidenciado abaixo e dividido em quatro partes.

Primeiro o arquivo define as configurações do servidor como nome, localização e porta de conexão, utilizando classes provenientes do Kit Jini:

```
import com.sun.jini.config.ConfigUtil;
import com.sun.jini.start.NonActivatableServiceDescriptor;
import com.sun.jini.start.ServiceDescriptor;
import java.io.File;
com.sun.jini.start
jini_home = System.getenv("JINI_HOME");
codebase_host = "localhost";
codebase_port = "8080";
codebase_url = "http://" + codebase_host + ":" + codebase_port + "/";
jini_lib = jini_home + File.separator + "lib" + File.separator; //
```

Neste ponto o arquivo define as diretrizes para rodar o servidor *HTTPD* e inicia o serviço:

```
//
// HTTPD Service
//
private static httpd_codebase =;
private static httpd_policy = "all.policy";
private static httpd_classpath = jini_lib + "classserver.jar";
private static httpd_impl = "com.sun.jini.tool.ClassServer";
private static httpd_service =
new NonActivatableServiceDescriptor(
httpd_codebase,
httpd_policy,
httpd_classpath,
httpd_impl,
new String[]
-port",
"8080",
-dir",
"www",
-verbose"

);
```

Agora o arquivo utiliza as informações, definidas acima, para realmente iniciar o *Reggie*. Importante observar essa parte do arquivo inserindo os aplicativos fornecidos pelo Kit, como por exemplo *reffie.jar*:

```
//
// Reggie (Lookup Service)
//
private static reggie_codebase =
codebase_url + "classes/reggie-dl.jar" +
codebase_url + "classes/jsk-dl.jar" +
codebase_url + "classes/";
private static reggie_policy = "reggie.policy";
private static reggie_classpath = jini_lib +
"reggie.jar"+ File.pathSeparator +
jini_lib + "reggie-dl.jar";
private static reggie_impl =
"com.sun.jini.reggie.TransientRegistrarImpl";
private static reggie_config = "jeri-reggie.config";
private static reggie_service =
new NonActivatableServiceDescriptor(
reggie_codebase,
reggie_policy,
reggie_classpath,
reggie_impl,
new String[]
reggie_config

);
```

Por fim é criado um descritor de serviços:

```
//
// Serviços
//
static serviceDescriptors = new ServiceDescriptor[]
httpd_service,
reggie_service
;
//end com.sun.jini.start
```

Importante observar que durante esse processo são utilizados vários aplicativos, os seguintes mais importantes listados abaixo:

- *jini-core.jar*;
- *jini-ext.jar*;
- *jsk-lib.jar*;
- *jsk-resourses.jar*;
- *jsk-plataform.jar*;
- *tools.jar*;
- *reggie.jar*.

Além disso utiliza-se os seguintes arquivos de configuração:

- *all.policy*;
- *jeri-reggie.config*;
- *jrmp-reggie.config*;
- *reggie.policy*.

Com esse procedimento já está configurado o ambiente para que os serviços e clientes, que são abordados posteriormente, possam se encontrar.

Foram criados vários serviços nesse trabalho, listados abaixo as classes nas quais eles foram implementados:

- ServicoCadastroDadosCampeonato.java;
- ServicoCadastroDadosEstadio.java;
- ServicoCadastroDadosPartida.java;
- ServicoCadastroOpnioes.java;
- ServicoListagemDadosCampeonato.java;
- ServicoListagemDadosEstadio.java;
- ServicoListagemDadosPartida.java;
- ServicoListagemOpnioes.java.

Como eles possuem a mesma estrutura, não é necessário exemplificar todos aqui, portanto será demonstrado como foi realizada a implementação do serviço *ServicoCadastroDadosEstadio.java*

Para que um serviço possa ser construído é necessário conhecer /it a priori a classe que é disponibilizada na rede, para o serviço analisado, a classe

br.organizacao.cadastro.CadastroDadosEstadioImpl, portanto ela será inserida como *import*.

A classe necessita de um método *public static void main()* que será executado inicialmente, esse método fará a simples tarefa de colocar a *thread* atual para "dormir", isso se aplica a situações em que o serviço requer alguma resposta da rede (clientes por exemplo) para realizar outra tarefa. Essa funcionalidade não foi utilizada nesse serviço, portanto não foi implementada. É ela que inicializa o objeto ServicoCadastroDadosEstadio();

```
public static void main(String argv[])
new ServicoCadastroDadosEstadio();
// ficar ativa até que receba uma interrupção
try
Thread.currentThread().sleep(1000000L);
 catch(java.lang.InterruptedException e)
// execute ação, não implementada nesse caso.
```

Com o objeto instanciado, o construtor da classe é chamado e tem a função de, resumidamente, encontrar o *lookup service* e enviar o objeto da implementação, se tornando um serviço. Perceba que nesse exemplo estamos procurando por qualquer *lookup service* em qualquer grupo.

```
public ServicoCadastroDadosEstadio()
System.setSecurityManager(new RMISecurityManager());
JoinManager joinMgr = null;
try  LookupDiscoveryManager mgr =
new LookupDiscoveryManager(LookupDiscovery.ALL_GROUPS,
null /* unicast locators */,
null /* DiscoveryListener */);
joinMgr = new JoinManager(new CadastroDadosEstadioImpl(),
null,
this,
mgr,
new LeaseRenewalManager());
 catch(Exception e)
e.printStackTrace();
System.exit(1);
```

Para que possamos rodar essa aplicação é importante colocar como argumento para a máquina virtual a informação sobre qual o tipo de politica que esse serviço está sujeito, pois será essa uma forma de restringir acesso a rede e ao serviço. Nesses exemplos não houve foco na segurança, portanto, foi utilizado um arquivo padrão. Na máquina virtual utilizou-se esse argumento:

```
-Djava.security.policy=all.policy
```

E esse arquivo de configuração, *all.policy*, deve ser desse modo:

```
/* GARANTE TODAS AS PERMISSÕES PARA AS CLASSES */
grant
permission java.security.AllPermission;
;
```

Com esse procedimento e um *lookup service* ativo na rede o serviço já se registra e fica disponível para ser consumido. Esse processo foi o mesmo utilizado para os outros serviços.

O cliente necessita encontrar um *lookup service* e nele encontrar o serviço pelo qual procura. Para poder encontrar-lo, necessita conhecer a interface da aplicação que o serviço disponibiliza.

Os clientes implementados nesse projeto são:

- ClienteCadastroComissaoArbitragem.java;
- ClienteListagemDadosEstadio.java;
- ClienteImprensa.java;
- ClienteTorcedor.java.

O funcionamento do cliente pode ser resumido em 3 principais etapas: primeiro encontrar o *lookup service*, depois procurar por um serviço utilizando uma interface já conhecida e em terceiro lugar receber o objeto do serviço e utilizar sua funcionalidade. Nesse projeto alguns clientes utilizam mais de um serviço e isso é feito de forma semelhante, apenas armazenando os objetos e utilizando-os quando convém.

Para demonstrar a implementação de um cliente, será utilizado *ClienteTorcedor.java*. Esse cliente utiliza mais de um serviço e após obtido os objetos ele instancia uma interface gráfica, na qual se pode usufruir de suas funcionalidades.

Um cliente necessita implementar a interface *net.jini.discovery.DiscoveryListener*, como mostrado abaixo:

```
public class ClienteTorcedor implements DiscoveryListener
```

E nesse exemplo são utilizados os seguintes objetos:

```
public br.organizacao.cadastro.CadastroOpnioes objCadORG;
public br.organizacao.listagem.ListagemOpnioes objListOrg;
public br.organizacao.listagem.ListagemDadosPartida objListDados;
public JanelaClienteListagemEventoCadOpniao janelaCliente;
```

Os três primeiros são relacionados aos objetos que chegarão dos clientes e o último é a interface gráfica que será utilizada por esse cliente.

O construtor dessa classe será o responsável por procurar pelo *lookup service*, como visto abaixo:

```
public ClienteTorcedor()
System.setSecurityManager(new RMISecurityManager());
LookupDiscovery discover = null;
try
discover = new LookupDiscovery(LookupDiscovery.ALL_GROUPS);
 catch(Exception e)
System.err.println(e.toString());
System.exit(1);

discover.addDiscoveryListener(this);
```

A classe *public static void main()* instancia a classe e depois colocar a *thread* atual para "dormir"enquanto não receber alguma resposta, processo idêntico ao executado pelo mesmo método no servidor:

```
public static void main(String[] args)
ClienteTorcedor f = new ClienteTorcedor();
// ficar ativa até que receba uma interrupção
try
Thread.currentThread().sleep(10000L);
 catch(java.lang.InterruptedException e)
// execute ação, não implementada neste caso
```

Por implementar a *interface DiscoveryListener* essa classe necessita implementar dois principais métodos:

```
public void discovered(DiscoveryEvent evt)
public void discarded(DiscoveryEvent evt)
```

O primeiro é, logicamente, invocado quando o construtor encontra um *lookup service* e é nele que os objetos são recebidos da rede. O segundo método é chamado quando, por algum motivo, seja o cliente, o servidor, ou outro fato, requer a desconexão da rede.

Nesse exemplo ficará evidente que o processo de aquisição que foi utilizado é bem semelhante para todos os outros clientes e é realizado da seguinte maneira:

Definir qual o serviço que deseja:

```
ServiceRegistrar[] registrars = evt.getRegistrars();
Class[] classes = new Class[]
br.organizacao.cadastro.CadastroOpnioes.class;
ServiceTemplate template = new ServiceTemplate(null, classes,
null);
```

Recuperar o objeto para cadastro de opiniões:

```
for (int n = 0; n < registrars.length; n++)
ServiceRegistrar registrar = registrars[n];
ServiceMatches matches;
try
matches = registrar.lookup(template, 10);
 catch(java.rmi.RemoteException e)
e.printStackTrace();
continue;

for (int m = 0; m < matches.items.length; m++)
objCadORG = (br.organizacao.cadastro.CadastroOpnioes)
matches.items[m].service;
System.out.println("objeto adiquirido cad opniao");
objCadORG.conectaServidor();
```

Define o próximo objeto a ser adquirido:

```
classes = new Class[]
br.organizacao.listagem.ListagemOpnioes.class;
template = new ServiceTemplate(null, classes,
null);
```

E recupera o novo objeto, da mesma forma do anterior. Faz esse procedimento mais uma vez, só que agora para o objeto *br.organizacao.listagem.ListagemDadosPartida.class*.

Não se deu importância para o caso de encontrar mais de um serviço correspondente para a mesma interface, pois o projeto foi realizado em um ambiente controlado e com a certeza que existe apenas um serviço por interface existe. Esse é mais um ponto que será tratado como melhoria para próximas abordagens, no Capítulo 6.

Após receber os três objetos é instanciada a interface gráfica, da qual o usuário poderá usufruir dos serviços:

```
janelaCliente = new JanelaClienteListagemEventoCadOpniao(
objCadORG.getIdPartidaAtual(), objCadORG,
objListOrg,
objListDados,
//1 - torcedor 0 - imprensa
1);
janelaCliente.setVisible(true);
```

É dessa maneira para os demais clientes. A utilização dos serviços é realizada na interface gráfica dos clientes, conforme mostrado nas imagens do Capítulo 5.2.3.

5.4 Funcionamento do sistema

Para que o sistema possa ser utilizado é necessário que os serviços Jini estejam disponíveis, esse processo será abordado na Seção 5.4.1. O cliente necessita que o serviço esteja ativo, portanto, a segunda etapa é disponibilizar o serviço, abordado na Seção 5.4.2. Com os serviços disponíveis, os clientes podem ser inicializados e estão aptos a consumir os servidos, como analizado na Seção 5.4.3. O cliente possui uma interface gráfica como o usuário final, podendo ele ser, por exemplo, um cliente Torcedor que queira listar e cadastrar a sua opinião, esse processo pode ser visto na Seção 5.4.4.

5.4.1 Iniciando o Servidor

Para iniciar os serviços básicos para se manter um sistema Jini funcionando, será utilizada a classe *JiniServiceStarter*, já citada na Seção 5.3.2.

Para rodar essa classe, é necessário passar o seguinte argumento para a Máquina Virtual:

```
-Djava.security.policy=all.policy
```

Quando executada, a classe executa o servidor e, como configurado nesse exemplo, fica escutando a porta 8080 no servidor *localhost*. Todos os serviços e clientes que requisitarem uma conexão irão utilizar esse servidor, pois é o único ativo na rede que o projeto foi testado.

5.4.2 Iniciando os Serviços

Primeiramente é necessário iniciar os serviços para depois poder iniciar os cliente. Para exemplificar, será demonstrado como é realizado no serviço *ServicoCadastroOpnioes*.

Quando executado, o serviço requisita o *ID* da partida na qual as opiniões podem ser cadastradas, nesse exemplo, exemplificado caso o administrador do sistema tenha inserido o *ID* 1 como sendo a partida atual:

```
ID da partida:
1
- Serviço Adquirido!  ID d703fed6-607b-488f-a3f3-1e5a53f734a6 -
```

Perceba que é recebido da rede (*lookup service*) um ID, que é único para esse serviço. Realizado isso, o serviço está disponível e oferecendo o objeto de cadastro de opiniões para qualquer cliente na rede que requisitar.

Esse processo é análogo aos demais serviços e não são exemplificados.

5.4.3 Iniciando os Clientes

Para iniciar o cliente, o processo é bastante parecido com o exemplificado anteriormente, para os serviços (Seção 5.4.2), internamente a diferença é maior, como abordado na Seção 5.3 e na Seção 4.5, mas se analisado exteriormente, o processo é semelhante.

O cliente requer que o serviço que utilizará esteja disponível na rede e pelo *lookup service* faz a requisição. Após obter o objeto, utiliza-o como desejar.

Para exemplificar, será utilizado o cliente *ClienteTorcedor*. Para ser executado, o arquivo *all.policy* deve estar acessível e o seguinte argumento presente na Máquina Virtual:

```
-Djava.security.policy=all.policy
```

Além disso, especificamente esse cliente, requer que mais de um serviço estejam disponíveis na rede:

- *ServicoListagemDadosPartida*: Necessário para que possa acessar e listar os dados referente a partida.
- *ServicoCadastroOpnioes*: Necessário para poder cadastrar as opniões sobre a partida.
- *ServicoListagemOpnioes*: Necessário para poder acessar as opiniões que já foram cadastradas sobre a partida.

O processo de registo de um serviço no *lookup service* e como a requisição é realizada pelo cliente é representado no diagrama de sequência na Figura 5.12, na qual o serviço, Serviço de Cadastro de Opiniões, requer o seu registro no *lookup service* e um cliente, Cliente Torcedor, interessado nesse serviço, faz a requisição ao *lookup service*, que retorna o objeto do serviço.

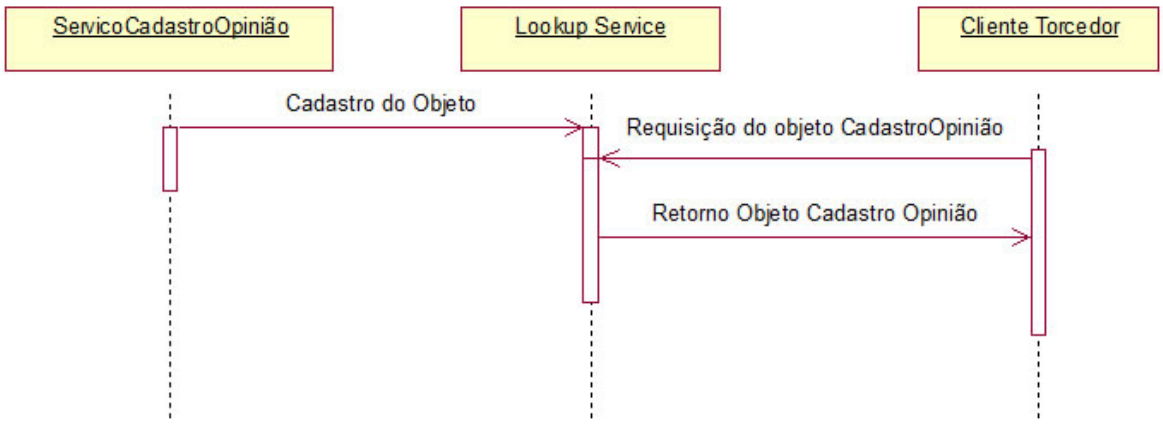

Figura 5.12: Diagrama de Sequência para o Serviço de Cadastro de Opinião e o Cliente Torcedor.

Se esses três serviços estiverem disponíveis instancia-se a interface gráfica, representada na Figura 5.5.

5.4.4 Utilizando o cliente

Para exemplificar o funcionamento do acesso realizado por um cliente, utilizaremos o Cliente Torcedor, que consume os serviços: Listagem de dados da Disputa, Listagem de Opiniões e Cadastro de Opiniões. Inicia-se os serviços de listagem e cadastro inserindo o código referente a disputa atual, isso fará com que o cliente possa apenas listar e cadastrar os dados referentes a essa disputa.

Depois inicia-se o cliente "“ClienteTorcedor"', que buscará os objetos para listagem e cadastro de opiniões e também o objeto de listagem de dados dados da disputa, com esses objetos o cliente inicia a interface gráfica "“JanelaClienteListagemEventoCadOpniao"', como por ser visto na Figura 5.13.

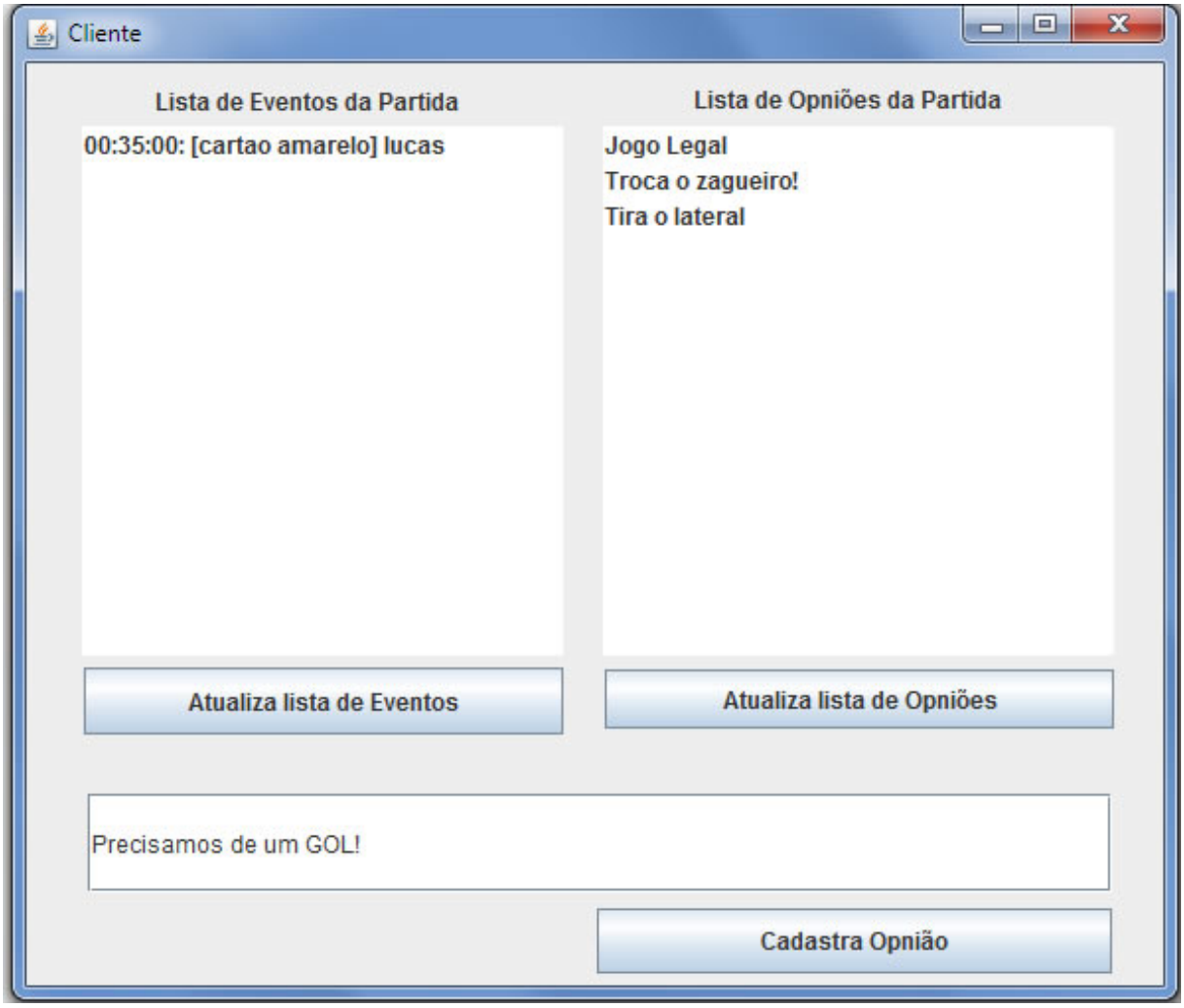

Figura 5.13: Interface gráfica do Cliente Torcedor.

O primeiro campo é referente a listagem de dados da partida e o botão abaixo dele faz uma consulta dos eventos que estão ocorrendo, por meio do objeto disponibilizado pelo serviço correspondente.

O campo exposto na direita é referente as opiniões da partida e o botão

abaixo faz a atualização das opiniões. O campo localizado no inferior da interface gráfica possibilita ao cliente o cadastro de uma opinião.

Capítulo 6

Considerações finais

Este trabalho apresenta um caso de teste para aplicabilidade da tecnologia Jini, em um cenário de Computação Ubíqua, utilizando os conceitos de SOA, ou seja, se propôs a encontrar um ambiente onde se pudesse ter um exemplo de Computação Ubíqua e implementar uma solução em que os conceitos de SOA pudessem ser úteis para a utilização da tecnologia Jini.

Durante a realização do presente projeto foram compreendidos diversos conceitos que não são habitualmente abordados pelas disciplinas do curso de graduação desta universidade, o que possibilitou uma rica pesquisa e, consequentemente, um grande enriquecimento para a formação e consolidação do aprendizado.

Conforme analisado ao longo do trabalho, os conceitos de SOA e Computação Ubíqua fundem-se naturalmente, embora não tenham sido abordados dessa maneira em sua criação e nem essa ser a principal característica de cada um, mas como observado, se forem tratados da maneira na qual foram nesse projeto, eles se complementam.

A implementação conseguiu fundir os conceitos e foi o uso da tecnologia Jini que tornou possível que, sem a necessidade de criar muitas classes e sem se importar com a maneira como as operações "baixo nível"funcionavam, o projeto apresentasse uma solução sólida e dentro do cronograma.

A escolha do caso de teste facilitou a implementação do processo, possibilitou a visualização da importância desse estudo para a computação e, ainda como estes conceitos podem contribuir para um evento que envolve milhões de pessoas.

O uso da tecnologia Jini nesse projeto, evidenciou a facilidade e robustez para gerência de aplicações em rede. O modo *plug-and-play* que trata os dispositivos conectados em sua rede se mostra indispensável para um sistema dinâmico, muito próximo do que deve ser encontrado em aplicações de Computação Ubíqua.

6.1 Dificuldades Enfrentadas

Apesar de ser muito interessante e sólida, a tecnologia Jini está "esquecida"pelos profissionais da área. Durante o desenvolvimento desse trabalho houve uma enorme dificuldade em encontrar materiais, livros e tutoriais, mais recentes. Foi necessário trabalhar com recursos mais antigos, que em média são da década passada. Esse fato é prejudicial, visto que a tecnologia, de *hardwares* e *softwares*, existente hoje é bem superior a encontrada quando foi desenvolvido o Jini, que mesmo suprindo a demanda desse projeto com sucesso, evidencia que é possível melhorar.

Outro ponto que retardou a realização do projeto foi a necessidade da implementação de um sistema genérico de cadastro, alteração e listagem de dados para que baseado neste sistema, se utilizasse o Jini. Mesmo não sendo o foco do trabalho, sem um sistema desse tipo, seria impossível a implementação e o funcionamento da arquitetura do projeto.

6.2 Perspectivas Futuras

A execução desse projeto incentiva o uso dos conceitos analisados em outras situações, já que se evidenciou a facilidade e a aplicabilidade dos mesmos.

O uso do SOA foi satisfatório mas, algumas características não foram abordadas por diversos motivos, entre eles o tamanho do projeto, que foi relativamente pequeno em sua implementação. Se o projeto for explorado mais minuciosamente e expandido será possível aplicar mais características.

O uso do Jini foi realizado de maneira básica e não foram implementados diversos pontos importantes em sua tecnologia, como é o caso de grupos de usuários dos serviços, a utilização de mais de um *lookup service* e até mesmo uma maior preocupação com os aspectos relativos a segurança, não só de como as classes eram manipuladas, mas também em relação a visibilidade dos serviços.

Outro aspecto que pode e deve ser abordado e, ainda apresentará ótimo crescimento, no caso de ser explorado de maneira correta, é a utilização dos conceitos em um sistema que já exista, exigindo uma interação entre eles é um outro ponto que pode ter um ótimo crescimento se explorado de maneira correta.

O caso de teste pode ser expandido de diversas maneiras. Pode ser adaptado para outros eventos, como, por exemplo, disputas de outros esportes ou até mesmo um estudo que compreenda todos eles.

Pode ainda, utilizar uma maior interação com sistemas já existentes de clientes, por exemplo, imprensa e comissão técnica. Estudar mais usuários

em potencial do sistema, como agências de turismo dispostas a dar mais facilidade a estrangeiros que estejam visitando o evento ou até mesmo órgãos públicos interessados em saber como melhorar a organização e acesso as informações dos eventos.

Referências Bibliográficas

ARAUJO, R. B. Computação ubíqua: Princípios, tecnologias e desafios. *XXI Simpósio Brasileiro de Redes de Computadores,*, 2003.

ERL, T. *Service-oriented architecture: Concepts, technology, and design*. Prentice Hall, 2005.

JOSUTTIS, N. M. *Soa in practice - the art of distributed systems design*. O'Reilly, 2007.

NEWMARCH, J. *A programmer's guide to jini technology*. APress, 2001.

OAKS, S. *Jini: In a nutshell*. O'Reilly, 2000.

OASIS Modelo de referência para arquitetura orientada a serviço 1.0. 2006.

WEISER, M. The computer for the 21st century. *Scientific American, vol.265*, 1991.

WIKIPEDIA Copa do mundo fifa de 2014. 2011a. Disponível em http://pt.wikipedia.org/wiki/Copa_do_Mundo_FIFA_de_2014

WIKIPEDIA Jogos olímpicos de verão de 2016. 2011b. Disponível em http://pt.wikipedia.org/wiki/Jogos_Ol%C3%ADmpicos_de_Ver%C3%A3o_de_2016