UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

LAR – Laboratório de Automação Residencial para análise comparativa entre Jini e UPnP

por

ANDRÉ ANTONIO PARMEGGIANI

Dissertação submetida à avaliação,
como requisito parcial para a obtenção do grau de
Mestre em Ciência da Computação

Prof. João César Netto
Orientador

Porto Alegre, junho de 2003.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Parmeggiani, André Antonio

LAR – Laboratório de Automação Residencial para análise comparativa entre Jini e UPnP / por André Antonio Parmeggiani. - Porto Alegre: PPGC da UFRGS, 2003.

85 f. : il.

Dissertação (mestrado) - Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2003. Orientador: Netto, João Cesar

1. Automação Residencial. 2. Domótica. 3. Jini. 4. UPnP. 5. Middleware. I. Netto, João Cesar. II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitora: Profª. Wrana Maria Panizzi
Pró-Reitor de Ensino: Prof. José Carlos Ferraz Hennemann
Pró-Reitor Adjunto de Pós-Graduação: Profª. Jocélia Grazia
Diretor do Instituto de Informática: Prof. Philippe Olivier Alexandre Navaux
Coordenador do PPGC: Prof. Carlos Alberto Heuser
Bibliotecária-chefe do Instituto de Informática: Beatriz Regina Bastos Haro

" I want to sell millions of toys, but what I really hope is that a bunch of kids who open them up use the motors and things to build something else ... They are my colleagues of the future."

Mark Tilden

Agradecimentos

A minha família, pelo amor, carinho e apoio incondicional em todos os momentos da minha vida.

Ao meu orientador, professor João Netto, pelas dicas, conselhos e paciência, sem as quais este trabalho não teria chegado ao fim, e ao professor Juergen Rochol, juntamente com os colegas da FAD, pelo convívio, amizade e presteza tanto dentro quanto fora da sala de aula.

Aos professores e colegas, por fazerem do Instituto de Informática um lugar saudável, dinâmico e comprometido com a pesquisa, e aos funcionários pela dedicação em todos os momentos.

À Fernanda, Letícia e Merissa, não só pela amizade, amor e carinho nestes últimos anos, mas principalmente por terem compartilhado comigo um tempo precioso de suas vidas.

Por último, mas não menos importante, aos amigos de trabalho (Howto Networking, ELO Sistemas Eletrônicos, Grátis1 do Brasil e RBS Direct) e da Salsa, tanto aqui quanto no Rio e em São Paulo, pelo trabalho, oportunidades, lazer, diversão, *indiadas*, *lerês*, e principalmente pelo equilíbrio que os verdadeiros amigos trazem às nossas vidas: nos colocarem para cima quando estamos muito para baixo e vice-versa.

Sumário

Lista de Abreviaturas

ACID	Atomicity Consistency Isolation Durability
AI	Artificial Intelligence
API	Application Program Intrerface
ARP	Address Resolution Protocol
CEBus	Consumer Electronics Bus
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
FXPP	Flexible XML Processing Profile
GENA	General Event Notification Architecture
HAVi	Home Audio Video interoperability
HCI	Human-Computer Interaction
HTTP	Hyper Text Transfer Protocol
HTTPMU	HTTP Multicast over UDP
HTTPU	HTTP (unicast) over UDP
IDE	Integrated Development Environment
IEEE	Institute of Electrical & Electronics Engineers
IETF	Internet Engineering Task Force
IGMP	Internet Group Management Protocol
IP	Internet Protocol
IrDA	Infrared Data Association
JCP	Jini Core Platform
JSK	Jini Starter Kit
JVM	Java Virtual Machine
JXP	Jini Extended Platform
PC	Personal Computer
PnP	Plug and Play
RAD	Rapid Application Development
RF	Radio Frequency
RFC	Request For Comments
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SCP	Simple Control Protocol
SCSL	Sun Community Source Licence
SDK	Software Development Kit
SMB	Server Message Block
SOAP	Simple Object Access Protocol
SSDP	Simple Service Discovery Protocol
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TTL	Time To Leave
UDP	User Datagram Protocol
UI	User Interface

UIC	UPnP Implementers Corporation
UML	Unified Modeling Language
UPC	Universal Product Code
UPnP	Universal Plug and Play
URI	Uniform Resource Identifier
URL	Unique Resource Locator
URN	Uniform Resource Name
USB	Universal Serial Bus
USN	Unique Service Name
UTC	Universal Time Coordinated
UUID	Universally Unique Identifier
VHS	Vertical Helix Scan
W3C	World Wide Web Consortium
XML	eXtensible Markup Language

Lista de Figuras

Lista de Tabelas

Resumo

A Automação Residencial é uma área em crescimento no mercado mundial. À medida que os avanços tecnológicos são incorporados aos dispositivos pertencentes ao ambiente residencial, e que estes se tornam mais disseminados, surgem necessidades para fabricantes e usuários. Para os fabricantes, é desejável maior capacidade de conexão e intercâmbio de dados entre os dispositivos e, conseqüentemente, padronização. Para os usuários, é desejável maior facilidade na utilização dos dispositivos e, principalmente, na configuração dos mesmos em ambientes dinâmicos, de forma a não requerer interação do usuário para a conexão e desconexão, característica das redes espontâneas. Grandes empresas têm unido esforços no desenvolvimento de novos padrões e uma das tendências observadas é o isolamento destas novas necessidades numa camada entre a aplicação e o sistema operacional, denominada *middleware*.

Este trabalho compara duas tecnologias que se enquadram nos critérios acima: Jini, da Sun Microsystems e Universal Plug and Play (UPnP), de um consórcio de empresas liderado pela Microsoft. O estudo é feito através do desenvolvimento de um protótipo, da definição de um conjunto de métricas qualitativas e quantitativas, e da implementação de um conjunto de *workloads* para a análise comparativa entre elas. Os resultados obtidos são apresentados e analisados, contribuindo com subsídios para os interessados em conhecer melhor ou optar por uma delas. Por fim, são registradas algumas necessidades observadas que poderão servir como base para trabalhos futuros.

Palavras-chave: automação residencial, domótica, middleware, Jini, UPnP

TITLE: ”HOME AUTOMATION LABORATORY FOR COPARATIVE ANALISYS BETWEEN JINI AND UPNP.”

Abstract

Home Automation is a growing area in the world market. As technological advances are incorporated into home appliances and as these appliances become more widespread, needs for manufacturers and end users arise. For the manufacturers, it is desirable greater connectivity and data interchange among the devices and, consequently, standardization. For the end users, a more user-friendly use of the devices is desirable and chiefly an easier configuration in dynamic environments, also known as spontaneous networking. Large companies have joined efforts to develop new standards, and one of the observed trends is the isolation of these new necessities in a layer between the application and the operating system, called middleware.

This work compares two technologies that fit the above criteria: Jini by Sun Microsystems, and Universal Plug and Play (UPnP) by a consortium of companies led by Microsoft. The study is made through the development of a prototype, the definition of a set of qualitative and quantitative metrics, and the implementation of a set of workloads for the comparative analysis between them. The results are presented and analyzed, contributing with subsidies for those interested in gaining a better knowledge or opting for one of them. Finally, some observed necessities have been registered, which may provide some base for future works.

Keywords: home automation, domotics, middleware, Jini, UPnP

1 Introdução

A automação é a técnica de fazer com que um processo, sistema ou aparelho opere automaticamente através de dispositivos mecânicos, eletrônicos ou eletromecânicos que possam simular operadores humanos. A automação residencial, também conhecida como *domótica*, é a área responsável pela automação do ambiente residencial. Em um primeiro contato com o tema, através do estudo da evolução das tecnologias, observação de produtos disponíveis no mercado e também de visões futurísticas, como em [GER 99] e [NEG 96], percebe-se que o estado da arte é composto pela evolução de três áreas: comunicação de dados, inteligência artificial (AI) e interface homem máquina (HCI).

A comunicação de dados garante que os dados gerados ou utilizados pelos dispositivos possam fluir através de camadas isoladas e bem definidas [TAN 94]. A inteligência, aqui definida como capacidade de aprender, permite aos dispositivos alterarem o seu comportamento sem a intervenção humana, através da percepção do estado do ambiente onde estão inseridos e do comportamento do usuário [MOZ 98]. A interface é a responsável pela qualidade da interação entre o usuário e os dispositivos [NOR 98]. Em geral é um dos itens de maior impacto para que uma tecnologia seja absorvida ou não pelos usuários, como pode ser constatado com a Internet antes e depois do advento da Web.

Investigando as características do contexto residencial, percebe-se a necessidade de acesso a dados e serviços distribuídos, impulsionada principalmente por dispositivos *wireless,* e a necessidade de auto-configuração, pois é desejável que o usuário final interaja o mínimo possível na configuração de equipamentos [GUP 2002], [NOR 2002]. Estas necessidade podem ser supridas com a ajuda de uma camada entre o sistema operacional e a aplicação, denominada *middleware* [BER 96], responsável, neste contexto, por facilitar a troca de informações e o processo de conexão entre dispositivos de uma rede residencial.

A concepção de novos equipamentos com estas características faz surgir uma nova classe de produtos. Como exemplo, alguns cenários são vislumbrados:

1. Uma impressora, ao ser conectada a uma rede, poderia anunciar a sua presença e esperar que clientes a descobrissem e utilizassem os seus serviços, sem a necessidade de configuração. Uma máquina fotográfica, conectada à mesma rede, poderia descobrir e utilizar o serviço de impressão. Novas capacidades poderiam ser adicionadas à

impressora sem prejuízo aos serviços já existentes nem a re-configuração dos clientes.

2. Um relógio, dentro de uma residência, poderia fornecer informações sobre data e hora para outros dispositivos ao invés de cada um deles ter a sua própria fonte destas informações. Qualquer alteração nas suas informações refletir-se-ia automaticamente em todos os dispositivos que as utilizam. Este componente, por ser único, poderia ser de melhor qualidade que os atualmente empregados em diversos dispositivos, tais como televisores, videocassetes e rádio-relógios.

3. A montagem e configuração de um *home theater* poderia ser tão simples quanto à aproximação física dos seus componentes. Tais dispositivos reconheceriam a existência um dos outros e ajustar-se-iam para a situação. O ato de escolher um filme a ser assistido poderia disparar um evento que colocaria a sala em modo apropriado, acionando os módulos de som, visualização e iluminação de forma adequada.

Além dos avanços tecnológicos, para que estes cenários tornem-se realidade de uma forma estruturada é necessário que haja uma padronização, possibilitando a estes novos produtos a coexistência sem a necessidade de novas estruturas ou modificações para cada novo aparelho adicionado ao conjunto.

1.1 Motivação

A padronização de ambientes residenciais automatizados é a motivação para este trabalho. Como exemplificado em [NOR 2002], para que bons produtos desenvolvidos separadamente possam coexistir, sem que o usuário enfrente o caos ao tentar utilizá-los em conjunto, são necessários padrões de conectividade e utilização. Os padrões USB e IEEE-1349 (*FireWire*) são citados como iniciativas bem sucedidas de padronização para periféricos de computadores pessoais.

A padronização é de vital importância para a evolução da automação residencial, pois fornece uma das condições necessária para o seu crescimento em escala. Grandes empresas, cientes destas necessidades e com vistas no enorme mercado em potencial, já mobilizam recursos para acelerar este processo. O resultado deste esforço pode ser observado pela criação de consórcios responsáveis pelo desenvolvimento, padronização e divulgação de novas tecnologias aplicáveis na automação residencial, como as tecnologias alvo deste trabalho.

1.2 Objetivo

O foco do trabalho está no estudo comparativo entre as tecnologias **Jini** [SUN 99], da Sun Microsystems, e **Universal Plug and Play** – **UPnP** [MIC 2000c], de um consórcio formado por várias empresas lideradas pela Microsoft. Os principais atrativos destas novas tecnologias são as suas características de *middleware* e a utilização do TCP/IP como meio de transporte, herdando toda a infra-estrutura de comunicação atualmente disponível. Apesar de bastante promissoras, estas tecnologias ainda são recentes e restritas a protótipos e alguns produtos. Além disso, vistas como tecnologias para automação residencial, elas são concorrentes diretas, assim como foram os padrões VHS e Betamax, para os videocassetes, na década de oitenta.

Como objetivo principal, este trabalho pretende fazer uma análise comparativa, baseada na especificação de um conjunto de métricas e *workloads*, entre as tecnologias Jini e UPnP, contribuindo com subsídios para os interessados em conhecer melhor ou optar por uma delas.

Este trabalho está organizado da seguinte forma: Os capítulos 2 e 3 têm por objetivo o detalhamento das duas tecnologias escolhidas, mostrando suas arquiteturas e funcionamento. O capítulo 4 contempla a especificação do protótipo. O capítulo 5 define as métricas e *workloads* a serem utilizadas na avaliação. O capítulo 6 é dedicado à execução e análise dos resultados e o capítulo 7 às considerações finais e trabalhos futuros.

2 Jini

Este capítulo é baseado na documentação original de Jini [SUN 99], [SUN 2001a], [SUN 2001b], contém elementos encontrados em outras bibliografias, principalmente [NEW 2000] e [EDW 99].

Jini é uma tecnologia para sistemas distribuídos. Ela surgiu da necessidade de se estender os benefícios de Java, uma linguagem de programação orientada a objetos e confiável na rede, a sistemas distribuídos.

Jini foi iniciada e desenvolvida nos laboratórios da Sun Microsystems por um time pequeno, responsável pela criação da tecnologia Java RMI (*Remote Method Invocation*), um dos mais úteis e interessantes componentes de Java, e da tecnologia *Java Spaces*, que mais tarde tornar-se-ia o modelo de transações de Jini. O projeto Jini iniciou por volta de 1997, mas até 98 poucas pessoas sabiam da existência do mesmo, que foi tornado público em janeiro de 99. Em março de 99 foi iniciada a Jini Community, um espaço colaborativo para a troca de informações entre os membros desta comunidade. A comunidade é abrigada no Jini.org, um site que mantém documentação, projetos, discussões e votações envolvendo a tecnologia.

Jini é licenciada através da SCSL (*Sun Community Source Licence*), onde qualquer pessoa pode ter acesso ao código fonte e participar da sua evolução, de forma semelhante ao movimento *Open Source*. Para fins comerciais, existe uma licença comercial e empresas de diversos segmentos já licenciaram a tecnologia; entre elas: Quantum, Seagate, Nokia, Ericsson, Xerox, Canon, Epson, Hewlett-Packard, Kodak, Cisco, Sony, Sharp, Philips e Toshiba.

2.1 Características

2.1.1 Simplicidade

Jini é baseada na idéia de grupos compostos por usuários e recursos. O objetivo final é fazer da rede um ambiente flexível e facilmente administrável, no qual clientes humanos e computacionais possam facilmente, através dos protocolos Jini, encontrar e utilizar recursos tanto de software quanto de hardware.

2.1.2 Escalabilidade

Por uma questão de controle e performance, sistemas Jini são divididos em grupos, também conhecidos como comunidades ou federações. Cada

federação corresponde ao tamanho de um grupo de trabalho e deve suportar a quantidade de dispositivos necessários para atender entre 10 e 100 pessoas.

2.1.3 Administração e configuração

Jini utiliza o conceito de *network plug and play*, onde dispositivos autônomos podem perceber a presença uns dos outros e cooperar quando necessário. Para facilitar este trabalho, um sistema Jini contém um conjunto de serviços que mantêm informações dinâmicas sobre os dispositivos disponíveis. Estes serviços são a chave para o funcionamento do Jini. Para entrar em uma federação, por exemplo, um dispositivo necessita um ou mais serviços de localização, que podem ser pré-conhecidos ou descobertos dinamicamente.

2.2 Componentes

Um sistema Jini é composto por três componentes básicos: serviço, cliente e serviço de localização (*lookup service*). Os objetos são movidos entre estes três componentes, através de serialização, ou invocados remotamente através de RMI, embora a especificação não exija que o RMI seja utilizado.

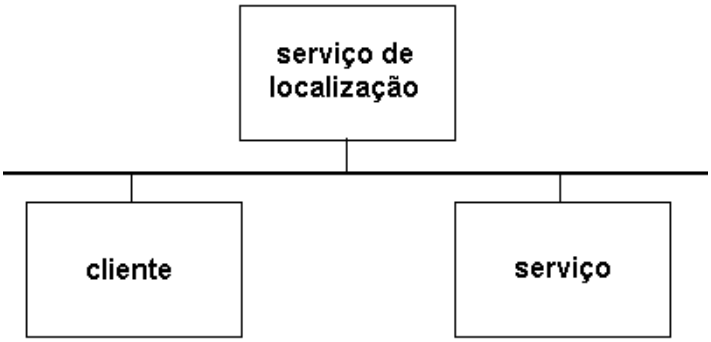

FIGURA 2.1 - Componentes básicos da arquitetura Jini.

2.2.1 Serviço

Entidade com a capacidade de descobrir e registrar-se em um serviço de localização com a finalidade de publicar os seus métodos. Pode ser utilizado por clientes ou por outros serviços e pode constituir-se de uma computação, armazenamento ou um dispositivo específico. A impressão de um documento, a conversão de um arquivo entre dois formatos e uma cópia de segurança (*backup*) são exemplos de serviços.

2.2.2 Cliente

Entidade com a capacidade de procurar, recuperar e utilizar serviços disponíveis em um serviço de localização. Não é necessária a capacidade de registro, visto que clientes puros não necessitam de registro junto aos servidores de localização. Em geral, os clientes são outros serviços ou alguma aplicação responsável pela interface com o usuário.

2.2.3 Serviço de Localização (*Lookup Service*)

É a entidade responsável por ligar clientes a serviços. É um serviço obrigatório, pois sem ele os clientes são incapazes de localizar os serviços. Vários serviços de procura podem coexistir numa rede, com propósito de redundância e para o atendimento a grupos distintos. A especificação do serviço de procura é pública e a Sun oferece uma implementação denominada *Reggie*.

Reggie: é o serviço de procura fornecido pela Sun como parte da distribuição padrão Jini. Para o seu funcionamento, o reggie necessita de um servidor HTTP e um *daemon* RMI (RMId).

O pacote Jini oferece um servidor HTTP mínimo[1] para a transferência de classes e também o RMId, que devem ser executados no mesmo dispositivo. Os comandos abaixo ilustram a inicialização dos três componentes :

```
# HTTPd
java -jar tools.jar  -port 8081 -dir \jini\lib -verbose
# RMId
rmid -log \jini\log\rmid
# Reggie
java -jar reggie.jar
```

2.3 Infraestrutrura de rede

A infraestrutura de rede é essencial para que os componentes de Jini possam se comunicar. A especificação não adota nenhum padrão, mas a única implementação existente é sobre redes TCP/IP. A linguagem Java é sugerida por ter algum suporte à programação distribuída, como, por exemplo, a mobilidade de código e dados.

Devido à grande diferença entre as necessidades de banda passante por parte dos dispositivos, são assumidas, em [SUN 99], "*velocidade e latência*

[1] Somente para a transferência de objetos. Pode-se, entretanto, utilizar qualquer outro servidor de HTTP.

razoáveis". Como exemplo da diferença entre bandas, são citadas uma impressora e uma TV de alta definição.

2.4 Funcionamento

2.4.1 Descoberta do serviço de localização

Sendo o serviço de procura um dos componentes básicos da tecnologia Jini, é necessário que tanto clientes quanto serviços possam encontrá-lo; os serviços para efetuarem o seu registro e os clientes para que possam encontrar os serviços. A descoberta do serviço de localização pode ser feita utilizando-se protocolos sobre UDP *unicast* ou *broadcast* na porta padrão 4160.

Descoberta Unicast: É um mecanismo síncrono, que pode ser utilizado quando se conhece o endereço da máquina onde o serviço de localização está sendo executado. A classe `LookupLocator` é utilizada para a descoberta *unicast*.

```
LookupLocator(java.lang.String url)
     throws java.net.MalformedURLException;
LookupLocator(java.lang.String host,int port);
```

A pesquisa é feita pelo método `getRegistrar()` que retorna um objeto da classe `ServiceRegistrar` através da pesquisa na URL fornecida ao `LookupLocator`. Este objeto age como um *proxy* para o serviço de localização, e é utilizado pelos clientes para localizarem serviços e pelos serviços para registrarem-se.

FIGURA 2.2 - Descoberta unicast.

Descoberta Broadcast: É um mecanismo assíncrono que necessita de um tratamento especial para receber as respostas, visto que podem existir mais de um serviço de localização na rede. É utilizado quando a localização do serviço é desconhecida, sendo necessário fazer um *broadcast* para procurar algum. A implementação de Jini utiliza o mecanismo de *multicast* do UDP. Os anúncios são feitos pelos serviços de localização no endereço `224.0.1.84:4160`, através do *multicast announcement protocol* e os pedidos são feitos pelos clientes ou serviços no endereço `224.0.1.85:4160` através *do multicast request protocol*.

O escopo dos pacotes *multicast* pode ser controlado de duas maneiras: a) pelo *time to live* (TTL) dos pacotes ou b) pela configuração dos *gateways* da rede. O TTL padrão da implementação é 15, mas pode ser modificado através do parâmetro `net.jini.discovery.ttl`. Em geral, pacotes *multicast* são restritos a redes locais devido ao seu custo na rede.

A classe `LookupDiscovery(String[] groups)` é utilizada para a descoberta broadcast, e são aceitos três tipos de parâmetros:

- `null` ou `LookupDiscovery.ALL_GROUPS` – procura por serviços em qualquer grupo
- *lista vazia* ou `LookupDiscovery.NO_GROUPS` - retorna serviços sem grupo, sendo necessário utilizar o método `setGroups()` posteriormente.
- *lista* - localiza todos os serviços num conjunto definido de grupos.

Para ser notificado das respostas, a aplicação deve registrar um ouvinte (*listener)* junto ao objeto `LookupDiscovery`, através do método `addDiscoveryListener()`. Este ouvinte precisa implementar a interface `DiscoveryListener`.

```
public abstract interface DiscoveryListener {
    public void discovered(DiscoveryEvent e);
    public void discarded(DiscoveryEvent e);
}
```

Após a chamada, o método `discovered()` é invocado quando um serviço de localização é descoberto. O método `discarded()` é invocado quando um serviço de localização descoberto não é mais válido, geralmente quando um servidor para de responder. O parâmetro para o método `discovered()` da interface `DiscoveryListener` é um objeto `DiscoveryEvent`.

```
public Class DiscoveryEvent {
    public net.jini.core.lookup.ServiceRegistrar[]
getRegistrars();
}
```

Este objeto tem um método público, `getRegistrars()`, que retorna uma matriz de objetos `ServiceRegistrar`.

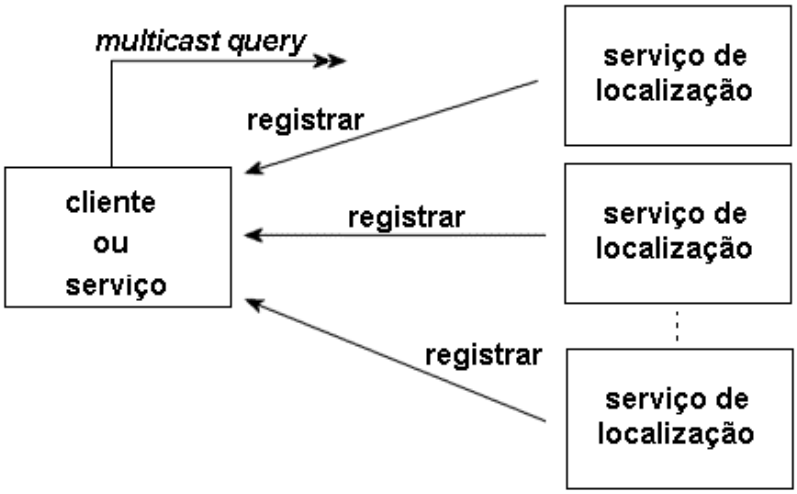

FIGURA 2.3 - Descoberta broadcast.

2.4.2 Descrição dos Serviços

Um cliente pode pesquisar um serviço de localização de duas maneiras: por um objeto específico ou por um objeto com determinadas características. A classe `Entry` ajuda na segunda situação, provendo informações sobre as classes sem alterar as suas interfaces. Objetos `Entry` são utilizados para a passagem de informações adicionais sobre os serviços para que os clientes possam decidir se um serviço em particular é o que eles necessitam. Este mecanismo é simples e só permite comparação por igualdade. Para cada `Entry` o cliente pode especificar quais campos devem possuir valores idênticos (valor diferente de nulo) e quais não devem ser considerados (nulo).

O exemplo a seguir ilustra a utilização da classe `Entry` para procurar serviços que tratem arquivos de determinado tipo:

```
public Class FileType implements Entry {
     public String type;

     public FileType(String type) {
          this.type = type;
     }
}
```

Para procurar um editor para arquivo texto:

```
Entry[] entries = new Entry[] {new FileType("text")};
```

Para procurar um editor para qualquer tipo de arquivo:

```
Entry[] entries = new Entry[] {new FileType(null)};
```

A implementação da Sun provê um conjunto de classes de conveniência, derivadas da classe `AbstractEntry`, embora estas classes não façam parte da especificação padrão. Este conjunto de classes é composto pela tabela a seguir:

TABELA 2.1 - Classes de conveniência, atributos de serviços.

Classe	Descrição
Address	endereço de um componente físico do serviço.
Comment	comentário sobre o serviço.
Location	endereço físico do local (dentro da organização)
Name	nome de um serviço utilizado pelo usuário. (um serviço pode ter múltiplos nomes).
ServiceInfo	informação genérica sobre o serviço, incluindo o nome do fabricante, produto e fornecedor.
ServiceType	informação sobre o tipo de serviço.
Status	estado do serviço.

A seguir é detalhada a classe address:

```
Address extends AbstractEntry {
     String country;
     String locality;
     String organization;
     String organizationalUnit;
     String postalCode;
     String stateOrProvince;
     String street;
}
```

2.4.3 Registro do serviço

O registro de um serviço é feito por um **provedor de serviço**, o qual tem as seguintes responsabilidades:

- criar os objetos que implementam o serviço.
- registrar um destes objetos, denominado **objeto do serviço** (*service object*) em um serviço de localização.
- manter o serviço ativo.
- manter a persistência do estado do serviço, em especial do seu identificador (`ServiceID`).

O provedor do serviço cria um objeto da classe `ServiceItem`, que é utilizado no registro do serviço.

```
public Class ServiceItem {
    public ServiceID serviceID;
    public java.lang.Object service;
    public Entry[] attributeSets;

    public ServiceItem(ServiceID serviceID,
                   java.lang.Object service,
                   Entry[] attrSets);
}
```

O primeiro parâmetro, `serviceID`, é um identificador para o serviço. Ele deve ser nulo quando o serviço é registrado pela primeira vez e o serviço de localização retornará um valor não nulo, que deverá ser persistido e utilizado em novas requisições. O ServiceID é um número de 128 bits, gerado exclusivamente pelo serviço de localização de forma pseudo-randômica, de modo a ser aceito como único devido à sua baixa probabilidade[2] de repetição [MEA 2002].

O segundo parâmetro, `service`, é o objeto que está sendo registrado. Ele é serializado e enviado ao serviço de localização para armazenamento. Quando um cliente requisitar o serviço, este objeto é retornado. O terceiro parâmetro, `attrSets`, é um vetor de atributos, com o objetivo de descrever o serviço, conforme 2.4.2.

Após localizar o serviço de localização, através da descoberta *unicast* ou *broadcast*, o provedor do serviço recebe um objeto `ServiceRegistrar` que age como um *proxy* para o serviço de localização. Este *proxy* executa na aplicação, que pode ser um serviço ou cliente e ele não deve armazenar informação no aplicativo, mas obtê-la diretamente no serviço de localização, caso necessário. O registro do serviço é feito utilizando-se o método `register()` do objeto `ServiceRegistrar`. Este passo envolve a cópia do objeto do serviço e o seu armazenamento no serviço de localização.

```
public Class ServiceRegistrar {
     public ServiceRegistration register(ServiceItem
     item,
     long leaseDuration)
     throws java.rmi.RemoteException;
}
```

[2] Visto que alguns bits são utilizados para outras funções, a probabilidade de geração de um ID repetido é dada como 2^{-120}.

O parâmetro `item` é o objeto criado no passo anterior e o `leaseDuration`, expresso em mili-segundos, refere-se ao tempo em que o serviço permanecerá registrado no serviço de localização. Ele pode ser rejeitado ou diminuído, conforme 2.4.5. O registro retorna um `ServiceRegistration`, que pode ser utilizado para recuperar o `ServiceID` e o `leaseDuration`.

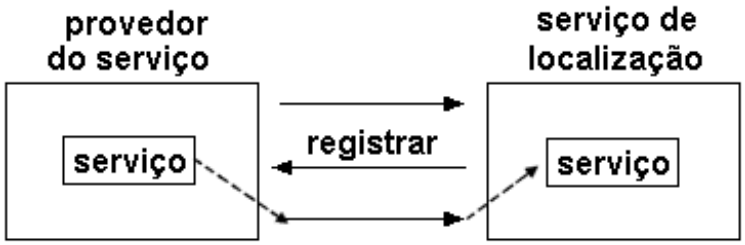

FIGURA 2.4 - Registro de um serviço.

O objeto do serviço, a ser registrado junto ao serviço de localização, é criado na JVM do provedor e transferido para o serviço de localização através do método `register()`. Ele pode ser de três tipos:

- *proxy* de serviço[3]: implementa todo o serviço e é transferido e executado integralmente na JVM do cliente no momento da execução.
- *proxy* RMI: é executado no provedor de serviço e utiliza-se do RMI para que o cliente consiga invocar os métodos do serviço e receber os resultados de forma transparente.
- *proxy* não RMI: é implementado parte no cliente, parte no provedor de serviço. A comunicação entre eles não é especificada e pode ser feita através de *sockets* e protocolos proprietários ou utilizando-se algum outro protocolo conhecido.

Visto que o objeto do serviço vai ser transmitido na rede, é recomendada atenção na concepção e implementação da classe em termos de tamanho, pois, teoricamente, não se sabe quem vai utilizá-lo nem as condições da rede.

2.4.4 Pesquisa do Serviço

Quando um cliente necessita de um serviço, ele precisa encontrar um ou mais serviços de localização e depois pesquisar pelo serviço desejado em cada um deles. A descoberta do serviço de localização retorna ao cliente um objeto

[3] O termo proxy de serviço (service proxy) é utilizado desta forma na literatura, apesar do objeto ser transportado e executado integralmente para a JVM do cliente.

`ServiceRegistrar`, o qual é utilizado para pesquisar por serviços através do método `lookup()`.

```
public Class ServiceRegistrar {
  public java.lang.Object lookup(ServiceTemplate tmpl)
    throws java.rmi.RemoteException;
  public ServiceMatches lookup(ServiceTemplate tmpl,
                               int maxMatches)
    throws java.rmi.RemoteException;
}
```

O parâmetro `tmpl` é uma classe que modela o serviço desejado e `maxMatches` é utilizado para limitar o número de respostas da consulta.

```
public Class ServiceTemplate {
    public ServiceID serviceID;
    public java.lang.Class[] serviceTypes;
    public Entry[] attributeSetTemplates;

    ServiceTemplate(ServiceID serviceID,
                    java.lang.Class[] serviceTypes,
                    Entry[] attrSetTemplates);
}
```

Quando o cliente não conhece o `serviceID` do serviço desejado, o mesmo deve ser deixado nulo. O atributo `attributesSetTemplates` é um vetor de atributos da classe `Entry`, utilizado para filtrar o serviço desejado. Para que um serviço seja retornado é necessário que todos os atributos sejam satisfeitos pela condição de igualdade. O parâmetro mais importante para a pesquisa é o vetor `serviceTypes`. Ele é composto por interfaces que devem ser implementadas pelos serviços desejados.

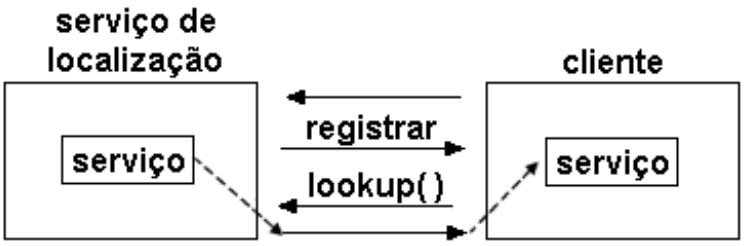

FIGURA 2.5 - Pesquisa e recuperação de um serviço.

Se o cliente procura por mais de um objeto, utilizando o parâmetro `maxMatches`, o resultado da consulta é um objeto `ServiceMatches`

```
public Class ServiceMatches {
    public ServiceItem[] items;
    public int totalMatches ;
}
```

Caso a consulta retorne um número de ocorrências maior que `maxMatches`, o resultado é truncado e somente as primeiras `maxMatches` ocorrências são retornadas . Em caso contrário, todos os itens são retornados, e `totalMatches` é ajustado para o número de ocorrências encontradas.

Um serviço presente no serviço de localização é considerado resultado de uma consulta, ou seja, um item, se todas as condições abaixo forem verdadeiras:

- `item.serviceID = tmpl.serviceID` ou `tmpl.serviceID` é nulo
- `item.service` é uma instância de cada um dos tipos contidos em `tmpl.serviceTypes`
- `item.attributeSets` contém pelo menos um resultado positivo para cada atributo em `tmpl.attributeSetTemplates`.

Um atributo valida um template se a classe do template é a mesma, ou uma superclasse, da classe do entry e cada campo não nulo no template é igual ao campo correspondente no atributo. Cada atributo pode ser utilizado para mais de uma comparação no template. É importante observar que para `serviceTypes` e `attributeSetTemplates`, um campo nulo é equivalente a um *array* vazio, ambos representam um caractere curinga.

2.4.5 Contratos, Eventos e Transações

Contratos:

Contratos são uma maneira que os serviços têm de sinalizarem que eles estão ativos. Quando um provedor de serviço registra um serviço no serviço de localização, ele deve informar por quanto tempo o objeto ficará disponível no mesmo. Este tempo de validade é dado em mili-segundos, e também pode receber os seguintes valores especiais:

- `Lease.ANY`: delega ao serviço de localização a escolha do tempo de expiração do contrato.
- `Lease.FOREVER`: para contratos sem tempo de expiração.

O contrato é retornado ao serviço e disponível através do método `ServiceRegistration.getLease()`. O serviço de localização age como o outorgante do contrato e suas funções são:

- Decidir pela aceitação, ou não, do tempo requerido no contrato;
- Decidir pela aceitação, ou não, dos pedidos de renovação;
- Fornecer o tempo do contrato, quando utilizado o valor `Lease.ANY`
- Garantir o cumprimento do contrato.

```
ServiceRegistration reg = registrar.register();
Lease lease = reg.getLease();
```

```
public interface Lease {
    void cancel() throws
                  UnknownLeaseException,
                  java.rmi.RemoteException;
    long getExpiration();
    void renew(long duration) throws
                  LeaseDeniedException,
                  UnknownLeaseException,
                  java.rmi.RemoteException;
}
```

Um serviço pode cancelar o contrato através do método `cancel()`, que acionará o serviço de localização para que descarte o armazenamento do serviço. Uma boa prática para um serviço desconectar-se da rede é invocar este método antes de terminar. A responsabilidade da renovação do contrato é do provedor do serviço. A renovação deve ser feita através do método `renew()` antes que o contrato expire.

O pacote *LandLord* é um pacote que permite contratos mais complexos. Ele não faz parte da especificação de Jini, mas é fornecido com o kit de desenvolvimento.

Eventos:

Jini utiliza eventos para fazer notificações assíncronas de mudanças de estado, assim como o Java. A diferença é que o modelo de Java foi desenvolvido para tratar eventos dentro da mesma JVM enquanto Jini trata eventos de forma distribuída.

Se um cliente quiser ser informado sobre as mudanças de um determinado serviço ele deve registrar-se como um ouvinte (*listener*) deste serviço. Isto é feito através do método `notify()` herdado da classe `Object`.

```
Public class RemoteEvent extends java.util.EventObject{
     RemoteEvent(Object source, long eventID,
     long seqNum, java.rmi.MarshalledObject handback)
```

```
        long getID()
        java.rmi.MarshalledObject getRegistrationObject()
        long getSequenceNumber()
}
```

Os problemas inerentes a este tipo de evento distribuído, tais como ordenação das mensagens e falhas parciais, não são endereçados por Jini, ficando a cargo do desenvolvedor.

Transações:

Transações são necessárias para coordenarem mudanças de estado entre múltiplos clientes e serviços. O modelo de transações de Jini obedece segue as propriedades ACID [GRA 81]:

- Atomicidade: todas as operações de uma transação são realizadas ou nenhuma delas é realizada.
- Consistência: O término de uma transação deve deixar os participantes em um estado consistente.
- Isolação: As atividade de uma transação não devem afetar outras transações.
- Durabilidade: Os resultados de uma transação devem ser persistentes.

É utilizado um modelo de duas fases (*two-phase commit protocol*) [EDW 99], no qual os participantes são solicitados a votar em uma determinada transação. Se todos concordarem, a transação é efetuada. Se um dos participantes desistir durante a votação, então a transação é abortada para todos os participantes, garantindo a propriedade de atomicidade. A disponibilidade de todos os envolvidos no processo é gerenciada através de contratos.

Quando um gerenciador de transações é solicitado para uma nova transação, ele retorna um objeto do tipo `TransactionManager.Created`, contendo o identificador da transação e o contrato

```
public interface TransactionManager {
    public static class Created {
        public final long id;
        public final Lease lease;
    }
    ...
}
```

```
public interface TransactionManager {
```

```
        Created create(long leaseFor) throws ...;
        void join(long id, TransactionParticipant part,
                  long crashCount) throws ...;
        void commit(long id) throws ...;
        void abort(long id) throws ...;
        ...
    }
```

O gerenciador de transações inicia o *two-phase* através do método `create()` e procura por todos os participantes da transação. Quando um participante junta-se à transação, ele registra um tratador do tipo `TransactionParticipant`. Se qualquer um dos participantes chamar `commit()`, o gerenciador de transação inicia o processo de votação invocando todos os participantes com `prepare()`. Se todos estiverem prontos para o *commit*, o gerenciador move todos os participantes para o estágio de *commit*. Se qualquer um dos participantes invocar `abort()`, o processo é abortado para todos.

O *Mahalo* é o gerenciador de transações da distribuição Jini. Ele é executado como um serviço Jini, assim como o serviço de localização (*Reggie)*.

3 Universal Plug and Play – UPnP

Este capítulo é baseado principalmene na documentação oficial do UPnP, [MIC 2000a], [MIC 2000c] e complementado com o resultado dos experimentos práticos.

O UPnP é uma arquitetura aberta e distribuída, cuja finalidade é estender os benefícios do conceito *Plug and Play*[4] (PnP), através do agrupamento de dispositivos, tais como computadores e eletrodomésticos, interligados em um ambiente de rede.

Surgiu no início de 1999, através de uma iniciativa da indústria liderada pela Microsoft e com a participação de outras 28 grandes empresas, incluindo Intel, Hewlett-Packard, Compaq, Kodak e Samsung, para possibilitar conectividade fácil e robusta entre dispositivos isolados e computadores de diferentes fabricantes. Em outubro de 1999 foi criado o UPnP Forum, com o objetivo de definir e publicar padrões, protocolos e modelos, permitindo a interoperabilidade entre dispositivos num ambiente de rede escalável. O fórum é aberto e as empresas são estimuladas a associarem-se mediante ao preenchimento de uma proposta de membro. Atualmente mais de 550 empresas fazem parte do fórum, organizadas em comitês por áreas específicas, encarregadas de criar padrões para dispositivos e desenvolver pequenas implementações para testes. Os comitês existentes atualmente estão divididos nas seguintes áreas:

- Automação residencial e segurança
- Áudio e vídeo
- Internet Gateway
- Imagem
- Dispositivos móveis
- Eletrodomésticos

Além do fórum, foi criada a *UPnP Implementers Corporation*[5] (UIC), uma corporação sem fins lucrativos responsável pelo processo de certificação de dispositivos UPnP. A UIC certifica dispositivos que implementam os padrões aprovados pelo fórum e também administra a utilização da marca UPnP.

[4] Refere-se à capacidade de um computador configurar automaticamente placas de expansão e outros dispositivos sem a necessidade de configuração através de *jumpers*, *DIP switches* ou outros elementos.
[5] http://www.upnp-ic.org/

3.1 Características

3.1.1 Comunicação entre dispositivos

O UPnP possibilita a comunicação entre quaisquer dois dispositivos na sua rede, independentemente do meio físico, linguagem de programação e sistema operacional. As informações básicas sobre os dispositivos são padronizadas pelo fórum, mas podem ser estendias pelos fabricantes.

3.1.2 Protocolos abertos

O modelo de protocolos abertos prega que cada fabricante deve estar em conformidade com o padrão proposto, não sendo obrigado a implementar nenhuma tecnologia de outro fabricante, e com garantia de interoperabilidade com um grande número de produtos de outros fabricantes. O UPnP utiliza-se de protocolos abertos, principalmente os utilizados na Internet, tais como o TCP/IP e o HTTP, definidos pela *Internet Engineering Task Force* (IETF), que provaram ser eficientes mesmo sendo implementados por diversos fabricantes.

3.1.3 Administração e configuração

A rede UPnP é descentralizada e pode funcionar sem a presença de computadores pois não é centrada na sua presença. Os dispositivos são tratados como nodos da rede e não como periféricos. Para dispositivos periféricos e redes não IP é necessária a utilização de *proxies*, *gateways,* ou de protocolos complementares, como o *Simple Control Protoco* (SCP) [MIC 2000b].

Tanto a administração quanto a configuração dos dispositivos é feita através dos conceitos de rede independente de configuração (*zero-config network*) e rede espontânea (*spontaneous networking*), onde um dispositivo pode entrar na rede dinamicamente, obter o seu IP, anunciar seu nome, transportar seus serviços sob demanda e aprender sobre a presença e capacidades de outros dispositivos. Os serviços de obtenção de endereço IP e de resolução de nomes, DHCP e DNS, são opcionais desde que os dispositivos consigam obter um endereço IP. A figura abaixo ilustra a detecção de um dispositivo UPnP por um computador rodando Windows XP com suporte ao UPnP habilitado:

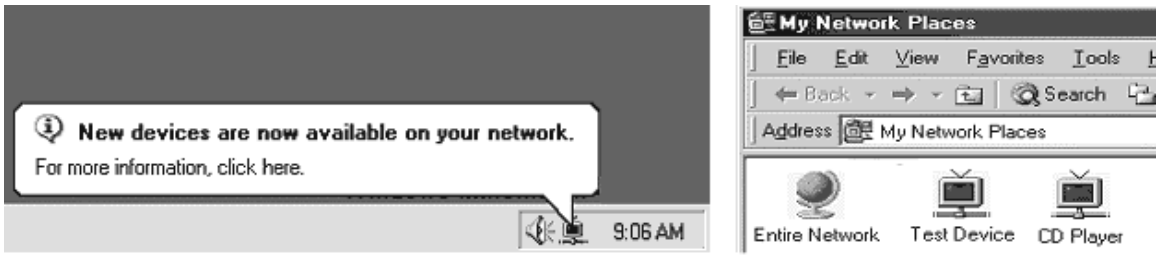

FIGURA 3.1 - Detecção de um dispositivo UPnP pelo Windows XP.

3.1.4 Desenvolvimento

Como o UPnP é baseado em protocolos abertos, ele é neutro quanto ao sistema operacional, linguagens de programação, APIs e ferramentas. A vantagem que esta característica proporciona é a maior flexibilidade para que os fabricantes e desenvolvedores possam escolher a melhor plataforma para seus dispositivos, desde que sejam atendidas as normas.

3.2 Componentes

O UPnP define três componentes básicos: dispositivos, serviços e pontos de controle.

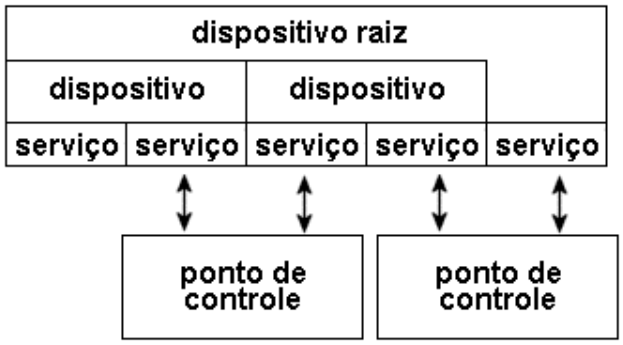

FIGURA 3.2 - Componentes do UPnP.

3.2.1 Dispositivo

É um repositório que contém serviços e/ou outros dispositivos aninhados. Um dispositivo relógio, por exemplo, pode ser composto de um serviço data_hora e um serviço de *display*. Um dispositivo rádio-relógio pode ser composto por um serviço sintonizador, data_hora e display ou um serviço sintonizador e um dispositivo relógio (aninhado). O dispositivo que contém

dispositivos aninhados, mas não está aninhado em outros dispositivos é denominado **dispositivo raiz**.

Os dispositivos são agrupados pela semelhança dos serviços. Grupos de trabalho, conforme mencionado na introdução deste capítulo, são responsáveis por padronizar os tipos de serviços que uma determinada classe de dispositivos deve prover, originando uma descrição de dispositivo. Essa descrição é escrita XML [W3C 2000], e deve ser armazenada pelos dispositivos.

3.2.2 Serviço

É a menor unidade de controle do UPnP. Ele expressa ações e armazena os seus estados através de variáveis de estado. Da mesma forma que nos dispositivos, essa descrição origina uma descrição de serviço, também escrita em XML e padronizada pelo fórum. Um ponteiro (URL) para cada descrição de serviço é armazenado na descrição do dispositivo. No exemplo do relógio, um serviço poderia ser modelado como uma variável de estado *current_time* e duas ações: *set_time* e *get_time*.

O serviço consiste em uma tabela de estados, um servidor de controle e um servidor de eventos. A tabela de estados modela o estado do serviço através de variáveis de estados e as atualiza quando o estado do dispositivo é alterado. O servidor de controle recebe requisições de controle (ações), as executa, atualiza a tabela de estados e retorna o resultado. O servidor de eventos informa eventos para os clientes interessados e inscritos em determinados eventos, notificando-os de quaisquer mudanças no serviço. No exemplo do relógio, um evento poderia ser enviado para os interessados quando o status do serviço fosse alterado para *despertar*.

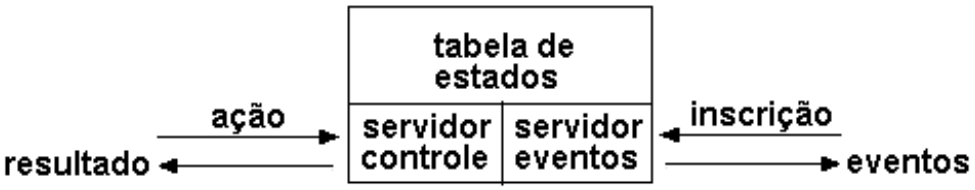

FIGURA 3.3 – Estrutura de um Serviço UPnP.

3.2.3 Ponto de Controle

É uma entidade capaz de descobrir e controlar dispositivos, com as seguintes funções:

- solicitar a descrição de um dispositivo e sua lista de serviços disponíveis.
- solicitar a descrição de dispositivos para serviços específicos.
- enviar solicitações de controle para controlar um dispositivo.
- inscrever-se no servidor de eventos de um dispositivo.

Fazendo uma analogia com a arquitetura cliente-servidor, o ponto de controle é o cliente enquanto o dispositivo é o servidor que disponibiliza os serviços. Para que haja maior interação entre os dispositivos, é permitido que dispositivos contenham pontos de controle e vice-versa.

3.3 Infraestrutura de Rede

O UPnP define a camada IP como o nível mais baixo e não especifica o meio físico. Os dispositivos componentes de uma rede UPnP podem se conectar utilizando qualquer meio, incluindo RF, wi-fi, *powerline*, IrDA, *Ethernet* e *FireWire*. A única restrição é que o meio suporte a banda necessária para a utilização pretendida. O UPnP utiliza protocolos abertos e padronizados, tais como o TCP/IP, OS e XML. Entretanto outras tecnologias podem ser utilizadas, como o HAVi, CeBus, LonWorks, e X10. Essas tecnologias poderão fazer parte de uma rede UPnP através da utilização de *proxies* ou do SCP.

3.4 Protocolos

O UPnP utiliza-se de protocolos padronizados para garantir a interoperabilidade entre as implementações de fabricantes distintos. Estes protocolos adotados são amplamente utilizados e difundidos em redes locais e na Internet. Tal escolha traz como vantagem toda a base de conhecimento e soluções baseadas nestes protocolos, além da própria infraestrutura existente.

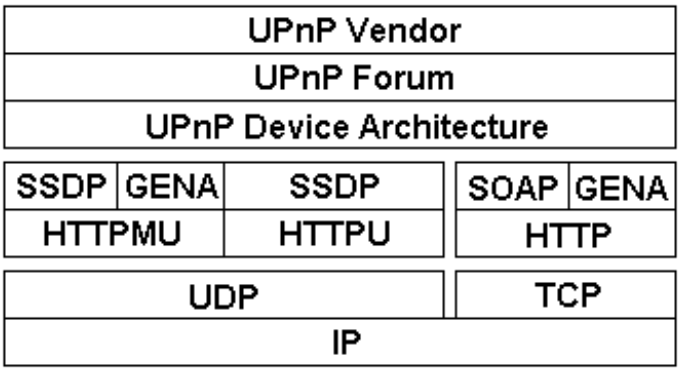

FIGURA 3.4 – Pilha de protocolos utilizada pelo UPnP.

3.4.1 TCP/IP

A camada de protocolos TCP/IP é a base sobre a qual funcionam os protocolos do UPnP. São utilizados os protocolos TCP, UDP, IGMP, ARP e o IP, além dos serviços de DHCP e DNS. Por serem protocolos bastante conhecidos, os mesmos são citados neste trabalho, e o seu entendimento básico poderá ser recuperado através das referências e RFCs.

3.4.2 HTTP, HTTPU, HTTPMU

O HTTP, também é um protocolo bastante conhecido e difundido, principalmente pela sua utilização na Internet. O HTTPU e o HTTPMU são variantes do HTTP, cujas mensagens trafegam sobre pacotes UDP/IP (*unicast* ou *multicast*) ao invés do TCP/IP.

3.4.3 Simple Service Discovery Protocol – SSDP

Este protocolo define como os serviços podem ser anunciados e descobertos na rede. O SSDP utiliza-se dos protocolos HTTPU e HTTPMU e define métodos tanto para pontos de controle localizarem dispositivos quanto para dispositivos anunciarem sua presença na rede. Esta característica contribui para um melhor conhecimento do estado da rede, visto que a informação pode ser enviada ou solicitada pelos nodos de forma independente.

Os dispositivos monitoram uma porta *multicast* (HTTPMU) e ao receberem um pedido de procura, caso se enquadrem nele, uma resposta SSDP *unicast* (HTTPU) é enviada ao ponto de controle solicitante. A procura por dispositivos pode ser refinada para um dispositivo único, específico, ou para uma determinada classe de dispositivos.

Quando um dispositivo é adicionado à rede, ele envia mensagens SSDP anunciando os serviços que ele suporta. Tanto os anúncios de presença quanto as mensagens de resposta *unicast* dos dispositivos contêm um ponteiro para um documento de descrição do dispositivo, o qual contém informações sobre as propriedades e serviços suportados pelo dispositivo. Quando um dispositivo é retirado da rede, o SSDP provê uma maneira de seus serviços deixarem a rede de uma maneira comportada e também inclui mecanismos de *timeout* para detectar e corrigir estados inconsistentes.

3.4.4 Generic Event Notification Architecture – GENA

É o protocolo responsável por enviar e receber notificações sobre eventos utilizando HTTP sobre TCP e UDP *multicast*. Ele define os conceitos de assinante e publicador de notificações para tratar da ocorrência de eventos.

Um ponto de controle interessado em receber notificações de eventos de um determinado dispositivo inscreve-se para um evento enviando uma requisição que inclui o serviço de interesse, um ponteiro para receber a notificação e o tempo de duração da inscrição. A inscrição deve ser renovada periodicamente para ter validade e também pode ser cancelada.

3.4.5 Simple Object Access Protocol – SOAP

O SOAP é utilizado para entregar mensagens de controle para dispositivos e retornar resultados ou erros para os pontos de controle. Cada requisição de controle é uma mensagem SOAP que contém uma ação a ser invocada juntamente com um conjunto de parâmetros. A resposta é uma mensagem SOAP que contém o *status* e o resultado.

O SOAP define a utilização do XML e do HTTP para executar procedimentos remotos. É um protocolo que está se tornando o padrão para comunicação baseada em RPC na Internet, pois aproveita-se dos benefícios disponíveis ao OS, como a presença de *firewalls* e conexões SSL.

3.4.6 Extensible Markup Language – XML

O XML é um padrão utilizado na descrição de dispositivos e serviços, nas mensagens de controle e eventos. É um formato universal para dados estruturados na Web, uma maneira de representar dados através de texto. Ele é muito semelhante ao HTML, pois utiliza marcadores e atributos. Os marcadores não são globalmente definidos, mas são interpretados dentro do contexto de seu uso. Essas características do XML o fazem ideal para o desenvolvimento de esquemas para vários tipos de documentos. O uso do XML como linguagem é definido pelo W3C.

3.5 Funcionamento

Uma rede UPnP deve prover suporte para a comunicação entre pontos de controle e dispositivos. O protocolo TCP/IP fornece a conectividade e o sistema de endereçamento básico necessário para a comunicação. Sobre esta camada, o UPnP define um conjunto de servidores HTTP para tratarem das funcionalidades do protocolo: descoberta, descrição, controle, eventos e apresentação.

3.5.1 Endereçamento

O endereçamento é o primeiro requisito que um dispositivo ou ponto de controle necessita para fazer parte de uma rede UPnP. Como o UPnP é baseado no TCP/IP, o endereçamento é uma questão primordial. Existem três maneiras para resolver o endereçamento: DHCP, Auto-IP e DNS. A documentação do UPnP [MIC 00] sugere que cada dispositivo tenha um cliente de DHCP e procure por um servidor quando o dispositivo é conectado à rede. Se o servidor estiver disponível o dispositivo deve utilizar o IP fornecido por ele. Caso não exista servidor DHCP, deve ser utilizado o protocolo Auto-IP, protocolo que escolhe um IP através de uma faixa reservada de endereços.

Um dispositivo ainda pode implementar protocolos de mais alto nível para a utilização de nomes, ao invés de endereços numéricos, e a sua conversão para endereços IP. Em geral o DNS é utilizado para isso. Dispositivos com essa característica necessitam de um cliente DNS e de um servidor disponível na rede. O DNS dinâmico também pode ser utilizado, permitindo que os dispositivos registrarem seus próprios nomes.

Um dispositivo que suporte o Auto-IP deve iniciar o processo de endereçamento dinâmico solicitando um endereço IP a um servidor DHCP, enviando uma mensagem `DHCPDISCOVER`. Se uma mensagem `DHCPOFFER` é recebida durante este procedimento, o dispositivo deve continuar a requisição através do DHCP. Caso não receba a mensagem[6], deve auto-configurar um endereço IP utilizando um algoritmo próprio para escolher um endereço dentro do domínio `169.254/16`, sendo que os primeiros e os últimos 256 endereços são reservados e não devem ser utilizados. Deve-se testar o endereço escolhido na rede e se o mesmo já estiver em uso, deve-se escolher e testar um novo endereço. O número de vezes permitido para este procedimento não é definido e recomenda-se um método de escolha randômico para evitar colisões quando vários dispositivos estejam tentando alocar um endereço.

Para testar o endereço escolhido, o dispositivo deve utilizar um *probe* ARP. Um *probe* ARP é uma requisição com o endereço de hardware do dispositivo utilizado como endereço de origem e o endereço IP igual a zero. Após a emissão do *probe*, o dispositivo espera por respostas ao *probe* ou outros *probes* para o mesmo IP. Se qualquer destes pacotes ARP forem recebidos, o dispositivo deve considerar o endereço em uso e tentar um novo endereço.

Um dispositivo cujo endereço foi obtido pelo Auto-IP deve verificar periodicamente pela existência de um servidor DHCP, através do envio de mensagens[7] `DHCPDISCOVER`. Se um DHCP for encontrado, um novo IP deve ser utilizado no lugar do antigo, que pode ser mantido por algum tempo para

[6] O tempo de resposta de DHCPOFFER pode variar de acordo com a implementação.
[7] O intervalo não é estabelecido, mas é sugerido o valor de 5 minutos para manter um balanço entre a banda e a conectividade.

evitar a perda de alguma conexão. Para fazer a troca dos IPs, o dispositivo deve cancelar quaisquer anúncios e re-enviar novos após a troca.

3.5.2 Descoberta

Uma vez que os dispositivos estejam endereçados na rede, o serviço de descoberta pode entrar em funcionamento. Este serviço é suportado pelo SSDP e permite que um dispositivo anuncie seus serviços aos pontos de controle e que os mesmos procurem dispositivos de seu interesse na rede.

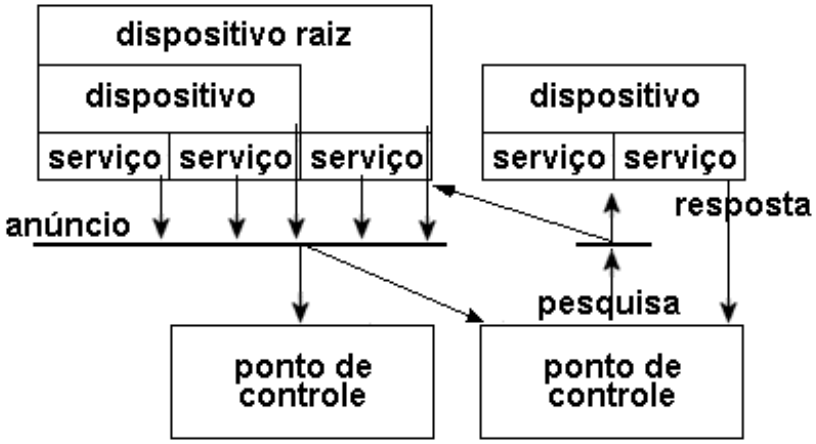

FIGURA 3.5 – Mecanismos de anúncio, pesquisa e resposta.

Os anúncios são feitos através de mensagens `NOTIFY` *multicast* `ssdp:alive` para o endereço padronizado `239.255.255.250:1900` e não necessitam de resposta. Para anunciar suas habilidades, o dispositivo raiz deve anunciar: três mensagens para o dispositivo raiz, duas mensagens para cada dispositivo aninhado e uma mensagem para cada serviço; ou seja, para um dado dispositivo raiz, com *d* dispositivos aninhados com *s* serviços e *k* serviços distintos, são enviadas ***3 + 2d + k*** mensagens para o seu completo anúncio.

Um dos parâmetros destas mensagens é o tempo de duração pelo qual o anúncio é válido, em segundos. Depois deste tempo, se o anúncio não for renovado, ele é descartado. O tempo de duração deve ser balanceado para minimizar o tráfego na rede e maximizar a atualização do status do dispositivo. Recomenda-se que sejam levadas em consideração as características de disponibilidade do dispositivo para ajustar seu tempo de duração. O formato da mensagem de anúncio é:

```
NOTIFY * HTTP/1.1
HOST: 239.255.255.250:1900
CACHE-CONTROL: eq-age = tempo de duração
```

```
LOCATION: URL de descrição do dispositivo raiz
NT: tipo de notificação
NTS: ssdp:alive
SERVER: versão_do_OS UPnP/1.0 versão_do_produto
USN: UUID
```

Quando um dispositivo e seus serviços são removidos, o dispositivo deve enviar uma mensagem `NOTIFY` *multicast* `ssdp:byebye` para cada `ssdp:alive` enviado no seu anúncio (*3+2d+k*) e que ainda não expirou. Caso não seja possível, vale o tempo de duração do anúncio. O formato da mensagem de remoção é:

```
NOTIFY * HTTP/1.1
HOST: 239.255.255.250:1900
NT: tipo de notificação
NTS: ssdp:byebye
USN: UUID
```

Quando um ponto de controle é conectado à rede ele pode procurar por dispositivos através do envio de mensagens `M-SEARCH` *multicast.* Esta procura pode ser por um dispositivo específico, ou para uma classe de dispositivos.

```
M-SEARCH * HTTP/1.1
HOST: 239.255.255.250:1900
MAN: "ssdp:discover"
MX: tempo de espera por uma resposta
ST: alvo
```

A resposta é *unicast*, semelhante à mensagem de anúncio do dispositivo:

```
HTTP/1.1 200 OK
CACHE-CONTROL: eq-age = tempo de duração
DATE: data da resposta
EXT:
LOCATION: URL for UPnP description for root device
SERVER: versão_do_OS UPnP/1.0 versão_do_produto
ST: search target
USN: UUID
```

A limitação[8] do escopo das mensagens *multicast* é dada através do *time to leave* (TTL) dos pacotes IP. Devido à não confiabilidade do UDP, as mensagens devem ser enviadas mais de uma vez pelos dispositivos.

[8] É recomendado um TTL padrão = 4.

3.5.3 Descrição

Quando um ponto de controle descobre um dispositivo, ele detém pouca informação sobre o mesmo. É necessário, então, que o ponto de controle recupere as características do dispositivo em questão, para que o mesmo possa ser utilizado. A recuperação das informações necessárias é feita através da URL provida pelo serviço de descoberta. O resultado é um documento em XML, que inclui a descrição detalhada do dispositivo, com seus dispositivos lógicos, serviços e dados específicos do fabricante. A descrição inclui uma lista de cada dispositivo ou serviço embarcado, bem como URLs de controle, eventos e apresentação.

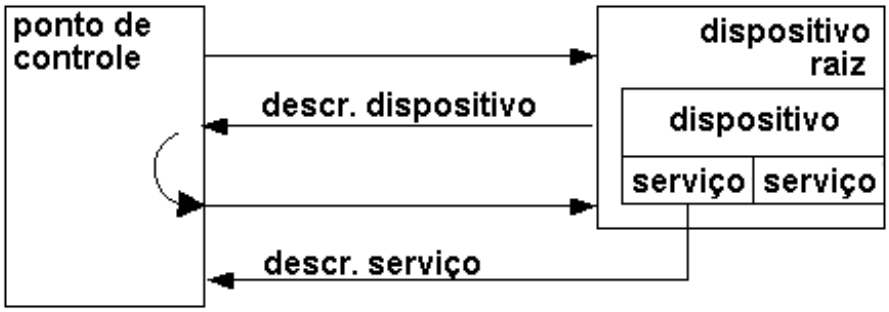

FIGURA 3.6 – Recuperação das descrições de dispositivos e serviços.

A descrição de um dispositivo UPnP é dividida em duas partes: descrição do dispositivo e descrição do serviço. Ela é escrita em XML e recomenda-se a utilização dos modelos *UPnP Device Template* e *UPnP Service Template* [MIC 2000[a]], produzidos pelo fórum UPnP.

A descrição é recuperada por um ponto de controle através de uma requisição `GET` do HTTP na URL fornecida na mensagem de descoberta. Se um dispositivo necessitar mudar sua descrição ele deve desconectar-se da rede, fazer a mudança e conectar-se novamente. Por este motivo, os pontos de controle devem considerar um dispositivo anunciado na rede como desconhecido. A seguir, o XML de um dispositivo:

```
<?xml version=”1.0”?>
<root xmlns=”urn:schemas-upnp-org:device-1-0”>
  <specVersion>
    <major>1</major>
    <minor>0</minor>
  </specVersion>
  <URLBase>URL base para URLs relativas</URLBase>
  <device>
    <deviceType>urn:schemas-upnp-
org:device:deviceType:v</deviceType>
```

```
    <friendlyName>apelido</friendlyName>
    <manufacturer>fabricante</manufacturer>
    <manufacturerURL>URL fabricante</manufacturerURL>
    <modelDescription>título</modelDescription>
    <modelName>nome do modelo</modelName>
    <modelNumber>número do modelo</modelNumber>
    <modelURL>URL do modelo</modelURL>
    <serialNumber>número serial do
fabricante</serialNumber>
    <UDN>uuid:UUID</UDN>
    <UPC>Código Universal do Produto</UPC>
    <iconList>
      <icon>
        <mimetype>imagem/formato</mimetype>
        <width>pixels horizontais</width>
        <height>pixels verticais</height>
        <depth>cores</depth>
        <url>URL do icone</url>
      </icon>
    </iconList>
    <serviceList>
      <service>
        <serviceType>urn:schemas-upnp-
org:service:serviceType:v</serviceType>
        <serviceId>urn:upnp-
org:serviceId:serviceID</serviceId>
        <SCPDURL>URL de descrição do serviço</SCPDURL>
        <controlURL>URL de controle</controlURL>
        <eventSubURL>URL de eventos</eventSubURL>
      </service>
    </serviceList>
    <deviceList>
    </deviceList>
    <presentationURL>URL apresentação</presentationURL>
  </device>
</root>
```

Como especificado pelo *Flexible XML Processing Profile* (FXPP), dispositivos e pontos de controle devem ignorar quaisquer elementos ou atributos desconhecidos. Dados binários devem ser convertidos[9] em texto para fazerem parte do XML ou passadas referências (URLs) para que os dados sejam transferidos por requisições OS. Os ícones da descrição do UPnP são transferidos por OS.

[9] São utilizadas as codificações bin.base64 (estilo MIME) ou bin.hex, onde dígitos hexadecimais representam octetos.

A descrição de um serviço define ações e seus argumentos, o estado das variáveis, tipos de dados, limites e características dos eventos:

```
<?xml version="1.0"?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0">
  <specVersion>
    <major>1</major>
    <minor>0</minor>
  </specVersion>
  <actionList>
    <action>
      <name>nome_da_ação</name>
      <argumentList>
        <argument>
          <name>nome_do_parâmetro</name>
          <direction>in xor out</direction>
          <retval />

<relatedStateVariable>nome_da_variável_de_estado</relat
edStateVariable>
        </argument>
      </argumentList>
    </action>
  </actionList>
  <serviceStateTable>
    <stateVariable sendEvents="yes">
      <name>nome_da_variável</name>
      <dataType>tipo da variável</dataType>
      <defaultValue>valor_padrão</defaultValue>
      <allowedValueList>
        <allowedValue>domínio</allowedValue>

      </allowedValueList>
    </stateVariable>
    <stateVariable sendEvents="yes">
      <name>nome_da_variável</name>
      <dataType>tipo_da_variável</dataType>
      <defaultValue>valor_padrão</defaultValue>
      <allowedValueRange>
        <minimum>valor mínio</minimum>
        <maximum>valor máximo</maximum>
        <step>incremento</step>
      </allowedValueRange>
    </stateVariable>
  </serviceStateTable>
</scpd>
```

Para recuperar uma descrição, o ponto de controle envia a seguinte mensagem:

```
GET caminho HTTP/1.1
HOST: servidor:porta
ACCEPT-LANGUAGE: linguagem do ponto de controle
```

E recebe a resposta :

```
OS/1.1 200 OK
CONTENT-LANGUAGE: linguagem utilizada
CONTENT-LENGTH: Bytes do corpo
CONTENT-TYPE: text/xml
DATE: data da resposta
```

3.5.4 Controle

Após recuperar a descrição do dispositivo, o ponto de controle tem as condições necessárias para controlá-lo. Ele utiliza o *Simple Object Access Protocol* (SOAP) para enviar uma requisição de ação a um dos serviço do dispositivo através de uma mensagem para a URL de controle do serviço escolhido, recuperada da descrição do dispositivo. As mensagens de controle também são expressas em XML e as respostas são valores específicos ou códigos de erro.

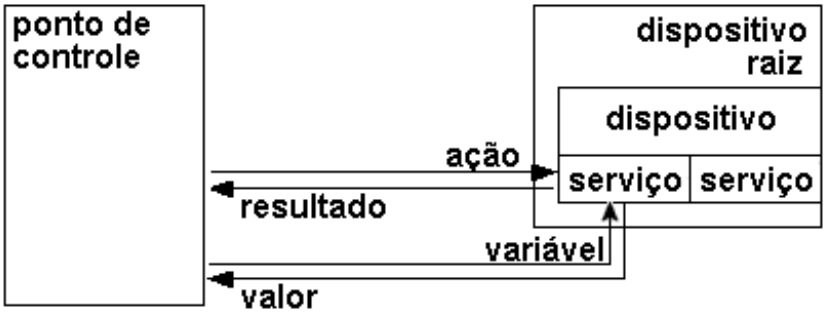

FIGURA 3.7 – Controlando um serviço.

Para invocar uma ação, um ponto de controle deve mandar uma mensagem com o método `POST` do OS, no seguinte formato:

```
POST caminho_controle URL HTTP/1.1
HOST: servidor:porta
CONTENT-LENGTH: bytes do corpo
CONTENT-TYPE: text/xml; charset="utf-8"
```

```
SOAPACTION: "urn:schemas-upnp-
org:service:serviceType:v#actionName"

<s:Envelope
    xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"

s:encodingStyle="http://schemas.xmlsoap.org/soap/encodi
ng/">
  <s:Body>
    <u:actionName xmlns:u="urn:schemas-upnp-
org:service:serviceType:v">
      <argumentName>valor</argumentName>
    </u:actionName>
  </s:Body>
</s:Envelope>
```

Caso a requisição, através do método `POST`, for rejeitada com uma resposta "`405 Method Not Allowed`", então o ponto de controle deve enviar uma segunda requisição com o método `M-POST` e `MAN`, no seguinte formato:

```
M-POST path of control URL HTTP/1.1
HOST: servidor:porta
CONTENT-LENGTH: bytes do corpo
CONTENT-TYPE: text/xml; charset="utf-8"
MAN: "http://schemas.xmlsoap.org/soap/envelope/"; ns=01
01-SOAPACTION: "urn:schemas-upnp-
org:service:serviceType:v#actionName"
```

Uma resposta à requisição deve ser retornada ao ponto de controle no intervalo de 30 segundos. Tarefas que necessitem de mais tempo para serem completadas devem retornar antecipadamente e enviar um evento quando finalizadas. O formato da resposta:

```
HTTP/1.1 200 OK
CONTENT-LENGTH: bytes do corpo
CONTENT-TYPE: text/xml; charset="utf-8"
DATE: data da resposta
EXT:
SERVER: versão_do_OS UPnP/1.0 versão_do_produto

<s:Envelope
    xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"

s:encodingStyle="http://schemas.xmlsoap.org/soap/encodi
ng/">
  <s:Body>
```

```
    <u:actionNameResponse xmlns:u=”urn:schemas-upnp-
org:service:serviceType:v”>
      <argumentName>valor</argumentName>
    </u:actionNameResponse>
  </s:Body>
</s:Envelope>
```

Em caso de erro, o serviço deve enviar uma mensagem de erro, ainda dentro do intervalo de resposta. Os argumentos das respostas não devem ser utilizados para a sinalização de erros. Uma mensagem de erro tem o seguinte formato:

```
HTTP/1.1 500 Internal Server Error
CONTENT-LENGTH: bytes do corpo
CONTENT-TYPE: text/xml; charset=”utf-8”
DATE: data da resposta
EXT:
SERVER: versão_do_SO UPnP/1.0 versão_do_produto

<s:Envelope
    xmlns:s=”http://schemas.xmlsoap.org/soap/envelope/”

s:encodingStyle=”http://schemas.xmlsoap.org/soap/encodi
ng/”>
  <s:Body>
    <s:Fault>
      <faultcode>s:Client</faultcode>
      <faultstring>UPnPError</faultstring>
      <detail>
        <UPnPError xmlns=”urn:schemas-upnp-org:control-
1-0”>
          <errorCode>código_erro</errorCode>
          <errorDescription>descrição do
erro</errorDescription>
        </UPnPError>
      </detail>
    </s:Fault>
  </s:Body>
</s:Envelope>
```

Os erros previstos são listados na tabela a seguir:

TABELA 3.1 – Erros para ações de controle.

Código	Mensagem	Descrição
401	Invalid Action	Ação solicitada inexistente.
402	Invalid Args	Argumentos inválidos.
403	Out of Sync	Fora de sincronismo.
501	Action Failed	O serviço não pode ser invocado neste momento.
600-699	Reservado	Erros comuns, definidos pelo Fórum.
700-799	Reservado	Erros específicos, definidos pelo Fórum.
800-899	Reservado	Erros específicos, definidos pelo fabricante.

Além de invocar ações, o ponto de controle pode requisitar valores das variáveis de estado. Este procedimento é feito através de uma mensagem de requisição, que pode solicitar uma única variável de estado por vez. Para solicitar mais de uma variável, várias mensagens são necessárias. O formato da mensagem:

```
POST caminho URL HTTP/1.1
HOST: servidor:porta
CONTENT-LENGTH: bytes do corpo
CONTENT-TYPE: text/xml; charset="utf-8"
SOAPACTION: "urn:schemas-upnp-org:control-1-
0#QueryStateVariable"

<s:Envelope
    xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"

s:encodingStyle="http://schemas.xmlsoap.org/soap/encodi
ng/">
  <s:Body>
    <u:QueryStateVariable xmlns:u="urn:schemas-upnp-
org:control-1-0">
      <u:varName>variável</u:varName>
    </u:QueryStateVariable>
  </s:Body>
</s:Envelope>
```

A resposta segue as mesmas regras do retorno de serviços: deve ser retornada no intervalo de 30 segundos, as falhas são tratadas pela aplicação e as mensagens de erro têm um formato especial. O formato da resposta:

```
HTTP/1.1 200 OK
CONTENT-LENGTH: bytes do corpo
CONTENT-TYPE: text/xml; charset="utf-8"
DATE: data da resposta
```

```
EXT:
SERVER: versão_do_SO UPnP/1.0 versão_do_produto

<s:Envelope
    xmlns:s=”http://schemas.xmlsoap.org/soap/envelope/”

s:encodingStyle=”http://schemas.xmlsoap.org/soap/encodi
ng/”>
  <s:Body>
    <u:QueryStateVariableResponse xmlns:u=”urn:schemas-
upnp-org:control-1-0”>
      <return>valor da variável</return>
    </u:QueryStateVariableResponse>
  </s:Body>
</s:Envelope>
```

Os erros previstos são listados na tabela a seguir:

TABELA 3.2 – Erros para a requisição de variáveis.

Código	Mensagem	Descrição
404	Invalid Var	Variável inexistente neste serviço.
600-624	Reservado	Erros comuns, definidos pelo Fórum.
625-649	Reservado	Reservado para uso futuro.
650-674	Reservado	Erros específicos, definidos pelo Fórum.
675-699	reservado	Erros específicos, definidos pelo fabricante.

3.5.5 Eventos

Um serviço (publicador) notifica atualizações quando suas variáveis internas mudam, e um ponto de controle (assinante) pode inscrever-se para receber tais notificações. As notificações são feitas utilizando-se o GENA, e os dados são expressos em XML. Quando o assinante se inscreve para receber as notificações, um evento especial é gerado, contendo os nomes e valores de todas as variáveis, permitindo ao assinante iniciar seu modelo com o estado atual do serviço. Para que possam existir múltiplos assinantes, todos os inscritos recebem notificações contendo todas as variáveis.

O publicador deve aceitar tantas inscrições quantas ele puder atender e é recomendado que o mesmo persista as inscrições para que, em caso de perda da conexão à rede, possa acelerar a recuperação do seu estado anterior. As mensagens são enviadas aos assinantes, dos quais são armazenadas as seguintes informações:

- Identificador único de inscrição: É gerado pelo publicador em resposta à mensagem de inscrição e deve ser único[10] durante o período da assinatura.
- URL de entrega: provida pelo assinante na mensagem de assinatura.
- chave de seqüência: recebe o valor 0 para a primeira notificação e é incrementada a cada nova notificação, para que o assinante possa conferir o recebimento de cada notificação.
- duração: intervalo de tempo pelo qual a assinatura é válida, ou o valor `infinite`, que indica uma assinatura permanente.

A inscrição é feita através de uma mensagem de inscrição:

```
SUBSCRIBE caminho_publicador HTTP/1.1
HOST: servidor:porta
CALLBACK: <URL de entrega>
NT: upnp:event
TIMEOUT: duração da assinatura
```

Se a inscrição é aceita, o publicador deve responder, num intervalo de 30 segundos, com um identificador único para a inscrição e a duração:

```
HTTP/1.1 200 OK
DATE: data da resposta
SERVER: versão_do_SO UPnP/1.0 versão_do_produto
SID: uuid:subscription-UUID
TIMEOUT: duração da assinatura
```

Os erros previstos são listados na tabela a seguir:

TABELA 3.3 – Erros para a inscrição.

Código	Mensagem	Descrição
400	Bad Request	Cabeçalho incompatível: presença de SID e (NT ou CALLBACK)
412	Precondition Failed	Sem CALLBACK ou URL com problema
412	Precondition Failed	NT diferente de upnp:event
5xx		Se o publicador não puder efetuar a inscrição, deve responder com os códigos de erro do HTTP (5xx)

Para manter a inscrição ativa, o assinante deve renovar a inscrição antes que ela expire, enviando uma mensagem de renovação, cuja resposta é igual a da inscrição.

[10] É recomendada a utilização de um método de geração universal para garantir a unicidade do identificador.

```
SUBSCRIBE caminho_publicador HTTP/1.1
HOST: servidor:porta
SID: uuid:subscription UUID
TIMEOUT: duração da assinatura
```

Caso a inscrição expire, o identificador de inscrição torna-se inválido e o publicador cessa o envio de notificações ao assinante. Para cancelar uma inscrição, o assinante deve enviar uma mensagem de cancelamento. Este procedimento é recomendado para evitar tráfego desnecessário na rede. Caso não seja possível, a inscrição continuará válida até o final de sua duração.

```
UNSUBSCRIBE caminho_publicador HTTP/1.1
HOST: servidor:porta
SID: uuid:subscription UUID
```

Os erros previstos são listados na tabela a seguir:

TABELA 3.4 – Erros para a renovação da inscrição.

Código	Mensagem	Descrição
400	Bad Request	Cabeçalhos incompatíveis: SID e (NT ou CALLBACK) presentes
412	Precondition Failed	SID inválido
412	Missing SID	SID vazio ou ausente

Para enviar uma notificação, um publicador deve enviar uma mensagem com o método `NOTIFY`, conforme o formato:

```
NOTIFY caminho_entrega HTTP/1.1
HOST: servidor:porta
CONTENT-TYPE: text/xml
CONTENT-LENGTH: Bytes do corpo
NT: upnp:event
NTS: upnp:propchange
SID: uuid:subscription-UUID
SEQ: chave de seqüência

<e:propertyset xmlns:e=”urn:schemas-upnp-org:event-1-
0”>
  <e:property>
    <variableName>new value</variableName>
  </e:property>
  Other variable names and values (if any) go here.
</e:propertyset>
```

Para confirmar o recebimento, o assinante deve responder no intervalo de 30 segundos. Se o publicador não receber uma confirmação neste intervalo, ele

deve desistir de enviar a mensagem corrente, mas deve manter o assinante na sua lista de assinantes até o final da duração da assinatura. O formato da resposta:

```
HTTP/1.1 200 OK
```

Os erros previstos são listados na tabela a seguir:

TABELA 3.5 – Erros para a notificação.

Código	Mensagem	Descrição
412	Precondition Failed	SID ausente
412	Precondition Failed	SID inválido
400	Bad Request	Cabeçalhos NT ou NTS ausentes
412	Precondition Failed	NT diferente de upnp:event
412	Precondition Failed	NTS diferente de upnp:propchange

3.5.6 Apresentação

Diferente dos modelos para dispositivos e serviços, a apresentação fica inteiramente a critério do fabricante, não sendo controlada pelo Fórum. Se o dispositivo possuir uma URL para apresentação, então o ponto de controle pode recuperá-la e, dependendo das suas características, utilizá-la para fornecer uma interface de controle e status ao usuário final.

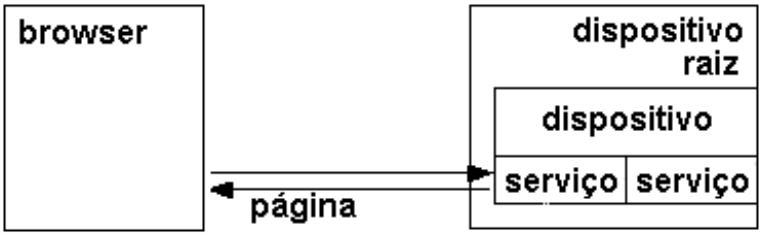

FIGURA 3.8 – Nível de apresentação do UPnP.

O nível de interação depende unicamente das características da página, que dever ser compatível com HTML 3.0 ou posterior e deve levar em consideração as características dos *browsers* onde são executadas. Assim como nas páginas web, recomenda-se um balanceamento entre as possibilidades tecnológicas e a compatibilidade com os *browsers*.

4 Protótipo

Com o objetivo de por em prática as tecnologias estudadas, no contexto residencial, um protótipo é proposto. Sua especificação é uma etapa muito importante do trabalho, pois sobre ela são codificadas implementações distintas em Jini e UPnP. Este capítulo trata da modelagem deste protótipo, de modo a evitar ambigüidades, ou situações não previstas, na hora da implementação.

4.1 Domínio da aplicação

O protótipo contempla alguns dispositivos existentes num ambiente residencial e serve como base para a implementação dos *workloads*. Os seguintes dispositivos foram escolhidos para serem implementados:

1. relógio
2. módulo utilitário
3. sensor de temperatura
4. condicionador de ar
5. MP3Player
6. medidor de energia elétrica

O critério de escolha dos dispositivos foi a possibilidade de cobrir as seguintes variações: dispositivo simples, dispositivo complexo, dispositivo composto e dispositivo de controle. Os componentes simples são o módulo utilitário, o relógio e o sensor de temperatura. O condicionador de ar, dispositivo composto, é modelado com o re-uso dos componentes relógio, utilitário e sensor de temperatura. O mp3player é um dispositivo um pouco mais complexo, com mais funcionalidades, e o medidor de energia implementa funções de controle. A figura a seguir ilustra fisicamente os dispositivos escolhidos para o protótipo:

FIGURA 4.1 – Visão geral dos dispositivos do protótipo

Além dos dispositivos, são necessários dois módulos adicionais: um para o controle e outro para o monitoramento do protótipo. O módulo de controle é responsável pela interação do usuário com os dispositivos. É através dele que o usuário irá comandar os dispositivos implementados. O módulo monitor é o responsável pela captura de dados relevantes do funcionamento do protótipo, agindo como um *log* do sistema.

Cada dispositivo do protótipo é modelado e implementado como uma aplicação em ambas as tecnologias estudadas, JINI e UPnP, levando-se em consideração, mas não necessariamente, suas propriedades e funções conhecidas no mundo real. As aplicações podem ser executadas em quaisquer computadores ou processadores que suportem as tecnologias e seus pré-requisitos, como o TCP/IP e a JVM, por exemplo. Elas não têm interface física com dispositivos reais, visto que tal tarefa depende de hardware especializado ou implementação de conversores, que fogem do escopo deste trabalho.

É importante salientar que a interface com o usuário não está no foco deste trabalho.

4.2 Objetos do domínio

A seguir são modelados e descritos os componentes do protótipo. Os componentes são os seis dispositivos e os dois módulos adicionais, totalizando oito componentes. A modelagem é feita através da *Unified Modeling Language* (UML) [OMG 2003] por ser uma linguagem orientada a objetos, padronizada, e que oferece recursos para a conversão do modelo em código, facilitando a etapa de implementação. A ferramenta utilizada para os diagramas UML é a Plastic[11] v2.0.

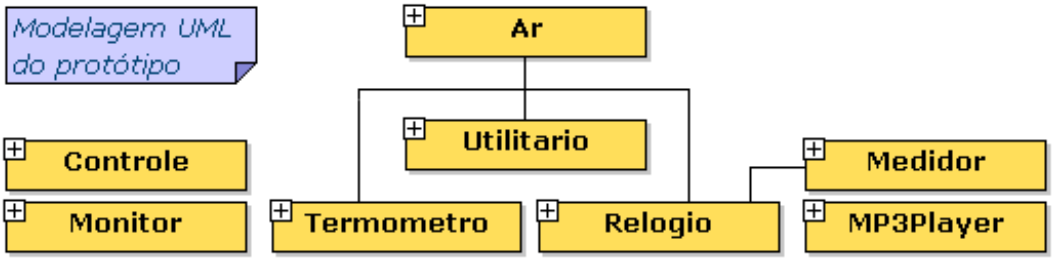

FIGURA 4.2 – Visão geral do protótipo em UML

[11] disponível para avaliação em http://www.plasticsoftware.com

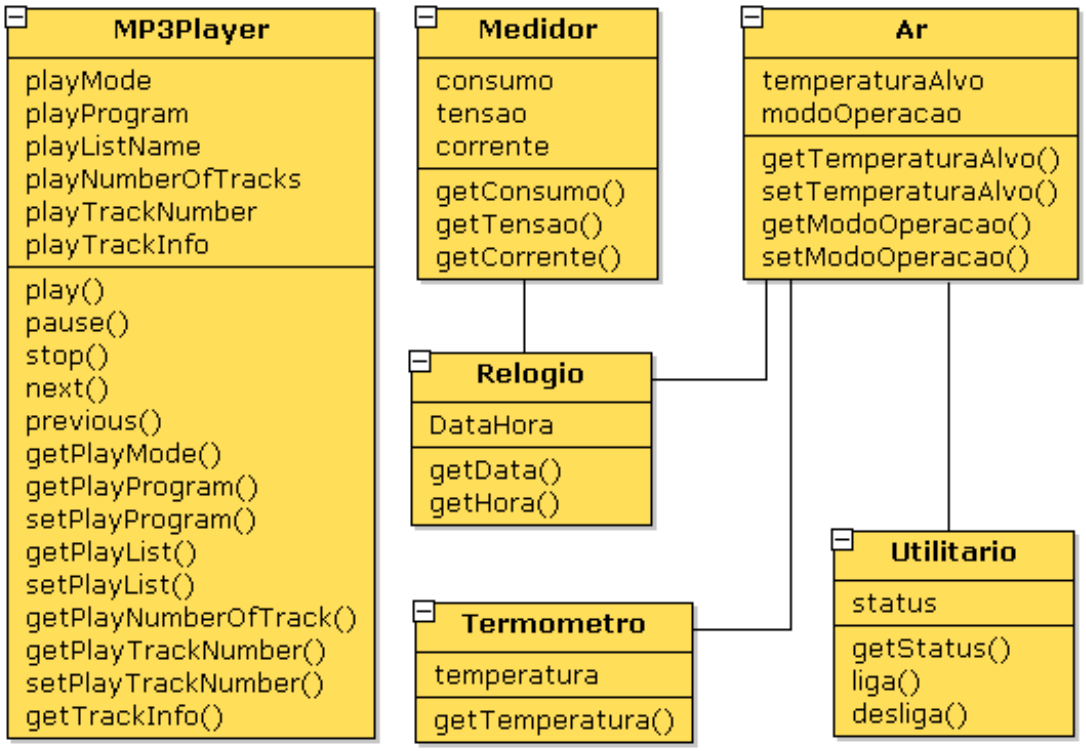

FIGURA 4.3 – Detalhamento dos componentes em UML.

A seguir, cada componente é detalhado, levando-se em consideração os modelos e *checklists* de serviços e dispositivos do UPnP, visto que Jini não oferece suporte para esta tarefa. Para cada componente são especificados: Variáveis de estado, Eventos, Ações e Funcionamento esperado.

4.2.1 Módulo Utilitário

É o componente responsável pelo acionamento e desligamento de qualquer dispositivo conectado à rede elétrica.

TABELA 4.1 - Descrição do Utilitário

Utilitario				
Variáveis:				
nome	**Tipo**	**val possível**	**val padrão**	**unidade**
status	lógico	0/1	0	
Ações:				
nome	**descrição**			
getStatus	Retorna o estado do utilitário: 0 ou 1			
liga	Liga o utilitário			
desliga	Desliga o utilitário			
Operação:				

O dispositivo é iniciado desligado (status=0). Nas ações de liga e desliga, após o acionamento físico o valor da variável status é atulizado refletindo o estado do dispositivo.

4.2.2 Sensor de Temperatura

Componente passivo que retorna a temperatura quando solicitado.

TABELA 4.2 - Descrição do Sensor de temperatura.

Termometro				
Variáveis:				
Nome	**tipo**	**val possível**	**val padrão**	**unidade**
Temperatura	numérico	-40.0 a 40.0	0	℃
Ações:				
Nome	**descrição**			
GetTemperatura	Retorna o valor numérico da temperatura			
Operação:				
Retorna o valor da temperatura quando solicitado.				

4.2.3 Relógio

Dispositivo responsável por medir o tempo, retornando informações sobre data e hora local. Em um ambiente real, seria uma fonte de tempo UTC parametrizada para retornar dados locais.

TABELA 4.3 - Descrição do Relógio

Relogio				
Variáveis:				
Nome	**Tipo**	**val possível**	**val padrão**	**unidade**
datahora	timestamp			
Ações:				
Nome	**Descrição**			
GetDataHora	Retorna o valor da variável datahora			
GetData	Retorna a data no formato DD/MM/AAAA			
GetHora	Retorna a hora no formato HH:MM:SS			
Operação:				
Retorna datahora, Data ou a Hora, conforme solicitado e no formato definido				

4.2.4 Condicionador de Ar

Dispositivo responsável pelas condições específicas de conforto e boa qualidade do ar, em recintos fechados, adequadas ao bem-estar dos ocupantes. É ajustado através dos seus modos de operação e do valor da temperatura desejada.

TABELA 4.4 - Descrição do condicionador de temperatura (ar).

Ar				
Componentes:				
Nome	**descrição**			
utilitario	Dispositivo interno			
relogio	Dispositivo na rede			
termometro	Dispositivo na rede			
Variáveis:				
Nome	**tipo**	**val possível**	**val padrão**	**unidade**
temperaturaAlvo	numérico	-10.0 a 40.0	20.0	℃
modoOperação	literal	desligado ventilação automático	automático	-
Ações:				
Nome	**Descrição**			
getTemperaturaAlvo	Retorna o valor da temperatura-alvo			
setTemperaturaAlvo	Ajusta o valor da temperatura-alvo			
getModoOperacao	Retorna o modo de operação atual			
setModoOperacao	Ajusta o valor do modo de operação atual			
Operação:				
Inicialmente procura pelos dispositivos relógio e termômetro na rede. Em modo ventilação, simplesmente faz a movimentação do ar. Em modo automático tenta aproximar o valor da temperatura ambiente, lida através do termômetro externo, para o valor desejado (temperaturaAlvo). Pode utilizar a leitura do relógio para auxiliar no controle da temperatura, aproveitando-se do conhecimento da variação natural da temperatura durante as estações do ano.				

4.2.5 MP3Player

Dispositivo capaz de reproduzir arquivos no formato MP3

TABELA 4.5 - Descrição do MP3Player.

MP3Player				
Variáveis:				
nome	**tipo**	**val possível**	**val padrão**	**unidade**
playMode	literal	play pause stop	stop	
playProgram		normal Contínuo	normal	
playListName	literal	arquivo existente	-	
playNumberOfTracks	numérico	1-1000	-	
playTrackNumber	numérico	1-1000	1	
playTrackInfo	literal		-	
Ações:				
nome	**descrição**			
play	Inicia a execução do arquivo corrente			
pause	Interrompe temporariamente a execução do arquivo corrente.			
Stop	Interrompe a execução do arquivo corrente			
next	Passa para a próximo arquivo da lista			
previous	Passa para o arquivo anterior da lista			
getPlayMode	Retorna o modo de execução atual			
getPlayProgram	Retorna o programa de execução			
setPlayProgram	Seta o programa de execução			
getPlayList	Retorna a lista de músicas			
setPlayList	Atribui a lista de músicas			
getPlayNumberOfTrack	Retorna o número de trilhas da lista			
getPlayTrackNumber	Retorna o número da trilha atual			
setPlayTrackNumber	Atribui o número da trilha atual			
getTrackInfo	Retorna informação sobre a trilha			
Operação:				
O dispositivo inicia em modo de execução ‘stop’ e com programa de execução ‘normal’, (ou seja, parado e sem repetição). Após a carga de um arquivo ou lista de músicas, o dispositivo pode alterar o modo de execução através das ações (play, pause, stop, next, previous).				

4.2.6 Medidor de Energia Elétrica

Dispositivo responsável pela leitura dos valores de tensão, corrente e consumo do ambiente residencial.

TABELA 4.6 - Descrição do Medidor de Energia Elétrica.

Medidor				
Variáveis:				
Nome	**tipo**	**val possível**	**val padrão**	**unidade**
tensao	numérico	0 a 340	-	V
corrente	numérico	0 a 50	-	A
consumo	numérico	0 a 999999	-	KW/h
Ações:				
Nome	**Descrição**			
getTensao	Lê o valor da tensão atual			
getCorrente	Lê o valor de corrente atual			
getConsumo	Lê o valor do consumo atual			
Operação:				
Monitora os valores de tensão, corrente e consumo podendo controlar outros dispositivos em função destes valores.				

Neste trabalho será explorado somente a possibilidade de controle (liga/desliga) dos dispositivos, visto que um bom algoritmo de controle não é trivial.

4.2.7 Módulo de Controle

O módulo de controle é implementado de forma a disponibilizar ao usuário uma interface básica para as funcionalidades dos dispositivos implementados, visto que a interface não é o foco deste trabalho.

4.2.8 Módulo Monitor

O módulo monitor é o responsável por receber e registrar informações dos dispositivos presentes na rede. É modelado e implementado como um receptor de eventos distribuídos.

4.3 Relacionamento entre os objetos

O funcionamento do protótipo é dado pelo funcionamento simultâneo de cada um dos componentes modelados, na mesma rede local. Para explorar melhor as características de conectividade das tecnologias, um dos objetivos deste trabalho, cada dispositivo deve ser executado como um nodo da rede, forçando a comunicação entre eles.

A tabela a seguir ilustra a comunicação esperada entre os dispositivos propostos, onde x representa a presença de comunicação :

TABELA 4.7 - Comunicação entre os dispositivos.

	utilitario	relogio	termometro	ar	Mp3player	medidor	controle	monitor
utilitario								x
relogio				x		x		x
termometro				x				x
ar		x	x					x
MP3Player								x
medidor	x	x	x	x	x		x	x
controle	x	x	x	x	x	x		x
monitor								

Os dispositivos simples são passivos, fornecendo informações aos solicitantes quando interrogados. O condicionador de ar, dispositivo composto, necessita da presença de outros dispositivos para iniciar o seu funcionamento. O medidor e o módulo de controle podem enviar mensagens de controle aos outros dispositivos e o módulo monitor só recebe eventos.

5 Métricas e *Workloads*

Neste capítulo são abordadas as métricas e *workloads* propostos para a avaliação das tecnologias.

5.1 Métricas

Visto que este é um trabalho de comparação, a definição e o entendimento das métricas é fundamental para a avaliação dos resultados e para as conclusões. As métricas propostas neste trabalho objetivam a avaliação da funcionalidade dos dispositivos em ambas as tecnologias. Dois tipos de métricas fazem parte do conjunto de métricas deste trabalho: qualitativas e quantitativas.

5.1.1 Métricas Qualitativas

Foram definidas no decorrer do estudo dos capítulos 1 e 2, com base nas principais semelhanças e diferenças entre as duas tecnologias. Algumas informações estão disponíveis nas especificações, mas outras somente ficam evidentes com o estudo detalhado das suas arquiteturas e na implementação do protótipo. Estão divididas em 5 grupos:

1. **Propriedade**: Confronta informações sobre propriedade, licença e regulamentação das tecnologias.

2. **Arquitetura**: Compara os requisitos mínimos de software e hardware (*footprint*) para que as tecnologia possam funcionar em um dispositivo, componentes básicos, infra-estrutura de rede, identidade, protocolos e tipo de comunicação.

3. **Funcionalidades**: confronta as funcionalidades disponíveis para o funcionamento das tecnologias. São consideradas as seguinte tarefas: anúncio/descoberta, descrição, controle, eventos, apresentação, contratos e transações.

4. **Ambiente e Facilidades de programação**: Confronta as condições necessárias para colocar as tecnologias em funcionamento e alguns recursos disponíveis para facilitar o trabalho dos desenvolvedores.

5. **Percepção do usuário**: Avalia o valor percebido pelo usuário e o seu grau de satisfação quando em contato com as tecnologias.

5.1.2 Métricas Quantitativas

Foram definidas com base no funcionamento dos dispositivos. O ciclo de vida de um dispositivo, em ambas as tecnologias estudadas, compreende os seguintes eventos: boot, anúncio, registro, descoberta, utilização, e *shutdown*.

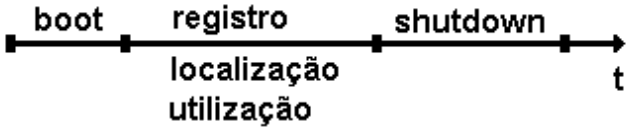

FIGURA 5.1 – Ciclo de vida de um dispositivo

1. ***Boot*****:** é o evento decorrido entre o momento do acionamento do dispositivo e o estado em que o mesmo se encontra pronto a executar a primeira instrução da aplicação, passando pela carga do *firmware* e/ou sistema operacional responsável por tarefas de suporte à aplicação.

2. **Registro**: é o evento compreendido entre o anúncio feito por um dispositivo e o seu reconhecimento por outro dispositivo (UPnP) ou pelo serviço de localização (Jini).

3. **Descoberta de um serviço/dispositivo**: é o evento onde um cliente (Jini) ou ponto de controle (UPnP) procura e descobre um serviço (Jini) ou um dispositivo (UPnP) específico na rede.

4. **Captura de um serviço/dispositivo**: é o evento onde um cliente recupera o *proxy* de um serviço ou um ponto de controle recupera as informações básicas do dispositivo a ser controlado.

5. **Resposta a ações**: é o evento compreendido entre a requisição de uma ação por parte do cliente ou ponto de controle, como por exemplo, o acionamento do play no mp3player, e a sua execução pelo dispositivo controlado, alterando o seu estado e retornando o resultado.

6. ***Shutdown***: a partir de uma ordem recebida, quanto tempo um dispositivo demora para desligar-se da rede de forma comportada, isto é, sem deixarem estados inconsistentes para trás. Presume o cancelamento de

quaisquer registros feitos anteriormente, para outros dispositivos (UPnP) ou serviço de localização (Jini).

Para cada evento, são definidas métricas quantitativas referentes ao tempo de cada evento, em mili-segundos, e a utilização da rede em bytes, dando origem ao seguinte conjunto de métricas: tempo/carga de boot, tempo/carga de registro, tempo/carga de descoberta de um serviço, tempo/carga de captura, tempo/carga de resposta a eventos, tempo/carga de *shutdown*.

5.2 *Workloads*

Um *workload* é uma carga de trabalho programada com o intuito de medir determinado comportamento, conforme métricas pré-definidas. O objetivo de um conjunto de *workloads* é a coleta de dados de forma metódica, de modo a propiciar uma análise consistente. Para este trabalho, cujo objetivo final é a análise comparativa entre as duas tecnologias em ambiente residencial, é definido um conjunto de *workloads* contemplando as métricas quantitativas.

A fim de manter maior proximidade à situação real, os *workloads* não são construídos de forma sintética, mas através da adaptação de aplicações reais baseadas no protótipo. Por isso, optou-se pela aquisição dos dados em alto nível, ou seja, na própria aplicação. Tal metodologia é válida, mas requer cuidado com a interferência na aquisição dos dados. Também procurou-se garantir a padronização das aplicações (protótipo), através de uma modelagem única dos dispositivos para ambas as tecnologias, e dos níveis inferiores, através da escolha do compilador, sistema operacional e hardware, detalhados em 5.3.

Para as medidas de tempo, dois tipos de eventos são distinguidos: eventos que iniciam e terminam no mesmo dispositivo e eventos que iniciam em um dispositivo e terminam em outro. Para os primeiros, a própria referência de tempo da máquina é suficiente, tomando o cuidado de conhecer a resolução do relógio e o comportamento da função da linguagem de programação que executa a operação. Para os demais, existem duas possibilidades: calcular o tempo de processamento em cada dispositivo e o tempo de transmissão entre eles, ou utilizar um relógio universal, como abordado em [AGN 2000]. O evento de *boot* não é considerado nos *workloads* visto que os dispositivos são simulados em PCs.

5.2.1 Workload-1

Tem por objetivo medir o tempo e a carga dos eventos de registro e *shutdown*. A figura a seguir ilustra a topologia preparada para este *workload*:

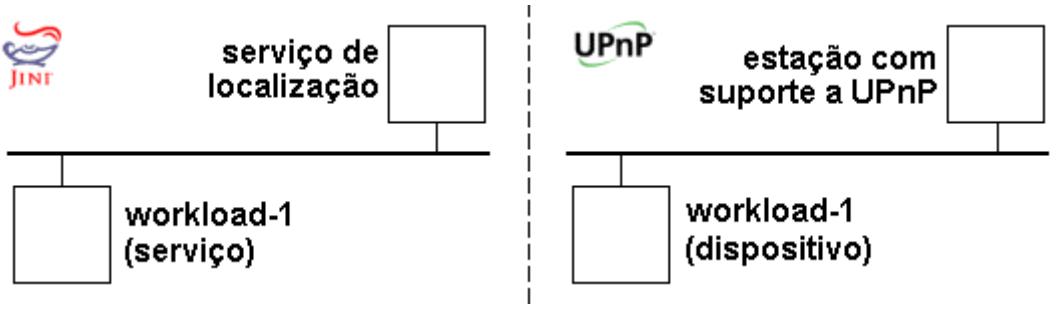

FIGURA 5.2 – Diagrama de topologia do *workload*-1 para Jini e UPnP.

Em Jini, um cliente é alterado para fazer a tomada de tempo nas instruções imediatamente anterior e posterior às chamadas da funções de registro e *shutdown*. Em UPnP, um dispositivo é alterado para fazer a tomada de tempo nas instruções imediatamente anterior e posterior às chamadas da função de anúncio. O diagrama de tempo abaixo auxilia na visualização dos tempos considerados :

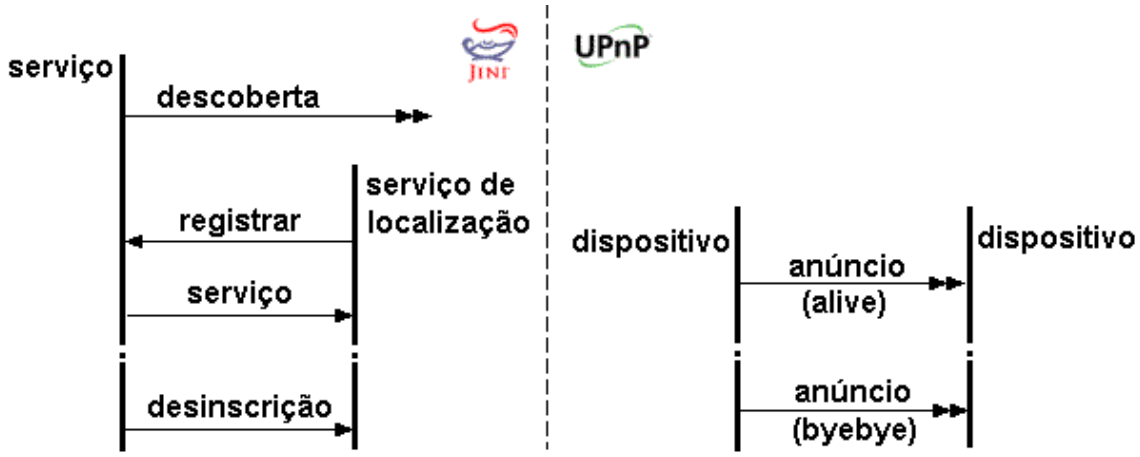

FIGURA 5.3 – Diagrama de tempo do *workload*-1 para Jini e UPnP.

5.2.2 Workload-2

Tem por objetivo medir o tempo e a carga dos seguintes eventos: descoberta, captura e resposta à uma requisição de ação, tanto para serviços quanto para dispositivos. A figura a seguir ilustra a topologia preparada para este *workload*:

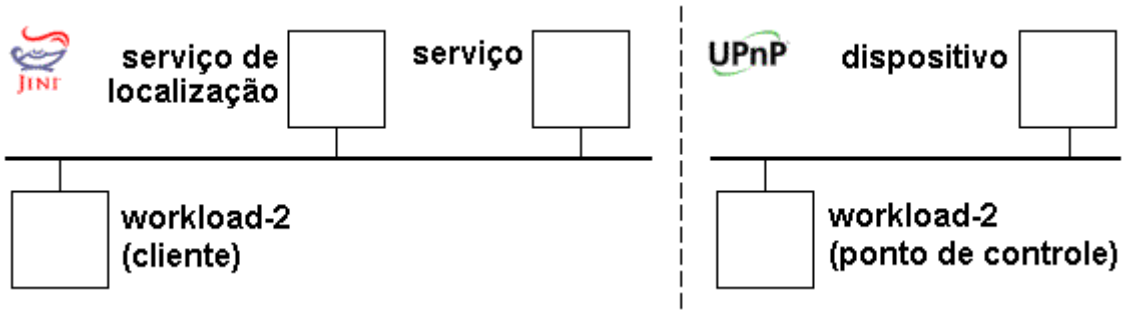

FIGURA 5.4 – Diagrama de topologia do *workload*-2 para Jini e UPnP.

Os clientes, em ambas as tecnologias, são modificado para medir o tempo antes e depois de cada uma das chamadas aos eventos. O diagrama de tempo a seguir auxilia na visualização dos tempos considerados:

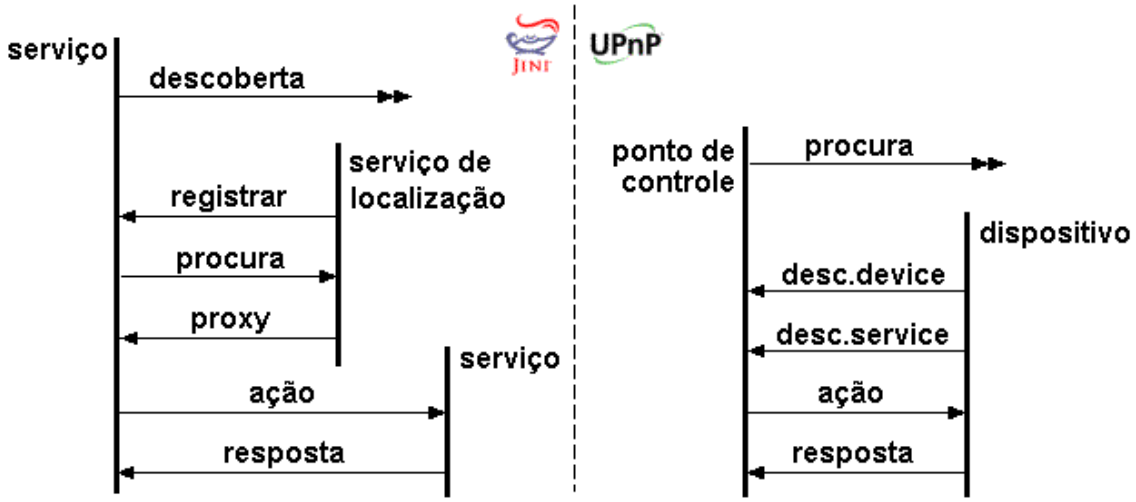

FIGURA 5.5 – Diagrama de tempo do *workload*-2 para Jini e UPnP.

5.3 Detalhes da implementação

5.3.1 Linguagem de programação

A linguagem Java foi escolhida para a implementação do protótipo e *workloads* em Jini, pois é a única alternativa gratuita conhecida atualmente para esta tecnologia. Com o objetivo de minimizar diferenças entre as implementações, e pela disponibilidade de SDKs em Java, também foi escolhida esta plataforma para a implementação em UPnP.

O *download* foi feito a partir do site da Sun[12] e a distribuição utilizada é a J2SE v1.4.1. Em todas as etapas do projeto, foi tomado o cuidado de se utilizar o mesmo compilador e JVM da distribuição.

5.3.2 Recursos temporais para as medições

A medição de tempo requer cuidado, pois os tempos a serem medidos não podem ser inferiores à resolução[13] do relógio utilizado. Em Java, a classe que provê acesso ao relógio da máquina é a `System.currentTimeMillis()` que retorna o número de mili-segundos a partir de 1º de janeiro de 1970 UTC. Apesar do retorno em mili-segundos, a resolução do relógio pode variar de acordo com a plataforma sobre a qual roda a JVM. Como exemplo, em [ROU 2003], o seguinte trecho de código procura conferir a resolução do relógio da máquina:

```
public static void main (String [] args) {

  for (int r = 0; r < 3000; ++ r)
    System.currentTimeMillis ();

  long t1 = System.currentTimeMillis ();
  long t2 = t1;

  for (int I = 0; I < 10; ++ I){
    while (t1 == t2)
      t1 = System.currentTimeMillis ();

     System.out.println("delta=" + (t1 - t2) + " ms");
     t2 = t1;
  }
}
```

A tabela a seguir apresenta a resolução obtida em algumas plataformas, utilizando o código acima apresentado:

TABELA 5.1 – Resolução de relógio em algumas plataformas.

Sistema Operacional	Hardware	Resolução (ms)
Windows 98	Pentium	50
Windows 2000/XP	Pentium	10
Linux 2.4	Pentium	1
SunOS 5.8	Sun Ultra-1	1

[12] Disponível em http://java.sun.com/downloads

[13] Nestre trabalho, **resolução** é definida como a menor unidade pela qual um relógio é atualizado.

5.3.3 Kits de Desenvolvimento (SDKs)

Para o desenvolvimento dos aplicativos Jini, a Sun disponibiliza em seu portal o *Jini Starter Kit* (JSK)[14]. A versão utilizada neste trabalho é a v1.2.1_001, composta por:

- Jini Technology Core Platform (JCP) provê a infra-estrutura de software básica da tecnologia Jini. Inclui a especificação, e as classes de interface correspondentes, para as funcionalidades básicas da tecnologia, como registro, eventos distribuídos, contratos e transações.
- Jini Technology Extended Platform (JXP), que provê uma infra-estrutura estendida de classes de interface de serviços e utilitários.
- Jini Software Kit (JSK), que provê implementação dos serviços de localização e gerenciador de transações (especificadas no JCP), implementações de classes utilitárias úteis a escrita de novas aplicações e serviços (especificadas no JXP) e serviços Jini, incluindo a tecnologia *Java Spaces* (também especificada no JXP)

Para o desenvolvimento dos aplicativos UPnP foram avaliadas 3 possibilidades de SKDs : Microsoft SDK[15], Siemens[16] [SIE 2001] e Celsius[17]. Foi adotado o SDK da Siemens por estar mais bem documentado e mais estável. O SKD da Microsoft foi descartado por exigir a linguagem C++ e a utilização do Internet Explorer Server como servidor de HTTP. O SDK da Celsius foi descartado por apresentar uma pequena incompatibilidade, apesar de vir acompanhado do código fonte.

[14] disponível no portal da Sun http://wwws.sun.com/software/communitysource/jini/download.html
[15] Disponível em http://www.microsoft.com/hwdev/tech/nonpc/UPnP/default.asp
[16] Disponível em http://plug-n-play-technologies.com/downloads.html
[17] Disponível em http://www.celsiustechnology.com/products/upnp.html

6 Execução e Resultados

6.1 Execução dos experimentos

Os experimentos foram realizados no Laboratório de Automação do Instituto de Informática. Com o objetivo de minimizar diferenças, foram utilizadas estações com as mesmas características:

TABELA 6.1 – Características do hardware utilizado.

Característica	Valor
Processador	Intel Pentium III
Freqüência	1 GHz
RAM	256 MB
Disco	10 GB
Sistema Operarional	Linux 2.4
JVM	1.4.1
Placa de Rede	Fast Ethernet

Para a conexão das estações foi utilizado um *hub* de 10 Mbits/s isolado da rede do Instituto. A captura e monitoração do tráfego entre as estações foi feita com a utilização do software *Ethereal*[18], o qual utiliza a biblioteca *WinPcap*[19], numa estação rodando Windows XP. As estações foram configuradas com resolução de nomes estática para evitar atrasos com a utilização do serviço de DNS. Mesmo com o isolamento da rede, alguns filtros foram necessários para evitar o aparecimento esporádico de pacotes indesejados (ARP, DHCP e SMB, entre outros), provenientes de serviços de rede básicos dos sistemas Linux e Windows.

Inicialmente o ambiente para a execução foi preparado e os aplicativos foram testados. Após, as máquinas foram reinicializadas e os experimentos realizados. Cada um dos 4 aplicativos (2 para cada *workload*) foi repetido 10 vezes. Para cada rodada, foram registrados os tempos de cada etapa prevista e a carga da rede. Uma amostra do tráfego gerado em cada aplicativo também foi armazenada para análise. Foram gastas cerca de duas horas na configuração do ambiente e mais quatro horas na execução dos experimentos. As tabelas com os dados colhidos encontram-se no Anexo-1.

[18] Disponível em http://www.ethereal.com
[19] Disponível em http://winpcap.polito.it/

6.2 Resultados

6.2.1 Qualitativos

Os dados qualitativos foram compilados nas tabelas a seguir, através da comparação entre os capítulos 2 e 3, e complementados com o conhecimento adquirido com a implementação do protótipo e dos *workloads*.

TABELA 6.2 – Resultados qualitativos: Propriedade.

Propriedade	**Jini**	**UPnP**
disponibilidade	1999	1999
proprietário	Sun Microsystems	Consórcio de empresas (liderado pela Microsoft)
licença	*Sun Community Source Licence* (open software)	Membership Agreement (open software)
grupo	Jini Community	UPnP Forum

Ambas tecnologias possuem o mesmo período de desenvolvimento, sendo Jini a mais conhecida. A única implementação gratuita e conhecida de Jini pertence à Sun enquanto UPnP possui diversas implementações, tanto gratuitas quanto pagas, por várias empresas. Ambos os fóruns são coordenados com a participação direta de funcionários da Sun e Microsoft.

TABELA 6.3 – Resultados qualitativos: Arquitetura.

Arquitetura	**Jini**	**UPnP**
footprint	JVM, pilha TCP, obtenção dinâmica de endereço IP, serviço de localização. (RMI é necessário para a utilização do Reggie)	Pilha TCP, obtenção dinâmica de endereço IP (DHCP, DNS ou Auto-IP).
componentes	cliente serviço (provedor de serviço) Serviço de Localização	ponto de controle serviço dispositivo (-)
rede	Não define padrão, mas utiliza TCP/IP. “velocidade e latência razoáveis”	Define o IP como nível mais baixo. “o meio deve suportar a banda necessária”
protocolos	IP,TCP,UDP,OS	IP, TCP, UDP, OS, SSDP, GENA, SOAP
comunicação	objetos distribuídos (RMI)	cliente/servidor
identidade	ServiceID (128 bits)	UUID obtido por algoritmo que garanta a unicidade
TTL	15	4

A primeira diferença entre as arquiteturas é a presença do serviço de localização em Jini. A JVM para Jini é um requisito bastante pesado quando considerados dispositivos com pouca capacidade de processamento. Ambas trafegam sobre redes IP, aproveitando-se de toda a infra-estrutura disponível atualmente, e baseiam-se em banda excedente para garantir o funcionamento dos serviços. A presença do serviço de localização e a diferença entre os mecanismos de comunicação (objetos distribuídos x cliente-servidor) impactam diretamente no comportamento das arquiteturas, conforme detalhado nos resultados das métricas quantitativas (6.2.2). Ambas utilizam algoritmos para a geração de IDs únicos, mas não existe garantia teórica para que dois serviços distintos não tenham o mesmo ID. Em Jini esta função é atribuída ao serviço de localização e cabe ao provedor do serviço a persistência do mesmo.

TABELA 6.4 – Resultados qualitativos: Funcionalidades.

Funcionalidades	**Jini**	**UPnP**
Descoberta	Para o serviço de localização existem 2 métodos: *unicast* e *multicast*. Para os demais serviços, o anúncio, a procura e a remoção são feitos através do serviço de localização.	São utilizadas mensagens do SSDP. As requisições podem ser *unicast* ou *multicast* para os anúncios, procura e remoção e as respostas são sempre *unicast*. São sugeridas 3+2d+k msgs para o anúncio de um dispositivo (3.5.2)
Descrição	Através da classe Entry, que descreve as propriedades do serviço. A classe não é padronizada, mas existe uma recomendação conhecida como *classes de conveniência.*	Através de documentos XML recuperados por OS, tanto para dispositivos quanto para serviços. O Fórum padroniza a descrição através de documentos específicos para cada dispositivo, baseados em *Device/Service Templates.*
Controle	Através da invocação dos métodos disponíveis nas interfaces dos serviços, tanto local quanto remotamente (RMI)	Através do SOAP, que utiliza XML e OS, de forma cliente/servidor.
Eventos	Mesmo conceito de eventos em Java, porém, estendido para funcionar remotamente.	Através do GENA, que utiliza XML e OS.
Apresentação	Aplicativo em Java.	Página em HTML.
Contratos	Gerenciados pelo serviço de localização. O cancelamento pode ser explícito ou por *timeout*, e o tempo expresso em mili-segundos.	Gerenciados pelos pontos de controle, através do tempo de duração recebido na ocasião do anúncio do dispositivo. O cancelamento pode ser

		explícito ou por *timeout* e o tempo expresso em segundos.
Transações	Transações ACID com protocolo *two-phase commit protocol* através de um Gerenciador de Transações	-

Novamente o serviço de localização introduz diferenças entre as arquiteturas. Outra diferença importante é a descrição dos dispositivos: enquanto em Jini ela é feita através de classes, sem um padrão específico, em UPnP esta descrição é bem padronizada e escrita em XML. Enquanto o controle e eventos são abordados de forma cliente-servidor em UPnP, através de OS, Jini aborda de forma mais elegante e eficiente através da utilização do RMI. A melhor eficiência do controle em Jini em relação ao UPnP é comprovada nos resultados das métricas quantitativas (6.2.2).

Avaliação do ambiente e facilidades de programação

Para esta métrica não foi utilizado um método científico. É baseada na prática adquirida durante toda a etapa de implementação do protótipo e dos *workloads*. O entendimento dos dois primeiros capítulos foi fundamental para saber o que estava ocorrendo e onde procurar quando o comportamento era diferente do esperado. A procura por ferramentas foi feita através de buscas na Internet e acompanhamento dos sites oficiais e listas de discussão.

Jini: A configuração do serviço de localização não é trivial, como constatado na literatura [NEW 2000], no acompanhamento da lista de discussão[20] da tecnologia e na configuração para o protótipo. Além dos problemas com a parametrização do ambiente Java, a implementação do serviço de localização também mostrou-se sensível ao serviço de DNS. Entretanto, após o seu estabelecimento, o ambiente comportou-se de modo estável, ficando ativo por mais de 30 dias durante os testes de implementação no laboratório sem a necessidade de ser reinicializado. Para a programação não foi encontrada nenhuma ferramenta específica além do JSK da Sun. Os aplicativos foram escritos em editores de texto e compilados em linha de comando. A ferramenta mais utilizada foi o *browser* de serviços que mostra os serviços contidos nos serviços de localização. A interface gráfica para disparo do ambiente (serviço de localização, seus componentes e serviços arbitrários) não foi utilizada, pois na prática constatou-se que *scripts* são mais eficientes.

UPnP: Praticamente não existe ambiente a ser configurado. As aplicações rodam diretamente nos dispositivos na forma de um programa único que em geral já incorpora o servidor de OS. A única configuração feita, como teste, foi a habilitação do serviço UPnP no Windows XP, para permitir a descoberta e controle de dispositivos UPnP através do Windows. Como a

[20] Arquivo disponível em http://archives.java.sun.com/archives/jini-users.html

linguagem de programação escolhida foi Java, vale o mesmo comentário feito para Jini sobre a sua parametrização. Na programação foram empregados editores de texto e XML, e compilação por linha de comando. O *browser* disponível no SDK da Siemens foi utilizado juntamente com o Windows XP para a interação com os dispositivos. Durante a finalização do trabalho foram encontradas e utilizadas algumas ferramentas bem interessantes desenvolvidas pela Intel[21]: *Device Sniffer*, *Device Spy*, *Device Validator* e *Device Builder*. Tais ferramentas são gratuitas e algumas requerem a instalação do *framework* .Net da Microsoft.

Percepção do usuário

Para esta métrica também não foi utilizado um método científico, visto que esta avaliação depende de um ambiente real e do valor percebido pelo usuário da tecnologia, que tende a ser subjetivo. Para simplificar o problema, podería-se avaliar somente o tempo de resposta dos dispositivos, conforme o seguinte teste: Diante do módulo de controle, o usuário deve interagir por alguns minutos com os dispositivos e no final da interação emitir um parecer binário sobre os tempos de resposta experimentados (se são semelhantes com os do dia-a-dia ou não).

6.2.2 Quantitativos

Os resultados quantitativos foram sintetizados a partir dos dados gerados pelos *workloads,* disponíveis no Anexo-1.

Workload-1:

FIGURA 6.1 – *Workload*-1: Tempo.

Observa-se um tempo de registro muito maior em Jini (fig.21) em relação à UPnP. Este tempo é reflexo da transferência do *proxy* `ServiceRegistrar`, que contém os métodos responsáveis pelo registro do objeto no serviço de localização. Em UPnP, tanto o registro quanto o *shutdown* é praticamente o tempo de broadcast UDP das mensagens, sendo que no registro há a transmissão da descrição do dispositivo em XML.

[21] Disponíveis em http://www.intel.com/labs/connectivity/upnp

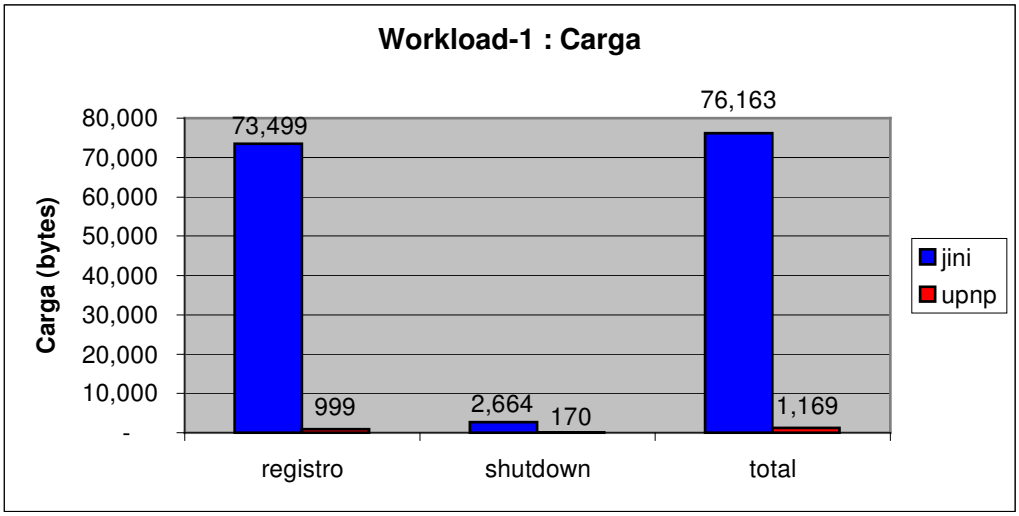

FIGURA 6.2 – *Workload*-1 – Carga

Na carga da rede (fig.22), novamente observa-se a influência da transferência do *proxy* `ServiceRegistrar`. Em Jini a carga é igual à transferência da soma do *proxy* registrar e do *proxy* do serviço a ser registrado. Em UPnP, a carga é praticamente a transferência da descrição do dispositivo, em XML.

Workload-2:

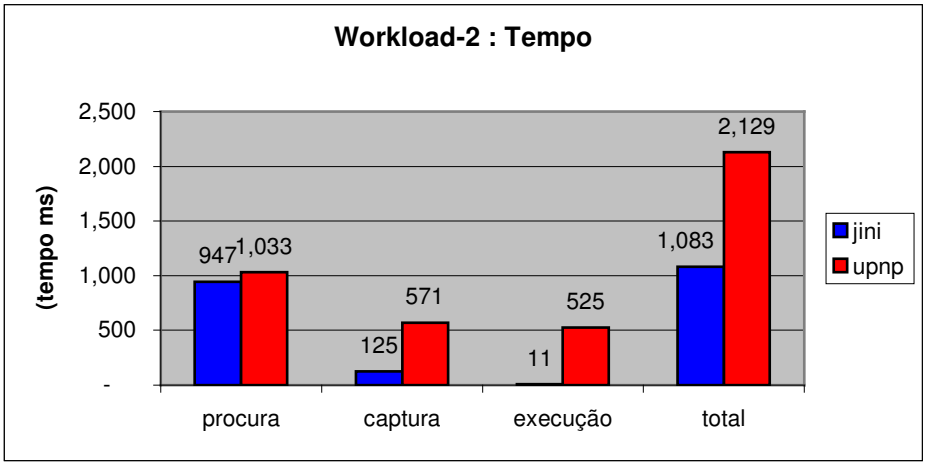

FIGURA 6.3 – *Workload*-2 : Tempo.

A análise de tempo do *workload*-2 (fig.23) nos mostra que: A procura é equivalente, sendo que Jini utiliza o serviço de localização e UPnP *broadcast*. Na captura dos dispositivos, Jini só precisa transferir o *proxy*, enquanto o UPnP necessita da descrição dos serviços e dispositivos aninhados. Na execução, Jini beneficia-se do RMI, recebendo os dados serializados (binários) enquanto UPnP necessita de conexões OS, para a transferência de XML (texto).

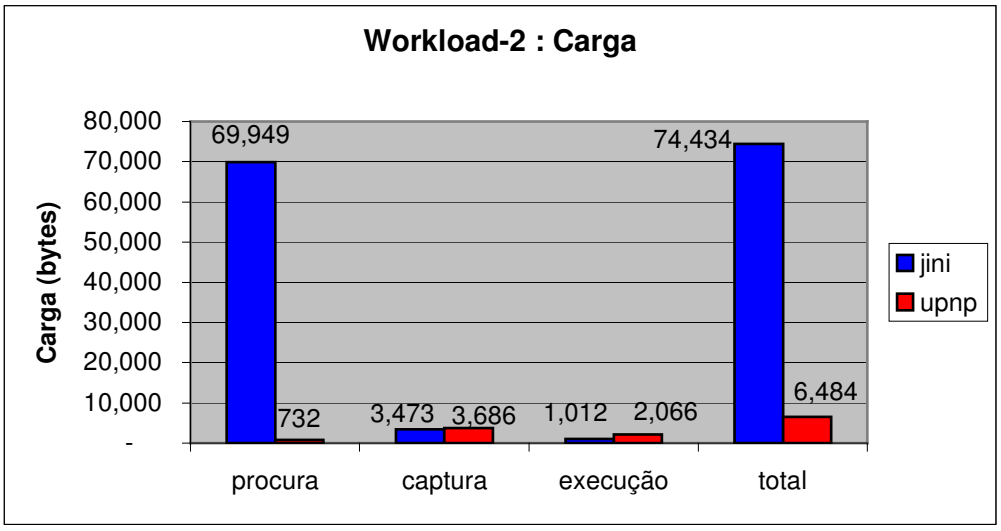

FIGURA 6.4 – *Worklolad*-2 : Carga.

Em relação à carga, Jini necessita da transferência do *proxy* `ServiceRegistrar` para a procura. A captura é equivalente: em Jini a transferência do *proxy* do serviço e em UPnP a transferência do XML de descrição do dispositivo. Na execução, UPnP tem um pouco mais de *overhead* em função do modelo cliente-servidor, com a transferência dos dados em XML.

Sobre os *Workloads*:

O desvio padrão observado nos tempos (Anexo-1), em Jini, ocorre em conseqüência da manutenção das conexões TCP entre o provedor de serviço e o serviço de localização. O baixo desvio padrão nos tempos registrados para UPnP (Anexo-1) deve-se à baixa manutenção da comunicação, feita através de UDP.

Os fatores que podem influenciar nos tempos e cagas dos evento de, em Jini e UPnP, são agrupados na tabela a seguir:

TABELA 6.5 – Fatores que influenciam no tempo dos eventos em Jini e UPnP

Evento	Jini	UPnP
Registro	Tamanho do *proxy*/objeto a ser registrado, transferido para o serviço de localização. Número de serviços de localização em que o serviço é registrado, pois o registro é feito seqüencialmente em cada um.	Tamanho do XML de descrição do dispositivo.
Procura	Número de serviços de localização	
Captura	Tamanho do *proxy*/objeto a ser capturado	Complexidade da descrição do dispositivo. (quanto mais complexo, maior a quantidade de descrições de serviços e dispositivos a serem transferidos
Execução	Quantidade de dados (binários)	Quantidade de dados (texto/XML)

Dados encontrados em [DAB 2001], onde é feita uma análise de escalabilidade entre as duas tecnologias, comprovam possíveis suposições sobre a escalabilidade dos *workloads* deste trabalho: Procura equivalente para ambas as tecnologias, com carga constante para um dispositivo e crescente para todos os dispositivos. Carga crescente nas execuções de UPnP, proporcionais ao número de dispositivos; onde Jini é menos sensível.

Entretanto, a redução do tempo na procura por dispositivos, observada em UPnP à medida que mais dispositivos do mesmo tipo encontram-se na rede, é um resultado que não foi suposto a partir dos *workloads* deste trabalho.

7 Conclusões

O desenvolvimento deste trabalho destaca a necessidade de novos padrões para a evolução da automação residencial em ambientes mais dinâmicos e com maiores benefícios para fabricantes e usuários. Foram comparadas duas tecnologias, Jini e UPnP, as quais diferem-se de outras tecnologias principalmente por suas características de *middleware*. Nos primeiros capítulos, ambas foram apresentadas de uma forma intermediária: não muito resumidas, como em um artigo, e nem tão detalhadas, quanto nas especificações técnicas das arquiteturas, de modo a obter-se um texto conciso que serve como base às etapas posteriores e para as pessoas interessada em conhecer melhor as tecnologias.

O protótipo auxiliou na validação do ciclo de desenvolvimento para dispositivos habituais em cada uma das tecnologias partindo-se de uma modelagem comum, e, principalmente, para elucidar dúvidas presentes na documentação, como no caso da utilização de UIDs na arquitetura Jini, por exemplo. Os *workloads* foram a base de toda a análise quantitativa e foi nesta etapa onde apareceram as maiores dificuldades, como a falta de resolução adequada para a medição dos tempos dos eventos mais breves e a sensibilidade da implementação de Jini à resolução de nomes (DNS). Os resultados desta etapa foram cruciais para o discernimento entre as tecnologias e os números evidenciaram alguns fatos despercebidos no estudo teórico.

Um dos principais resultados deste trabalho é a definição de um conjunto de métricas qualitativas e quantitativas para a avaliação não só das tecnologias deste trabalho, mas de quaisquer outras semelhantes e aplicáveis ao contexto residencial. Outro resultado importante é a análise dos resultados, no capítulo 6, que permite salientar as diferenças entre as tecnologias estudadas, resumidas a seguir :

Jini é mais flexível e pode ser utilizada para diversos fins além da automação residencial. O serviço de localização em Jini é um ponto crítico, um ponto a mais a ser configurado e monitorado (justamente quando um dos objetivos é evitar a configuração de dispositivos no ambiente residencial) em contrapartida aos benefícios oferecidos. A possibilidade de utilização de objetos distribuídos, através do uso do RMI, facilita a programação e oferece maior performance nas execuções remotas. O *footprint* necessário para a utilização da tecnologia, principalmente em função dos requisitos e performance das JVMs, é apontado como um dos empecilhos da utilização de Jini, apesar das soluções existentes.

UPnP assemelha-se a um *framework*, com modelos de dispositivos prontos a serem utilizados. Os dispositivos e serviços atuam de maneira

independente e os procedimentos de anúncio e descoberta são bastante simples e ágeis em função da utilização de mensagens UDP *broadcast*. A padronização dos dispositivos através dos modelos em XML é um ponto forte, embora degrade a performance para dispositivos muito complexos ou com muita execução distribuída.

Ao final deste trabalho, tanto Jini como UPnP já têm atualizações divulgadas: A versão 2.0 de Jini recém foi anunciada e UPnP já possui um grupo de trabalho empenhado na versão 2.0 do *framework*. Ambas prometem melhorias, principalmente em relação aos mecanismos de segurança.

Além dos resultados obtidos, acredito que a maior contribuição deste trabalho seja a exposição de um campo ainda pouco explorado e com bastante potencial. Independente do sucesso ou não das tecnologias estudadas, o mais importante é a aplicação dos seus conceitos no ambiente residencial.

7.1 Trabalhos futuros

No decorrer deste trabalho, algumas necessidades foram observadas e podem servir para a continuidade em trabalhos futuros. As mais importantes:

1. A continuidade na exploração do assunto, através do estudo de novas tecnologias semelhantes, como Zeroconf[22] (Apple Rendevouz[23]), Internet Zero (MIT) e Salutation pode cooperar para que se chegue mais rapidamente a um padrão.

2. O desenvolvimento e padronização de uma metodologia específica para a modelagem de dispositivos e ambientes residenciais que seja independente das tecnologias empregadas, que permita a integração entre elas, ao menos em alto nível.

3. O desenvolvimento de editores gráficos (IDE/RAD) para facilitar a modelagem e desenvolvimento de dispositivos e aplicações, visto que não é necessário dominar toda uma tecnologia para poder usufruir dos benefícios oferecidos por ela – assim como um programador Delphi não precisa conhecer toda a API do Windows para programar para a plataforma.

[22] http://www.zeroconf.org/
[23] http://www.apple.com/macosx/jaguar/rendezvous.html

4. Estudo de QoS e RT aplicados a redes residenciais, pois hoje ambas as tecnologias são baseadas em *banda excedente*, não oferecendo garantia teórica para o funcionamento de serviços críticos.

5. Aprofundamento das questões envolvendo a segurança deste tipo de tecnologia, além dos problemas já conhecidos[24], herdados pela utilização do protocolo IP. As vulnerabilidades podem trazer grandes transtornos aos usuários, como, por exemplo, um vírus residencial com o objetivo de descobrir e ligar todos os dispositivos de uma residência nos seus limites máximos, todos os dias, às 4 horas da manhã, ou sempre que não houver alguém na residência.

6. Exploração e aplicação das camadas superiores: Interface com usuário (HCI) e inteligência dos dispositivos (IA), no ambiente residencial.

[24] http://www.cert.org/advisories/CA-2001-37.html

Anexo 1 Dados gerados pelos *workloads*

Workload-1

	Jini					
	registro			shutdown		
	t (ms)	pacotes	bytes	t (ms)	pacotes	bytes
1	1,125	143	72,506	38	14	1,165
2	1,142	150	72,960	13	10	843
3	1,159	165	73,972	13	10	843
4	1,096	169	74,214	17	10	843
5	1,089	165	73,950	13	9	777
6	1,112	165	73,952	13	9	777
7	1,151	170	74,288	18	14	1,165
8	1,181	140	72,300	18	15	1,245
9	1,069	154	73,224	18	14	1,165
10	1,123	160	73,620	18	14	1,165
round(Media)	1,125	158	73,499	18	12	999
Media	1,124.70	158.10	73,498.60	17.90	11.90	998.80
Desvio Padrão	34.48	10.77	712.63	7.46	2.47	195.03
IC(95%)	0.68	0.21	14.13	0.15	0.05	3.87

	UPnP					
	registro			shutdown		
	t (ms)	pacotes	bytes	t (ms)	pacotes	bytes
1	41	8	2,644	30	1	170
2	39	8	2,644	30	1	170
3	29	8	2,644	29	1	170
4	41	8	2,644	30	1	170
5	41	8	2,644	30	1	170
6	40	8	2,644	30	1	170
7	41	8	2,644	31	1	170
8	32	8	2,644	31	1	170
9	41	8	2,644	30	1	170
10	31	8	2,644	30	1	170
round(Media)	38	8	2,644	30	1	170
Media	37.60	8.00	2,644.00	30.10	1.00	170.00
Desvio Padrão	4.88	-	-	0.57	-	-
IC(95%)	0.10			0.01		

Workload-2

	Jini								
	procura			captura			execução		
	t (ms)	pacotes	bytes	t (ms)	pacotes	bytes	t (ms)	pacotes	bytes
1	925	137	69,837	139	27	3,618	31	13	1,012
2	968	136	69,969	135	25	3,472	6	13	1,012
3	944	139	69,969	133	25	3,472	5	13	1,012
4	1,036	140	70,035	107	25	3,472	7	13	1,012
5	925	139	69,969	101	24	3,406	23	13	1,012
6	934	141	70,101	160	25	3,472	6	13	1,012
7	948	139	69,969	137	25	3,472	6	13	1,012
8	923	137	69,837	104	24	3,406	5	13	1,012
9	936	139	69,969	103	25	3,472	8	13	1,012
10	931	137	69,837	134	25	3,472	14	13	1,012
round(Média)	947	138	69,949	125	25	3,473	11	13	1,012
Média	947.00	138.40	69,949.20	125.30	25.00	3,473.40	11.10	13.00	1,012.00
Desvio Padão	34.09	1.58	88.27	20.09	0.82	57.74	8.97	-	-
IC (95%)	0.68	0.03	1.75	0.40	0.02	1.15	0.18		

	UPnP								
	procura			captura			execução		
	t (ms)	pacotes	bytes	t (ms)	pacotes	bytes	t (ms)	pacotes	bytes
1	166	4	804	566	18	3,686	521	15	2,066
2	1,959	3	724	591	18	3,686	527	15	2,066
3	921	3	724	576	18	3,686	523	15	2,066
4	91	3	724	563	18	3,686	527	15	2,066
5	1,594	3	724	569	18	3,686	525	15	2,066
6	964	3	724	572	18	3,686	524	15	2,066
7	480	3	724	570	18	3,686	520	15	2,066
8	872	3	724	566	18	3,686	527	15	2,066
9	2,003	3	724	572	18	3,686	526	15	2,066
10	1,279	3	724	566	18	3,686	528	15	2,066
round(Média)	1,033	3	732	571	18	3,686	525	15	2,066
Média	1,032.90	3.10	732.00	571.10	18.00	3,686.00	524.80	15.00	2,066.00
Desvio Padão	680.01	0.32	25.30	7.96	-	-	2.74	-	-
IC (95%)	13.48	0.01	0.50	0.16			0.05		

Anexo 2 Estrutura básica de clientes e serviços Jini

Cliente

Pela definição de cliente, um cliente deve ser capaz de encontrar um servidor de localização e requisitar um serviço desejado.

```
Import net.core.discovery.LookupLocator;
import net.core.lookup.ServiceRegistrar;
import net.core.lookup.ServiceItem;
import net.core.lookup.ServiceRegistration;
import java.rmi.RMISecurityManager;
import net.core.lookup.ServiceTemplate;

public class Cliente {

  public static void main(String argv[])
  {
    newClient();
  }

  public Cliente() {
// Inicialização das variáveis
LookupLocator       lookup         = null;
ServiceRegistrar    registrar      = null;
Servico   servico   = null;
Sring     servidor  = "jini://lookup_service_address/"

// Descoberta (unicast) do serviço de localização
try {
     lookup = new LookupLocator(servidor);
} catch(java.net.MalformedURLException e) {
     System.err.println("falha na procura do servidor:
" + e.toString());
     System.exit(1);
}

// Tratamento dos requisitos de segurança
System.setSecurityManager(new RMISecurityManager());

// Obtendo o objeto Registrar
try {
     registrar = lookup.getRegistrar();
} catch(java.io.IOException e) {
```

```
     System.err.println("falha obtendo registrar: " +
e.toString());
     System.exit(1);
} catch (java.lang.ClassNotFoundException e) {
     System.err.println("falha obtendo registrar: " +
e.toString());
     System.exit(1);
}

// Prepara o template do serviço para fazer a pesquisa
Class[] classes = new Class[] {Servico.class};
ServiceTemplate template = new ServiceTemplate(null,
classes, null);
try {
     servico = (Servico) registrar.lookup(template);
} catch(java.rmi.RemoteException e) {
     e.printStackTrace();
     System.exit(1);
}
```

Serviço

```
import java.rmi.RMISecurityManager;
import net.discovery.LookupDiscovery;
import net.discovery.DiscoveryListener;
import net.discovery.DiscoveryEvent;
import net.core.lookup.ServiceRegistrar;
import net.core.lookup.ServiceItem;
import net.core.lookup.ServiceRegistration;
import net.core.lease.Lease;
import net.core.lookup.ServiceID ;
import net.lease.LeaseListener;
import net.lease.LeaseRenewalEvent;
import net.lease.LeaseRenewalManager;

import java.io.*;


public class Servidor implements DiscoveryListener,
LeaseListener
{
protected LeaseRenewalManager leaseManager   = new
LeaseRenewalManager();
protected ServiceID      serviceID      = null;

// Disparo do Servidor
```

```
public static void main(String argv[]) {

     // Cria um objeto Servidor
     new Servidor();

     // Mantem o servidor vivo
     Object keepAlive = new Object();
     synchronized(keepAlive) {
          try {
               keepAlive.wait();
          } catch(java.lang.InterruptedException e) {
               System.err.println("Erro: " +
e.toString());
          }
     }
}


// Descoberta do serviço de localização, modo broadcast
public Servidor() {

     // controle de segurança
     System.setSecurityManager(new
RMISecurityManager());
     LookupDiscovery discover = null;

     // procura por um serviço de localização, de forma
assíncrona
     try {
          discover = new
LookupDiscovery(LookupDiscovery.ALL_GROUPS);
     } catch(Exception e) {
          System.err.println("falha na procura" +
e.toString());
          System.exit(1);
     }
     discover.addDiscoveryListener(this);
}

// Tratador dos serviços de procura descobertos
public void discovered(DiscoveryEvent evt) {

     // Cria um vetor com os registrars retornados
     ServiceRegistrar[] registrars =
evt.getRegistrars();

     // Registra o serviço em cada servidor encontrado
```

```
     for (int n = 0; n < registrars.length; n++) {
          ServiceRegistrar registrar = registrars[n];
          ServiceItem item = new ServiceItem(null, new
Servico(), null);
          ServiceRegistration reg = null;
          try { // efetua o registro
               reg = registrar.register(item,
Lease.FOREVER);
          } catch(java.rmi.RemoteException e) {
               System.err.println("erro tentando
registrar: " + e.toString());
               continue;
          }
          System.out.println("serviço registrado com
ID=" + reg.getServiceID());

          // Tratamento do contrato
          leaseManager.renewUntil(reg.getLease(),
Lease.FOREVER, this);

          // Recupera o ID do □equisi
          serviceID = reg.getServiceID();

          }
     }
}

// Tratamento de requisições descartadas
    public void discarded(DiscoveryEvent evt) {
     // Trata os eventos descartados
    }

// Notificador do contrato
    public void notify(LeaseRenewalEvent evt) {
   System.out.println("o contrato expirou:" +
evt.toString());
    }

} // Servidor
```

Bibliografia

[AGN 2000] AGNOLETTO, Alessandro Dario. **Cronus**: um serviço configurável de disponibilização e difusão do UTC para computadores PC. 2000. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[BER 96] BERNSTEIN, Philip A. Middleware: A Model for Distributed Systems. **Communication of ACM**, New York, v.39, n.2, p.86-98, Feb.1996.

[DAB 2001] DABROWSKI, Crhistopher. **Acessing the state-of-the-art in Dynamic Discovery of Ad Hoc Network Services**. Trabalho apresentado no NRC review meeting, Feb. 2001.

[EDW 99] EDWARDS, W. Keith. **Core JINI**. Upper Saddle River: Prentice Hall, 1999.

[FLA 99] FLANAGAN, David. **Java in a Nutshell.** 3rd ed. Sebastopol: O'Reilly & Associates, 1999.

[GER 99] GERSHENFELD, Neil. **When Things Start to Think**. [S.l.]: OWL Books, 1999.

[GRA 81] GRAY, J. The Transaction Concept: Virtues and Limitations. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, VLDB, 7., 1981, Cannes. **Proceedings…** Los Angeles: IEEE, 1991.

[GUP 2002] GUPTA, Rahul; TALWAR, Sumeet; AGRAWAL, Dharma. Jini Home Networking: A Step Toward Pervasive Computing. **Computer**, Los Alamitos, v.35, n.8, p.34-40, Aug. 2002.

[HAN 2000] HANA (The Home Automation and Networking Association). **FAQ**. 2000.
Disponível em: <http://www.homeautomation.org/about/faq.html>. Acesso em: jan. 2001.

[HIR 99] HIRSH, Haim. Roomservice, AI-Style. **IEEE Intelligent Systems**, Los Alamitos, v.14, n.2, p.8 – 19, Mar./Apr. 1999.

[MEA 2002] MEALLING, M.; LEACH, P.; SALZ, R. **A UUID URN Namespace**. 2002. Internet Draft.
Disponível em: <http://www.ietf.org/internet-drafts/draft-mealling-uuid-urn-00.txt>. Acesso em: jan. 2003.

[MIC 2000a] MICROSOFT CORPORATION. **Universal Plug and Play Device Arquitecture**. 2000.
Disponível em: <http://upnp.org>. Acesso em: jun. 2000.

[MIC 2000b] MICROSOFT CORPORATION. **An Overview of the Simple Control Protocol (SCP)**. 2000.
Disponível em: <http://www.microsoft.com/>. Acesso em: jun. 2000.

[MIC 2000c] MICROSOFT CORPORATION. **Understanding Universal Plug and Play**. 2000.
Disponível em: <http://upnp.org>. Acesso em: jun. 2000.

[MOZ 98] MOZER, Michael C. **The Neural Network House:** an environment that adapts to its inhabitants. 1998.
Disponível em: <ftp://ftp.cs.colorado.edu/users/mozer/papers/nnhadapt.os>. Acesso em: dez. 2000.

[NEG 96] NEGROPONTE, Nicholas. **Being Digital**. New York : Vintage Books, 1996.

[NEW 2000] NEWMARCH, Jan. **A Programmer's Guide to Jini Technology**. Apress, 2000.
Disponível em: <http://jan.netcomp.monash.edu.au/java/jini/tutorial/Jini.xml>. Acesso em: jan. 2001.

[NOR 2002] NORMAN, Donald A. Home Theater**:** Not Ready for Prime Time. **Computer**, Los Alamitos, v.35, n.6, p.100-102, June 2002.

[NOR 98] NORMAN, Donald A. **The Design of Everyday Things**. New York: Basic Books, 1998.

[RIG 2002] RIGOLE, Peter; HOLVOET, Tom; BERBERS, Yolande. Using Jini to integrate Home Automation in a distributed software-system. In: INTERNATIONAL CONFERENCE ON DISTRIBUTED COMMUNITIES ON THE WEB. DCW, 4., 2002. **Proceedings...** [S.l.:s.n.], 2002.

[ROU 2003] ROUBSTOV, Vladimir. My Kingdom for a Good Timer ! **Java World**, Jan. 2003.
Disponível em: <http://www.javaworld.com/javaworld/javaqa/2003-01/01-qa-0110-timing.html>. Acesso em: mar. 2003.

[SIE 2001] SIEMENS AG. **UPnP Stack Programming Guide**. 2001.
Disponível em: <http://www.plug-n-play-technologies.com/whitepapers.htm>. Acesso em: jan. 2003.

[SUN 2001a] SUN MICROSYSTEMS. **Jini Architecture Specification**. 2001. Disponível em: <http://java.sun.com>. Acesso em: mar. 2001.

[SUN 2001b] SUN MICROSYSTEMS. **Jini Technology Core Platform Specification**. 2001. Disponível em: <http://java.sun.com>. Acesso em: mar. 2001.

[SUN 99] SUN MICROSYSTEMS. **Jini Technology Architectural Overview.** 1999. Disponível em: <http://www.sun.com/jini/whitepapers/architecture.pdf>. Acesso em: nov. 2000.

[TAN 94] TANENBAUM, Andrew S. **Redes de Computadores**. Rio de Janeiro: Campus, 1994.

[WAL 2000] WALDO, Jim. Alive and Well – Jini Technology Today. **Computer**, Los Alamitos, v.33, n.6, p.107-109, June 2000.

[WAL 99] WALDO, Jim. The Jini Architecture for Network-Centric Computing. **Communications of the ACM,** New York**,** v.42, n.7, July 1999.