

BRiGaS: Um Servidor de Jogos Pervasivo para UPnP Usando o Arcabouço BRisa

Diogo Dutra Albuquerque
IC/UFAL

Vitor Normande Vieira
IC/UFAL

João Pedro Pontes
IC/UFAL

Leandro Melo de Sales
IC/UFAL

Leandro Dias da Silva
IC/UFAL

Resumo

O protocolo UPnP (*Universal Plug and Play*) visa prover uma forma de conexão simples, robusta e interativa entre dispositivos móveis, eletro-eletrônicos e até eletrodomésticos provenientes de diversos fabricantes, provendo e facilitando a convergência de serviços e aplicações. Apesar de ser um padrão internacional emergente (2008), ele tem se tornado bastante popular na indústria, permitindo novas funcionalidades com o desenvolvimento de diversas aplicações, introduzindo a nova era de computação pervasiva. Devido a pervasividade do UPnP, que utiliza um sistema de descoberta de dispositivos, torna-se possível a integração instantânea de novos dispositivos em uma rede local. Neste trabalho, uma nova utilidade para o protocolo UPnP é apresentada: prover serviços e funcionalidades para jogos. É introduzida a especificação de um padrão para servidor de jogos pervasivo e a implementação do mesmo: o BRiGaS (*BRisa Game Server*). O BRiGaS estende o arcabouço BRisa para o protocolo UPnP, adicionando funcionalidades para ambientes de jogos. A pervasividade provida e a possibilidade do uso de dispositivos portáteis e móveis facilita e torna mais divertida a integração e interação entre jogadores, além de favorecer a convergência.

Keywords:: UPnP, Games, Computação Pervasiva

Author's Contact:

{diogo.comp,vitunv,joaopedro,plima,leandroal,
leandrodds}@gmail.com

1 Introdução

Devido a explosão e popularização do consumo de dispositivos móveis e a ininterrupta evolução da tecnologia móvel, novos conceitos e tecnologias surgem a cada dia. A pervasividade permite que dispositivos entrem em uma rede local automaticamente interagindo e fazendo uso de serviços providos por outros elementos do domínio da rede. O paradigma da computação pervasiva objetiva que um dispositivo, que esteja presente em qualquer ambiente, possa se comunicar com várias redes de forma transparente e interativa [Weiser 1991]. O padrão UPnP (*Universal Plug and Play*) vem sendo amplamente utilizado para flexibilizar e facilitar a integração de dispositivos em uma rede local. Pela fácil implantação e por seu sistema de descoberta de dispositivos, somados à utilização de tecnologias abertas consagradas da internet como HTTP, XML e SOAP, este padrão é uma ótima escolha para criar sistemas pervasivos. Desta forma, cada vez mais a computação móvel está promovendo a convergência de mídias, tecnologias, serviços, informações e aplicações para proporcionar mais funcionalidades e conforto aos usuários nas mais diversas áreas da sociedade.

Devido às características do UPnP discutidas anteriormente, é útil criar um servidor de jogos usando a tecnologia UPnP. Neste caso é especificado um dispositivo UPnP centrado em jogos, definindo dessa forma um servidor de jogos interoperável, aceitando clientes de diversas naturezas como celulares, PDAs¹, *Internet Tablets*, computadores *desktop*, consoles de vídeo games e qualquer outro dispositivo eletrônico capaz de implementar o padrão, como exemplificado na Figura 1. Nesse cenário, as pessoas podem jogar em rede, interagindo com o(s) servidor(es) de jogos disponível(is) no ambiente sem a necessidade de configurar diretamente seus dispositivos.

¹ Personal Digital Assistants

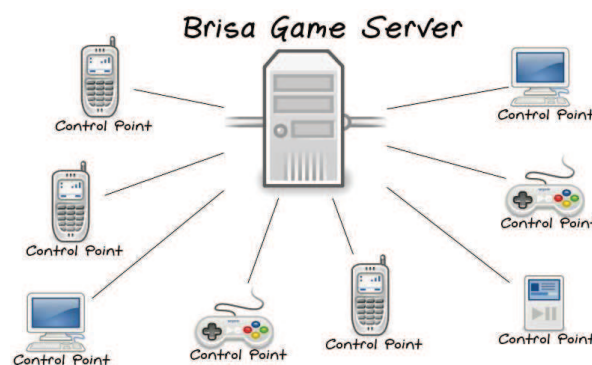


Figura 1: Interação com o BRisa Game Server (BRiGaS)

Porém, o uso do padrão UPnP no universo dos jogos atualmente está apenas em *media servers* e *renderers*, controle remoto e mecanismos para facilitar o acesso às redes locais. Portanto, neste trabalho é introduzida uma nova possibilidade ao uso do padrão UPnP: prover controle e interação para jogos através de serviços providos por um dispositivo centrado em jogos. Foi então especificado um padrão para servidores de jogos UPnP e também foi feita a implementação do mesmo, de código aberto e portátil que permite a interação entre diversos dispositivos, o BRiGaS (*BRisa Game Server*).

O BRiGaS está sendo desenvolvido utilizando o arcabouço BRisa [Guedes et al. 2008] para UPnP. O BRisa é um arcabouço de código livre escrito que permite a implementação de dispositivos, serviços e pontos de controle UPnP. A implementação do *Game Server* levou a questionamentos e alterações na implementação do arcabouço, influenciando e contribuindo em seu desenvolvimento. Também, levanta algumas questões sobre o funcionamento do padrão UPnP para o uso em certas aplicações, em que os dispositivos de uma rede local interajam de forma similar a jogos eletrônicos. Foi implementado um jogo de cartas como caso de teste para o servidor de jogos.

O restante deste trabalho está organizado da seguinte forma. O padrão UPnP é introduzido na Seção 2. Na Seção 3 é apresentada a especificação do BRiGaS seguida pela sua implementação na Seção 4. Em seguida, teste de conceito é introduzido na Seção 5. As conclusões do trabalho são discutidas na Seção 6.

2 UPnP

De acordo com Sales [Sales et al. 2008], o UPnP é uma especificação que agrega a descoberta de dispositivos com a utilização de serviços, utilizando tecnologias consagradas da internet como HTTP, XML e SOAP. A sua funcionalidade é descrita por seis passos: endereçamento, descoberta, descrição, controle, eventos e apresentação.

Considerando que o endereçamento IP dos pontos de controle e dispositivos está devidamente correto, então a descoberta dos dispositivos é realizada. Durante o processo de descoberta, a descrição dos dispositivos é enviada, e o ponto de controle descobre suas informações. Os pontos de controle realizam o controle dos dispositivos através do envio de comandos. Os dispositivos se comunicam com os pontos de controle através de eventos. Finalmente após enviar e receber informações, o ponto de controle pode utilizar a apresentação para atualizar a interface com o usuário da forma desejada.

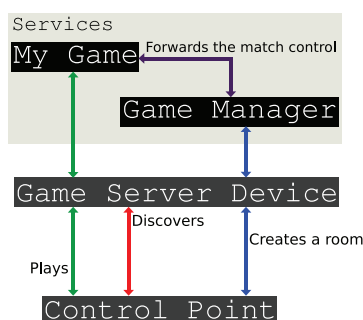


Figura 2: Esquema de funcionamento do BRisa Game Server (BRiGaS)

No caso do BRiGaS, um ponto de controle UPnP descobre um dispositivo *Game Server* (UPnP) na rede. Ao identificar seu tipo, o ponto de controle já pode conversar com o serviço *Game Manager*. Este último é encarregado de gerenciar os jogadores e as salas de jogos. Cada jogo neste servidor possui salas, que são criadas pelos próprios jogadores. As salas são ambientes virtuais onde os jogadores se encontram para começar uma partida. Então os pontos de controle se comunicam com o *Game Manager* para obter informações sobre salas existentes e se juntarem às mesmas ou criar novas salas. Ao começar uma partida, o *Game Manager* libera o controle da partida para o serviço do jogo em questão. Este fluxo é exemplificado na Figura 2.

2.1 BRisa

O BRisa é um arcabouço que implementa o protocolo e arquitetura UPnP, escrito na linguagem de programação Python². Inicialmente, utilizou-se uma abordagem que adotou a parte de áudio/vídeo do UPnP como foco principal. Atualmente, alcançou-se um estado de arcabouço geral para o UPnP. Além disso, possui certas características como API³ de alto nível para construir dispositivos UPnP e serviços, através da utilização de programação orientada a objeto, integração com Qt⁴ [Blanchette and Summerfield 2008], Gtk⁵, Glib2⁶ e Ecore⁷. Neste arcabouço, também estão implementadas facilidades para *login*, configuração, *multi-threading*, *networking* e possui ferramentas para trabalhar com UDP, HTTP, XML, IP e interfaces de rede.

3 Especificação do Game Server

O *Game Server* controla e possibilita a interação de dispositivos durante uma sessão, como por exemplo, criar um jogo de domínio entre quatro dispositivos. A especificação se divide em duas partes: O *Game Server Device*, que constitui o dispositivo propriamente dito, e o *Game Manager Service*, encarregado de gerenciar jogos e jogadores do servidor.

3.1 Game Server Device

No contexto de um servidor de jogos, este deve disponibilizar, ao menos, um jogo e também deve gerenciar como os jogadores podem reconhecer esses jogos. Tornando abstrato este fato para o UPnP, foi criada a seguinte temática: existe um dispositivo UPnP, denominado *Game Server*, e cada jogo dentro deste servidor é um serviço UPnP provido pelo mesmo. Isto quer dizer, que para cada jogo existente no servidor, existe um serviço relacionado, através do qual os pontos de controle se comunicarão, o que permitirá serem feitas as jogadas pelos clientes. Os serviços dos jogos não são descritos na especificação do *Game Server*, pois, cada jogo possui uma forma de comunicação, o que não permite criar uma

comunicação padrão para todos os tipos de jogos. Há somente uma única exigência para os serviços de jogos: eles devem possuir os UUIDs dos jogadores participantes. Serão dados mais detalhes na próxima seção. Como este servidor deve gerenciar os jogadores e o acesso dos mesmos aos jogos, foi criado o serviço UPnP *Game Manager*, encarregado de gerenciar os jogadores conectados ao servidor, mostrar quais jogos estão disponíveis e realizar a transição dos jogadores entre os jogos e suas salas. O XML que especifica o *Game Server Device* pode ser encontrado no site⁸ do projeto. Este XML possui somente o serviço *Game Manager*. Os serviços adicionais, os jogos, ficam a critério da implementação deste servidor. Um exemplo de um desses serviços de jogos é mostrado abaixo, explicado detalhadamente na Seção 5:

```
1<?xml version="1.0" encoding="utf-8"?>
2<root xmlns="urn:schemas-upnp-org:device-1-0">
3
4[...]
```

```
11<serviceList>
12  <service>
13    <serviceType>
14      urn:upnp-ic-ufal-br:service:GameManager:1
15    </serviceType>
16    <serviceId>GameManager</serviceId>
17    <SCPURL>/GameManager/scpd.xml</SCPURL>
18    <controlURL>/GameManager/control</controlURL>
19    <eventSubURL>/GameManager/eventSub</eventSubURL>
20    <presentationURL>
21      /GameManager/presentation
22    </presentationURL>
23  </service>
24  <service>
25    <serviceType>
26      urn:ic-ufal-br:service:Truco:1
27    </serviceType>
28    <serviceId>Truco</serviceId>
29    <SCPURL>/Truco/scpd.xml</SCPURL>
30    <controlURL>/Truco/control</controlURL>
31    <eventSubURL>/Truco/eventSub</eventSubURL>
32    <presentationURL>
33      /Truco/presentation
34    </presentationURL>
35  </service>
36</serviceList>
37
38[...]
```

Exemplo de descrição do jogo de truco

3.2 Game Manager Service

Como destacado anteriormente, este serviço é encarregado de gerenciar os jogos e os jogadores. Para o *Game Manager* funcionar corretamente, devem ser disponibilizados pelo *Game Server Device*, quais jogos estão disponíveis. A partir disso, este serviço criará uma estrutura, em que cada jogo possui salas, que permitirão aos jogadores interagirem. O *Game Manager Service* possui os seguintes métodos SOAP: *GetUUID*, *GetAvailableGames*, *GetAvailableRooms*, *CreateRoom*, *EnterRoom*, *LeaveRoom* e *GoodBye*. Também possui a *State Variable RoomUpdate* com a propriedade *Send Event*. O XML que o especifica pode ser encontrado no site⁹ do projeto.

O método *GetUUID* gera um novo UUID, utilizado para identificar o jogador dentro do servidor e nos jogos. Já o método *GetAvailableGames* retorna os jogos registrados no dispositivo. É importante destacar que o nome dos jogos são separados pelo caractere ':'. Exemplo:

"Truco:Domino:Xadrez"

O método SOAP *GetAvailableRooms* recebe o nome de um jogo como parâmetro, devendo retornar as variáveis *RoomsID*, *MaxPlayers* e *NumPlayers*, como especificadas no XML. As propriedades das salas são retornadas nessas três variáveis, onde cada posição de

²<http://www.python.org/>

³*Application Programming Interface*

⁴<http://www.qtsoftware.com/products>

⁵<http://www.gtk.org>

⁶<http://library.gnome.org/devel/glib/stable/>

⁷<http://wiki.enlightenment.org/index.php/Ecore>

⁸<http://launchpad.net/brigas/trunk/0.1/+download/example-game-server-device.xml>

⁹<http://launchpad.net/brigas/trunk/0.1/+download/example-game-manager-scpd.xml>

cada vetor, separado pelo caractere ':' refere-se a uma sala. Exemplo:

```
RoomsID: "0:1:3"
MaxPlayers: "3:2:4"
NumPlayers: "2:1:1"
```

Neste exemplo existem três salas: a primeira sala com o identificador 0, com número máximo de jogadores 3 e com número atual de jogadores 2. O mesmo vale para as outras duas salas. Em *CreateRoom* é recebido o UUID do jogador que criou a sala, o jogo que ele pretende jogar e a quantidade máxima de jogadores na sala. O serviço *Game Manager* deve criar em sua estrutura esta sala, adicionar o UUID dado a ela e então retornar o ID da nova sala criada.

No método *EnterRoom*, o serviço deve atualizar o jogo especificado como parâmetro *Game*, adicionando o valor de UUID à sala de *RoomID*. Esse método possui uma função muito importante no *Game Manager*, pois ele é encarregado de iniciar a partida. Quando a sala chegar ao número limite de jogadores, este serviço tem de informar ao jogo que ela está cheia, e portanto que partida deve começar. Ao iniciar a partida, são passados os UUIDs de todos os jogadores para este jogo. É retornado o valor 1 em caso de sucesso. O método *LeaveRoom* deve realizar a tarefa de retirar de sua estrutura de salas o jogador com o valor de UUID, do jogo de nome *Game* e da sala com o identificador *RoomID*. Retorna o valor 1 em caso de sucesso.

O método SOAP *GoodBye* é utilizado quando um jogador se desconecta do servidor. Então é enviado seu UUID para ser retirado pelo serviço *Game Manager* da lista de jogadores. Retorna o valor 1 em caso de sucesso.

Nos métodos *CreateRoom*, *EnterRoom* e *LeaveRoom* deve ser alterado o valor da *State Variable RoomUpdate* para que os pontos de controle tenham conhecimento de uma nova atualização na lista de salas dos jogos do servidor. O novo valor da variável deve seguir o seguinte padrão:

```
"Game:RoomID:NumPlayers"
```

4 Implementação do Game Server

Para implementar o *Game Server* que foi especificado na seção anterior, utilizou-se o arcabouço *BRisa*. Esta implementação foi denominada de *BRiGaS*, que possui dois componentes principais, o *BRisa Game Server* para o lado servidor e o *BRiGaS Control Point* para o lado cliente. O arcabouço *BRisa* foi criado de forma que a implementação do padrão UPnP seja simples, fácil e prática.

4.1 BRisa Game Server

Para o *BRisa Game Server* ser implementado, foram utilizadas as classes *Device* e *Service* do arcabouço. Com a classe *Device* facilmente cria-se um dispositivo UPnP, como é mostrado no exemplo a seguir a criação do *Game Server Device*:

```
1 [...]
2
3 self.device = \
4     Device (
5         'urn:upnp-ic-ufal-br:device:GameServer:1',
6         self._server_name,
7         force_listen_url=self._listen_url,
8         manufacturer='BRiGaS Team. UFAL students',
9         manufacturer_url=project_page,
10        model_description='An UPnP Game Server',
11        model_name = 'BRisa Game Server version 0.1',
12        model_number='0.1',
13        model_url=project_page,
14        serial_number='01'.rjust(7, '0'))
15
16 [...]
```

Implementação do dispositivo

Uma funcionalidade interessante do *BRisa Game Server* é o modo como ele reconhece os jogos disponíveis no servidor. Foi criada uma estrutura para que terceiros possam estender o *BRisa Game*

Server com a implementação de seus próprios jogos, e o dispositivo seja capaz de identificar automaticamente esses jogos, como se fossem *plugins*. Para que isso seja possível, o *BRisa Game Server* possui duas classes extensíveis, das quais todo novo jogo no servidor deve fazer a extensão de pelo menos uma dessas classes. Essas classes pertencem ao módulo *games* chamadas *BaseRoom* e *BaseGame*. A classe *BaseRoom* não é obrigatória de extensão, pois ela já possui as funcionalidades básicas de uma sala de jogo, mas às vezes pode ser necessário implementar algo específico. A classe *BaseGame* deve sempre ser estendida por um novo jogo no servidor, pois ela possui métodos usados pelo *Game Manager* que devem ser implementados, que são *start_new_match* e *add_room*. É importante ressaltar também que conforme a especificação, cada jogo no servidor é um novo serviço disponível no dispositivo. Então, os jogos implementados para o *BRisa Game Server* devem estender a classe *Service* do arcabouço *BRisa*.

```
17 [...]
18
19 class MyGame(BaseGame):
20
21     # Start new match
22     def start_new_match(self, room):
23         print "New Match!!!"
24
25     # Create new room
26     def add_room(self, uuid, max_players):
27         print "New Room created!"
28
29 [...]
```

Extensão dos métodos da classe BaseGame

Para que esses jogos sejam reconhecidos automaticamente pelo *BRisa Game Server* eles devem obedecer algumas regras. A primeira delas é que o módulo do jogo precisa estar dentro do módulo *games* do *BRisa Game Server*, e este módulo deve possuir uma variável chamada *service_name*, que deve ser o nome da classe criada para o serviço do jogo. É a partir dessa variável que a classe é instanciada dinamicamente. Uma forma de implementação pode ser a seguinte:

```
brisa_game_server/games/mygame/__init__.py:
```

```
30 from brisa_game_server.games.mygame.impl\
31 import MyGame
32 from brisa_game_server.games.mygame.impl\
33 import service_name
```

Inicialização do módulo do jogo

```
brisa_game_server/games/mygame/impl.py:
```

```
34 [...]
35
36 service_name = 'MyGame'
37
38 class MyGame(BaseGame, Service):
39     def __init__(self):
40         Service.__init__(self, service_name,
41                         service_type, '',
42                         scpd_xml_path)
43         BaseGame.__init__(self)
44
45 [...]
```

Implementação do serviço

4.2 BRiGaS Control Point

No lado cliente temos um encapsulamento da API de ponto de controle do *BRisa*, uma API facilitadora. O *BRiGaS Control Point* faz o acesso direto ao serviço *Game Manager*, abstraindo toda a lógica do UPnP para a aplicação cliente. A chamada dos métodos SOAP do serviço são encapsuladas, criando uma interface simples de utilização do *Game Manager*. Os eventos gerados pelo serviço *Game Manager* também são encapsulados e tratados internamente. Este também possui uma forma de comunicação com seus

utilizadores. O BRiGaS CP (*BRiGaS Control Point*) tem a possibilidade de inscrição para eventos, onde os interessados em saber das atualizações dos estados do mesmo podem ser notificados. A inscrição é feita passando-se uma função ou método que se deseja executar quando da ocorrência deste evento.

brisa_game_server/games/mygame/impl.py:

```

46 [...]
47
48 from brisa_game_server.control_points.game_manager \
49 import GameManagerCP
50
51 [...]
52
53 def fun():
54     print 'Hello!'
55
56 [...]
57
58 control_point = GameManagerCP()
59 control_point.event_subscribe('RoomUpdate', fun)
60
61 [...]

```

Exemplo de uso da API de eventos

Quando o evento *RoomUpdate* acontecer a função *fun* será executada e será escrito 'Hello!' na tela. Seguindo este modelo do *BRiGaS Control Point*, podem ser criados pontos de controle para qualquer jogo que o servidor possua. Desta forma, os clientes podem ser criados sem a utilização do protocolo UPnP diretamente, provendo uma abstração para que sejam criados clientes de forma fácil e ágil. Um estudo de caso é apresentado a seguir para ilustrar o uso do BRiGaS.

5 Estudo de caso: BRiGaS de Truco

A prova de conceito utilizada é um jogo de cartas originado na América do Sul chamado Truco, por conta de sua popularidade e simplicidade, o que facilita sua implementação. O Truco é jogado em duplas e somente parte das cartas de um baralho é utilizada. A principal característica do jogo é a competição por pontos, fazendo com que as duplas aumentem o valor de uma partida compelindo a dupla adversária a desistir, o que exige interação e negociação entre jogadores.

A idéia foi construir uma aplicação portátil para que os jogadores pudessem executar o Truco em qualquer sistema e arquitetura que, por sua vez, pudesse executar o arcabouço BRisa. A implementação SOA (*Service Oriented Architecture*) do serviço no *Game Server* envia notificações de acordo com a seguinte estratégia: quando uma certa variável definida no serviço é alterada, o serviço manda mensagens informando os estados específicos da mudança. A mensagem é formatada no padrão de *string SOAP* permitindo que aplicações do jogo em qualquer linguagem usem a informação, ou seja, a comunicação é definida usando definições de serviço para permitir a portabilidade da aplicação.

Nesse estudo de caso uma primeira aplicação cliente foi desenvolvida para dispositivos *Internet Tablet* da Nokia usando a plataforma *Maemo*¹⁰ por ser um sistema Linux. A linguagem Python foi utilizada juntamente com o arcabouço BRisa e a API *Edje*¹¹ foi utilizada para desenvolver a interface com o usuário, por prover bons efeitos visuais e por fácil uso na implementação. A interface do jogo de Truco é ilustrada na Figura 3.

6 Conclusões e Trabalhos Futuros

Neste artigo é descrita a criação de uma especificação para servidores de jogos utilizando o padrão UPnP. A partir desta especificação, foi implementado o BRiGaS (*BRisa Game Server*). Com a especificação, definimos um novo dispositivo (*UPnP device*), o *Game Server*, e um serviço, o *Game Manager*. Para provar



Figura 3: Interface do jogo de Truco

os conceitos dessa especificação, a ferramenta foi implementada utilizando o arcabouço BRisa para o padrão UPnP. Além disso, foi apresentada a implementação de um jogo de Truco como estudo de caso.

Com o desenvolvimento do trabalho, algumas necessidades e também possibilidades de aperfeiçoamento foram observados, inclusive sobre limitações do próprio padrão UPnP. Primeiramente, é necessária a criação de uma extensão para o protocolo, pois devido às limitações do mesmo com relação ao envio de eventos *multicast*, vários pacotes desnecessários são enviados pela rede. Sabe-se que é dessa forma que os pontos de controle recebem a notificação das mudanças de variáveis de estado. Ainda não há uma forma de envio de eventos *unicast*. Portanto, a nova extensão deve especificar uma forma de envio de eventos *unicast*, ou seja, fazer a comunicação entre um dispositivo e um ponto de controle específico.

Em relação ao próprio BRiGaS, há a necessidade da especificação de um esquema de segurança e criptografia no tráfego de algumas mensagens enviadas e recebidas pelo *Game Server* para dificultar a interceptação de informações dos jogos. Outro trabalho futuro é prover a identificação única dos jogadores para criar perfis no servidor, estabelecendo *ranking* e reputação. Uma solução é utilizar a extensão UPnP-UP (*UPnP User Profile*) [Sales et al. 2008] para prover essa funcionalidade.

Com a identificação do usuário através do UPnP-UP pode-se também criar um sistema de personalização do jogo, onde o jogador pode mudar suas configurações no servidor e esta personalização pode ser carregada para qualquer servidor de jogos que o jogador se conectar. Ainda com o UPnP-UP, pode ser feito um esquema de recomendação de jogos, oponentes e parceiros, devido ao armazenamento das preferências e histórico do usuário. Recomendar oponentes também é significativo, pois aumenta a competitividade no servidor. Os jogadores que estão no mesmo nível de habilidade poderão jogar entre si. Os parceiros em jogos que mais obtiveram vitórias podem ser recomendados para jogarem juntos novamente, já que teoricamente eles possuem um entrosamento maior.

Referências

- BLANCHETTE, J., AND SUMMERFIELD, M. 2008. *C++ GUI Programming with Qt 4*, 2 ed. Prentice Hall.
- GUEDES, A., SANTOS, D., DO NASCIMENTO, J., SALES, L., PERKUSICH, A., AND ALMEIDA, H. 2008. Set your multimedia application free with brisa framework: An open source upnp implementation for resource limited devices. In *Proceedings of the 5th IEEE Consumer Communications and Networking Conference*, 1257–1258.
- SALES, T. B. M., SALES, L. M., PEREIRA, M. F., OLIVEIRA, H. A., PERKUSICH, A., AND JUNIOR, M. A. S. 2008. Towards the upnp-up: Enabling user profile to support customized services in upnp networks. In *Proceedings of the Second International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, 206–211.
- WEISER, M. 1991. The Computer for the 21st Century. *Scientific American* 265, 3 (Sept.), 66–75.

¹⁰<http://maemo.org/>

¹¹<http://wiki.enlightenment.org/index.php/Edje>